МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБСИСТЕМА АДАПТИВНОЇ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ДЛЯ ФІНАНСОВИХ УСТАНОВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Дмитро ЗАВОРОТНІЙ

«___»_____ 2025 р.

Керівник ст. викладач _____ Іван БУРЛАЧЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

Факультет:	Комп’ютерних наук
Кафедра:	Інтелектуальних інформаційних систем
Рівень вищої освіти:	Перший (бакалаврський) рівень
Спеціальність:	122 Комп’ютерних наук
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____»_____2025 року

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Заворотнього Дмитра Олександровича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебсистема адаптивної автентифікації користувачів для фінансових установ».

Керівник роботи: Бурлаченко Іван Сергійович, ст. викладач.

Затверджено наказом по ЧНУ ім. Петра Могили від «18» листопада 2024 р. №321.

2. Строк представлення кваліфікаційної роботи «18» червня 2025 р.

3. Вхідні (початкові) дані до роботи: база даних з інформацією про ризики потенційних атак, камера смартфону з високою роздільною здатністю, телефонний номер з оплаченим тарифним планом та мережа Інтернет чи мобільний інтернет. Очікуваний результат: алгоритми адаптивної автентифікації та машинного навчання, адаптивний вебзастосунок, програмний код.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз адаптивних методів автентифікації користувачів у фінансових установах;

– дослідження методу класифікації пристроїв користувачів (UAS) з використанням моделі FastText для оцінки ризиків;

– проєктування архітектури вебсистеми адаптивної автентифікації відповідно до сучасних стандартів інформаційної безпеки;

– реалізація програмної частини вебсистеми, включаючи: клієнтську частину (Front-End) з використанням Vue.js та Socket.IO та серверну частину (Back-End) з використанням Nest.js та Spring Boot;

– створення та інтеграція бази даних для системи з використанням PostgreSQL;

– тестування та оцінка ефективності розробленої системи з точки зору безпеки та user experience.

5. Перелік графічного матеріалу: презентація.

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Дмитро ЗАВОРОТНІЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: «Вебсистема адаптивної автентифікації користувачів для фінансових установ»

	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд механізмів та існуючих адаптивних методів автентифікації користувачів для фінансових установ	31.01.2025	01.03.2025	Виконано
4	Огляд існуючих принципів функціонування алгоритмів машинного навчання у системах адаптивної автентифікації	02.03.2025	01.04.2025	Виконано
5	Проєктування вебсистеми адаптивної автентифікації	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	Виконано

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Дмитро ЗАВОРОТНІЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 402 ЧНУ ім. Петра Могили

Заворотнього Дмитра Олександровича

на тему: **«ВЕБСИСТЕМА АДАПТИВНОЇ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ДЛЯ ФІНАНСОВИХ УСТАНОВ»**

Актуальність даної роботи полягає у тому, що трансформація кібербезпеки фінансового сектору вимагає впровадження адаптивних рішень автентифікації, здатних аналізувати контекст операцій та динамічно реагувати на рівень загрози.

Об’єктом роботи є процеси автентифікації та управління персонального доступом до ресурсів фінансових установ для запобігання кіберзлочинності.

Предметом роботи є методи і моделі побудови адаптивної автентифікації користувачів фінансових установ.

Метою роботи є підвищення надійності вебсистеми автентифікації за рахунок адаптивності на основі використання моделей машинного навчання.

У результаті виконання роботи було підвищено надійність сервісу адаптивної автентифікації користувачів фінансових установ. Впровадження розробленого сервісу в українських фінансових установах дозволить знизити ризики фінансових втрат від кібератак та підвищити довіру клієнтів до електронних фінансових сервісів.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та додатків. У першому розділі описуються існуючі адаптивні методи автентифікації, аналізуються нормативні акти для впровадження механізмів автентифікації у фінансових установах. У другому розділі описується використання методу класифікації UAS з використанням моделі FastText. У третьому розділі описується проєктування вебсистеми адаптивної автентифікації. У четвертому розділі описується програмна реалізація, структура бази даних, тестування та інструкції користувача. У висновках проводиться аналіз роботи та отриманих результатів. Загальний обсяг роботи – 123 сторінок. Кваліфікаційна робота містить 42 рисунки, 5 таблиць, 33 джерел посилання.

Ключові слова: адаптивна автентифікація, кібербезпека, машинне навчання.

ABSTRACT

to the qualification work by the student of the group 402 of Petro Mohyla Black Sea National University

Zavorotnii Dmytro

«WEB-BASED ADAPTIVE USER AUTHENTICATION SYSTEM FOR FINANCIAL INSTITUTIONS»

The relevance of this research lies in the fact that the transformation of cybersecurity in the financial sector requires the implementation of adaptive authentication solutions capable of analyzing the context of operations and dynamically responding to the threat level.

The object of the research is the authentication processes and personal access management to financial institution resources for preventing cybercrime.

The subject of the research is the methods and models for building adaptive authentication of financial institution users.

The purpose of the work is to improve the reliability of the web authentication system through adaptability based on the use of machine learning models.

As a result of the work the reliability of the adaptive authentication service for financial institution users was improved. The implementation of the developed service in Ukrainian financial institutions will reduce the risks of financial losses from cyberattacks and increase customer trust in electronic financial services.

This work consists of four sections: the introduction, four sections, conclusions, and appendices. The first section describes existing adaptive authentication methods and analyzes regulatory acts for implementing authentication mechanisms in financial institutions. The second section describes the use of the UAS classification method using the FastText model. The third section describes the design of the adaptive authentication web system. The fourth section describes the software implementation, database structure, testing, and user instructions. The conclusions provide an analysis of the work and the obtained results. The overall scope of the work is 123 pages. Thesis contains 42 figures, 5 tables, and 33 reference in it.

Keywords: adaptive authentication, cybersecurity, machine learning.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ	–	Програмне забезпечення
ШІ	–	Штучний інтелект
2FA	–	Two-factor authentication
CBOW	–	Continuous Bag of Words
KBA	–	Knowledge-Based Authentication
ML	–	Machine Learning
MFA	–	Multi-factor authentication
PSD2	–	Payment Services Directive 2
SFA	–	Single-factor authentication
UAS	–	User-Agent string
UML	–	Unified Modeling Language

ВСТУП

В умовах діджиталізації фінансових послуг верифікація користувачів стала ключовим технологічним викликом. Трансформація кібербезпеки вимагає впровадження адаптивних рішень автентифікації, що реагують на контекст і рівень загрози. Фінансовий сектор залишається пріоритетною ціллю кібератак через доступ до активів. Понад 20% випадків несанкціонованого доступу відбуваються не з причин технічних вразливостей, а через викрадення автентифікаційних даних, що підкреслює необхідність систем виявлення поведінкових аномалій. Регуляторні вимоги, зокрема європейська директива PSD2 та українські нормативи, встановлюють обов’язковість посиленої верифікації при електронних транзакціях. Фінансові установи впроваджують багаторівневу автентифікацію, а адаптивні механізми дозволяють балансувати безпеку та зручність, таким чином більше половини клієнтів припиняють користуватися фінансовими платформами через складність автентифікації, оскільки вважають зручність взаємодії вирішальним фактором для вибору фінансового партнера. Розвиток біометричних технологій та алгоритмів штучного інтелекту розширює інструментарій автентифікації.

Проєктування та імплементація ефективної системи адаптивної автентифікації виступає не лише технологічним завданням, але й стратегічною конкурентною перевагою для фінансових установ в умовах цифрового розвитку галузі. Практичне значення роботи посилюється фундаментальними зрушеннями у моделях взаємодії споживачів із фінансовими сервісами, де довіра та безпека становлять основу тривалих клієнтських відносин. Особливої актуальності дана проблематика набуває в українському контексті трансформації фінансового сектору та його інтеграції до глобальних стандартів інформаційної безпеки в умовах підвищених геополітичних ризиків. Таким чином, розробка вітчизняних інноваційних рішень у сфері адаптивної автентифікації користувачів фінансових систем сприятиме не лише підвищенню конкурентоспроможності окремих установ, але й зміцненню стійкості фінансової системи держави загалом.

Актуальність роботи також зумовлена статистикою НБУ, згідно з якою шахрайські операції з платіжними картками в Україні за 2024 рік зросли на 27%, що підтверджує необхідність удосконалення механізмів верифікації. Впровадження євроінтеграційних процесів у фінансовому секторі вимагає відповідності українських фінансових установ вимогам європейських регуляторів щодо безпеки електронних транзакцій, зокрема директиві PSD2. Додатковим фактором актуальності є прискорена цифрова трансформація в умовах воєнного стану, що створює нові вектори кіберзагроз у фінансовому секторі України.

Аналіз сучасного стану проблеми демонструє суттєвий прогрес у розробці адаптивних методів автентифікації, однак виявляє й значні прогалини. Провідні дослідники галузі (Д. Бірн, К. Лі, М. Аллен) відзначають недостатню інтеграцію поведінкової біометрії з контекстними факторами автентифікації. Компанії-лідери ринку (наприклад, IBM Security) пропонують комплексні рішення, проте їхні системи часто не враховують специфіку фінансового сектору пострадянських країн. Також недоліками є відсутність універсальних стандартів оцінки ризиків при адаптивній автентифікації, що ускладнює імплементацію таких систем у регульованих галузях. Вітчизняні наукові розробки у цій сфері (О. Петренко, В. Ковальчук) концентруються на окремих аспектах біометричної верифікації, залишаючи поза увагою комплексний підхід до створення адаптивних екосистем автентифікації.

Дане дослідження корелює з науковими працями у сфері кібербезпеки фінансових систем (Д. Андерсон, І. Коваленко), біометричних технологій автентифікації (Т. Сміт, А. Мельник) та методів машинного навчання для виявлення аномалій у поведінці користувачів (Л. Чжан, О. Семенов). Значний внесок у розвиток теоретичних засад адаптивної автентифікації здійснили дослідження Массачусетського технологічного інституту та Київського політехнічного інституту щодо застосування нейромереж для аналізу поведінкових патернів. Запропонована система розвиває ці напрацювання, інтегруючи їх у

цілісне програмне рішення з урахуванням специфіки українського фінансового ринку та сучасних технологічних тенденцій.

Метою кваліфікаційної роботи бакалавра є підвищення надійності вебсистеми автентифікації за рахунок адаптивності на основі використання моделей машинного навчання.

Об’єктом роботи є процеси автентифікації та управління персонального доступом до ресурсів фінансових установ для запобігання кіберзлочинності.

Предметом роботи є методи і моделі побудови адаптивної автентифікації користувачів фінансових установ.

Для досягнення поставленої мети необхідно розв’язати наступні завдання: дослідити та описати механізми і нормативно-правову базу автентифікації у фінансових установах України, проаналізувати існуючі адаптивні методи автентифікації користувачів, визначити їх переваги і недоліки, опрацювати метод класифікації пристроїв користувачів (UAS) з використанням моделі FastText для оцінки ризиків, спроєктувати архітектуру вебсистеми відповідно до сучасних стандартів інформаційної безпеки, реалізувати її програмну частину, створити та інтегрувати базу даних для вебсистеми, провести тестування та оцінку ефективності, підготувати інструкції з використання.

Для виконання поставлених завдань будуть застосовані такі методи дослідження: методи системного аналізу, об’єктно-орієнтованого аналізу та проєктування, методи машинного навчання та інтелектуального аналізу даних, гібридний вид методології гнучкої розробки (agile methodology), методи оцінки користувацького досвіду (UX).

Практичне значення роботи полягає у підвищенні надійності вебсистеми адаптивної автентифікації користувачів фінансових установ та її подальше впровадження, що дозволить знизити ризики фінансових втрат від кібератак, а також підвищить довіру клієнтів до електронних фінансових сервісів.

Апробація результатів роботи. Міжнародна наукова конференція «Ольвійський форум – 2025: стратегії країн Причорноморського регіону в геополітичному просторі» [33].

1 АНАЛІЗ МЕТОДІВ АВТЕНТИФІКАЦІЇ У ФІНАНСОВИХ УСТАНОВАХ

1.1 Опис механізмів автентифікації у фінансових установах

Комплексний аналіз автентифікації у фінансовому секторі вимагає врахування особливостей галузі, її процесів та нормативного регулювання. Висока регульованість, критична важливість захисту даних та значні ризики безпеки обумовлюють підвищені вимоги до систем автентифікації.

Фундаментальним етапом забезпечення безпеки є встановлення особи клієнта через процеси первинної ідентифікації, автентифікації та авторизації. Надійність цих процедур критично важлива для безпеки фінансових операцій, таких як перекази коштів, платежі, операції з валютою та інвестиційна діяльність. Окрім початкової перевірки, важливим є постійний моніторинг транзакцій для своєчасного виявлення шахрайської активності та ефективного управління інцидентами безпеки.

Комплексний аналіз дозволяє визначити основні технологічні, операційні та бізнес-вимоги до сучасних систем автентифікації у фінансовому секторі. Серед технологічних вимог ключовими є багатофакторність, адаптивність, стійкість до атак, масштабованість і забезпечення високої зручності використання. Технологічні характеристики тісно пов’язані з особливостями експлуатації систем у специфічних умовах фінансових установ.

Розглядаючи операційні аспекти функціонування систем автентифікації, слід зазначити необхідність їх безперервної роботи в режимі 24/7/365 для забезпечення постійної доступності фінансових сервісів. Критичним є також забезпечення високої швидкості обробки транзакцій без шкоди для безпеки, оскільки затримки можуть призвести до фінансових втрат та репутаційних ризиків.

Ефективне функціонування систем автентифікації потребує регулярного оновлення захисних механізмів у відповідь на еволюцію кіберзагроз, що передбачає впровадження новітніх технологій безпеки та постійне удосконалення програмного забезпечення. Важливою складовою операційної діяльності є

систематичне навчання персоналу основам інформаційної безпеки, що мінімізує ризики, пов’язані з людським фактором. Додатково, фінансові установи повинні здійснювати періодичний аудит безпеки та тестування на проникнення для своєчасного виявлення та усунення вразливостей.

Бізнес-вимоги до систем автентифікації фокусуються на мінімізації частоти помилкових відмов для покращення клієнтського досвіду та зниженні ризику помилкових допусків, що може призвести до порушення безпеки. Крім того, пріоритетними є оптимізація швидкості обробки запитів, забезпечення інтеграції з існуючими системами та досягнення економічної ефективності як на етапі впровадження, так і протягом експлуатації.

Таким чином, дослідження технологічних, операційних та бізнес-аспектів автентифікації створює підґрунтя для подальшого аналізу нормативно-правових вимог, які суттєво впливають на проєктування і реалізацію відповідних систем у фінансовому секторі.

Розглядаючи технічні та операційні аспекти систем автентифікації, необхідно звернути увагу на регуляторне середовище, в якому функціонують фінансові установи. Проведене дослідження свідчить, що системи автентифікації у фінансових установах підпадають під дію численних нормативно-правових актів та стандартів, які встановлюють вимоги до безпеки, захисту даних та процесів верифікації особи. Розглянемо детальніше нормативно-правову базу у цій сфері.

Аналізуючи нормативно-правове регулювання у сфері автентифікації, слід насамперед звернути увагу на Закон України «Про захист персональних даних» №2297-VI, який визначає загальні принципи обробки персональної інформації громадян. Цей нормативний акт регулює правові відносини, що виникають у процесі обробки персональних даних, спрямовані на забезпечення їх захисту від неправомірного використання та доступу [1].

У контексті функціонування систем автентифікації положення закону зобов’язують операторів отримувати попередню згоду користувачів на обробку їхніх персональних даних, гарантувати належний рівень їх захисту від

несанкціонованого доступу, обмежувати обсяг зібраної інформації до мінімально необхідного для досягнення визначених цілей. Водночас користувачам має бути надано право на доступ до своїх персональних даних, а також можливість їх коригування або видалення у випадках, передбачених законодавством. Особливої уваги вимагає дотримання вимог щодо запровадження процедур повідомлення про інциденти витоку даних.

Слід підкреслити, що зазначений закон має універсальний характер і встановлює базові вимоги до захисту персональних даних, які у сфері фінансових послуг доповнюються спеціалізованими положеннями, зокрема нормами Закону України «Про платіжні послуги».

Постанова Правління Національного банку України № 58 від 3 травня 2023 року затверджує «Положення про автентифікацію та застосування посиленої автентифікації на платіжному ринку» [2]. Цей документ є ключовим нормативним актом, що регулює процеси автентифікації користувачів у фінансовому секторі України та встановлює вимоги до надавачів платіжних послуг. Положення було розроблено з метою підвищення безпеки електронних платежів та захисту користувачів платіжних послуг від шахрайства. Воно визначає основні принципи та вимоги до процедур автентифікації, які повинні застосовуватися надавачами платіжних послуг.

Постанова № 58 встановлює ключові нормативно-правові засади, що регулюють вимоги до автентифікації в сфері надання платіжних послуг. У документі детально визначено базові терміни, а саме: автентифікація, посилена автентифікація, елементи посиленої автентифікації та динамічний код автентифікації. Таке термінологічне визначення сприяє уніфікації розуміння зазначених понять серед учасників ринку, зменшує ризики неоднозначного тлумачення та забезпечує послідовність у впровадженні регуляторних вимог.

Основною вимогою Постанови є обов’язковість застосування автентифікації користувачів платіжних послуг під час отримання доступу до рахунків або ініціювання електронних платіжних операцій. У випадках підвищеного ризику або

здійснення дій через віддалені канали зв’язку, що можуть бути вразливими до шахрайства, передбачено застосування посиленої автентифікації. Зокрема, її необхідно впроваджувати при доступі до рахунку онлайн, під час проведення електронних платіжних операцій, а також у випадках, коли здійснюється взаємодія з платіжними системами через потенційно небезпечні канали.

Посилена автентифікація повинна ґрунтуватися на комбінації щонайменше двох елементів, що належать до різних категорій автентифікаційних засобів: *знання* (інформація, відома лише користувачу, наприклад, пароль або PIN-код), *володіння* (пристрій або об’єкт, яким користується виключно користувач, як-от смартфон або токен) та *притаманність* (індивідуальні біометричні характеристики – наприклад, відбиток пальця або зображення обличчя). Такий підхід підвищує рівень захисту персональних даних та ускладнює несанкціонований доступ.

Крім того, Постанова вимагає, щоб при ініціюванні електронних платіжних операцій посилена автентифікація включала динамічне зв’язування операції з конкретною сумою та отримувачем коштів. Це положення має на меті забезпечити додатковий рівень захисту, який дозволяє підтвердити легітимність кожної окремої транзакції.

Документ також передбачає певні винятки, за яких посилену автентифікацію можна не застосовувати. До таких винятків належать, зокрема, операції з невеликою сумою, транзакції з низьким рівнем ризику, а також регулярні платежі, що здійснюються на підставі попереднього погодження користувача.

Постанова акцентує увагу на необхідності забезпечення конфіденційності та цілісності персоналізованих облікових даних користувачів. Надавачі платіжних послуг повинні впроваджувати відповідні технічні та організаційні заходи, що гарантують захист таких даних від несанкціонованого доступу або втручання.

На завершення, регулятор чітко визначає відповідальність надавачів платіжних послуг за дотримання встановлених вимог. Вони зобов’язані впроваджувати належні заходи безпеки для захисту як фінансових операцій, так і персональних даних користувачів.

Регулювання процесів автентифікації у фінансових установах України здійснюється не лише Постановою Правління Національного банку України від 5 травня 2023 року № 58, але також рядом інших нормативно-правових актів. Кожен з них має свої особливості, що визначають підходи до автентифікації та інформаційної безпеки. Постанова № 58 є логічним продовженням та оновленням регулювання в частині автентифікації. Вона конкретизує вимоги щодо сильної автентифікації користувачів у банківській системі України відповідно до найкращих європейських практик, визначаючи нові стандарти автентифікації для проведення електронних операцій, враховуючи принципи PSD2 і сучасні ризики. На відміну від інших НПА, вона чітко структурує випадки застосування сильної автентифікації, виключення з неї, а також встановлює критерії оцінки ефективності автентифікаційних механізмів.

Закон України «Про платіжні послуги» від 30 червня 2021 року № 1591-IX. Цей закон закладає фундаментальні принципи безпеки у сфері надання платіжних послуг, акцентуючи увагу на необхідності застосування заходів сильної клієнтської автентифікації. Вимоги щодо автентифікації охоплюють встановлення особи користувача шляхом використання щонайменше двох елементів з різних категорій: знання (пароль), володіння (пристрій) та притаманної характеристики (біометричні дані). Закон чітко інтегрує європейські стандарти (зокрема, PSD2) у національне регулювання, сприяючи підвищенню довіри до платіжних систем. Водночас його загальний характер не забезпечує детального розкриття конкретних технічних механізмів автентифікації, що залишає простір для неоднозначного тлумачення на практиці [3].

Постанова Правління НБУ № 95 від 28 вересня 2017 року «Про затвердження Положення про організацію заходів із забезпечення інформаційної безпеки в банківській системі України». Положення встановлює комплексну систему заходів з інформаційної безпеки в банківському секторі. У контексті автентифікації акцент зроблено на впровадженні багаторівневої автентифікації, регулярному оновленні механізмів контролю доступу та обов’язковій ідентифікації користувачів у

банківських інформаційних системах. Структурованість вимог дозволяє банкам формувати системи безпеки на основі ризик-орієнтованого підходу, але положення певною мірою застаріле через динамічний розвиток кіберзагроз, що зумовлює необхідність постійного оновлення вимог [4].

Постанова Правління НБУ № 122 від 19 вересня 2019 року «Про затвердження Положення про кіберзахист та інформаційну безпеку в платіжних системах та системах розрахунків». Цей документ деталізує вимоги до кіберзахисту саме у сфері платіжних систем. Зокрема, йдеться про обов'язкове використання методів сильної клієнтської автентифікації при проведенні операцій, а також про впровадження політик управління ризиками безпеки інформації. Дана постанова робить фокус на платіжних системах, що дозволяє краще адаптувати безпекові заходи до специфіки платіжної інфраструктури. Але зазначені у документі вимоги можуть бути надмірно загальними для певних невеликих операторів платіжних послуг, що ускладнює їхню імплементацію.

Дослідження міжнародного регуляторного середовища у сфері автентифікації користувачів фінансових послуг є важливим етапом аналізу предметної області, оскільки ці стандарти часто є орієнтирами для розвитку національного законодавства та внутрішніх політик фінансових установ [5].

Одним із ключових міжнародних стандартів захисту платіжної інформації є PCI DSS (Payment Card Industry Data Security Standard), який встановлює комплекс вимог до систем, що обробляють дані платіжних карток. Зокрема, стандарт передбачає побудову і підтримку захищених мереж і систем, захист даних власників карток, управління вразливостями, суворий контроль доступу, регулярний моніторинг і тестування мереж, а також підтримку політик інформаційної безпеки. У контексті автентифікації PCI DSS визначає необхідність застосування сильних методів автентифікації, забезпечення унікальності облікових записів, впровадження двофакторної автентифікації для віддаленого доступу та автоматичне блокування облікових записів після певної кількості невдалих спроб

входу. Вимоги PCI DSS нерідко доповнюються положеннями інших міжнародних стандартів, зокрема серії ISO/IEC [6].

Серед міжнародних стандартів у сфері інформаційної безпеки особливе місце займає ISO/IEC 27001, який визначає вимоги до побудови, впровадження та постійного вдосконалення систем управління інформаційною безпекою (СУІБ). Стандарт передбачає системний підхід до захисту конфіденційної інформації, оцінку ризиків із подальшим впровадженням контрольних заходів, регулярний перегляд і вдосконалення безпекових процесів, а також формалізацію політик і процедур. У сфері автентифікації ISO/IEC 27001 акцентує увагу на необхідності формалізації політик контролю доступу, регламентуванні процесів реєстрації та скасування прав доступу користувачів, обмеженні привілейованого доступу, періодичному перегляді прав доступу та використанні надійних механізмів автентифікації [7].

Суттєвий вплив на розвиток вимог до систем автентифікації у фінансовому секторі Європи справила також Директива ЄС PSD2 (Payment Services Directive 2). Хоча її положення не є обов'язковими для України, процес гармонізації національного законодавства з правом ЄС обумовлює їхню актуальність. PSD2 вимагає впровадження сильної автентифікації клієнтів (SCA) під час проведення платіжних операцій, встановлює підвищені вимоги до безпеки електронних платежів та регламентує доступ третіх сторін до платіжних рахунків у межах концепції Open Banking [8].

Отже, проаналізувавши системи автентифікації у фінансовому секторі, варто зазначити складність та багатогранність даного процесу, що вимагає ретельного балансування між високим рівнем безпеки та зручністю для користувачів. Системи автентифікації повинні відповідати вимогам національних та міжнародних стандартів, зокрема таким як PCI DSS, ISO/IEC 27001 і вимогам Директиви PSD2, що підкреслює прагнення інтегрувати кращі світові практики у національну регуляторну систему. З іншого боку, регуляторний контекст є ключовим елементом у забезпеченні відповідності автентифікаційних систем міжнародним стандартам

безпеки. Постанова НБУ № 58 від 03.05.2023 року, яка орієнтована на вимоги Директиви PSD2, підтверджує прагнення до інтеграції міжнародних стандартів, але для забезпечення ефективності цих стандартів необхідно постійно переглядати нормативно-правову базу з урахуванням нових технологічних викликів.

Таким чином, для успішної реалізації систем автентифікації в умовах швидко змінюваного цифрового середовища важливо забезпечити їх гнучкість та адаптивність до нових кіберзагроз. Для цього необхідно як удосконалювати технологічні рішення, так і регулярно оновлювати нормативну базу, щоб підтримувати баланс між високим рівнем безпеки та зручністю використання для кінцевих користувачів.

1.2 Огляд існуючих адаптивних методів автентифікації

Автентифікація – це термін, який стосується процесу доведення справжності певного факту чи документа (рис.1.1). Зазвичай асоціюється з підтвердженням особистості користувача. Зазвичай користувач підтверджує свою особу, надаючи свої облікові дані, тобто узгоджену частину інформації, якою обмінюються користувач і система [9].

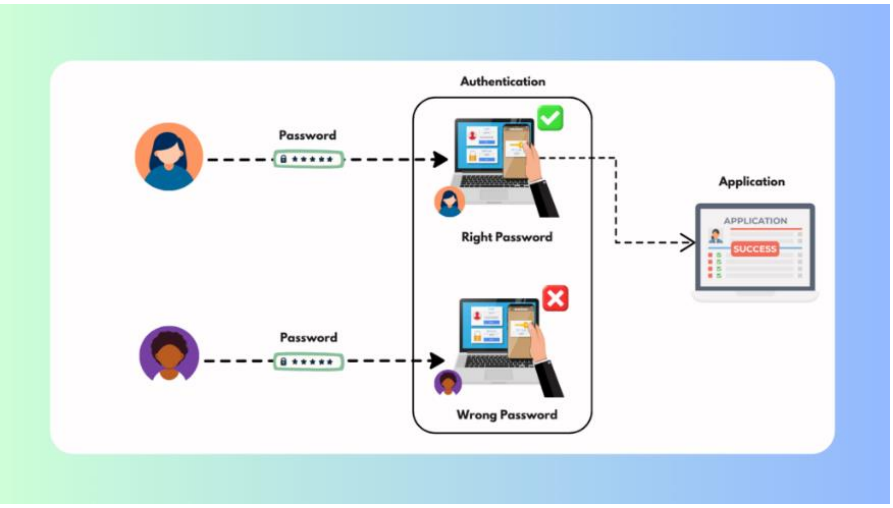


Рисунок 1.1 – Процес автентифікації

Сучасні системи інформаційної безпеки виокремлюють декілька основних видів автентифікації, класифікація яких базується на використовуваних факторах, а саме: однофакторна, двофакторна, багатофакторна.

Однофакторна автентифікація (SFA) передбачає верифікацію користувача на основі єдиного параметра, найчастіше представленого парольним захистом. Даний метод характеризується відносною простотою імплементації, проте демонструє вразливість до низки атак, включаючи методи соціальної інженерії та брутфорс.

Двофакторна автентифікація (2FA) поєднує два незалежні фактори верифікації, що суттєво підвищує рівень захисту системи. Типова реалізація включає комбінацію пароля та одноразового коду, генерованого апаратним токеном або програмним застосунком.

Багатофакторна (мультифакторна, MFA) автентифікація розширює двофакторний підхід, залучаючи три або більше факторів, що належать до різних категорій. Це максимізує ентропію системи автентифікації та мінімізує ймовірність несанкціонованого доступу [10].

Питання безпеки доступу до даних та сервісів є критично важливим у фінансовому секторі. У відповідь на існуючі кіберзагрози з’явилась концепція адаптивної автентифікації, яка забезпечує динамічну зміну рівня перевірки особи користувача залежно від різних факторів ризику.

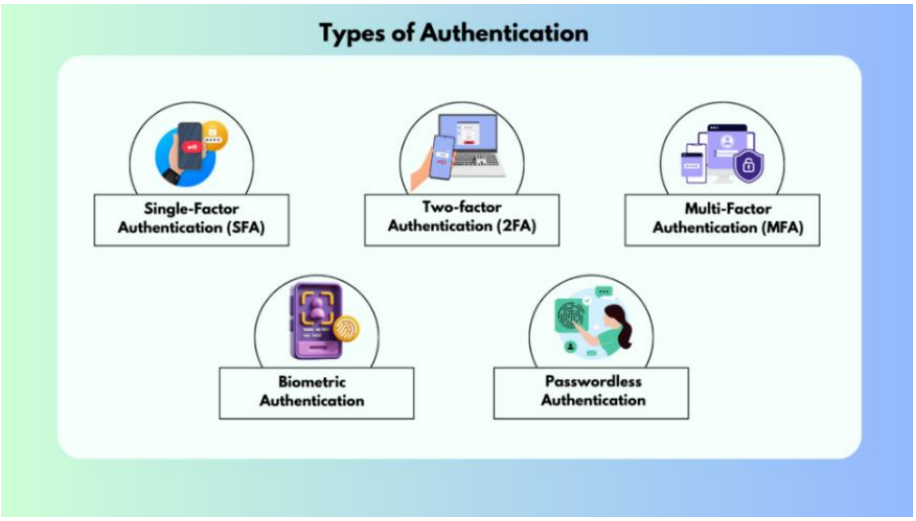


Рисунок 1.2 – Типи методів автентифікації

Адаптивна автентифікація – це динамічний процес перевірки особи користувача, що змінює свої параметри та вимоги залежно від оцінюваного рівня ризику конкретної операції. Система адаптивної автентифікації аналізує різноманітні фактори, включаючи поведінкові патерни користувача, геолокацію, час доступу, тип пристрою та характер запитуваної операції, для визначення необхідного рівня захисту. Концептуальна відмінність адаптивної автентифікації від традиційних методів полягає у переході від статичного до динамічного підходу при визначенні вимог до перевірки особи. Замість застосування однакових процедур для всіх користувачів і операцій, адаптивна система індивідуалізує процес автентифікації на основі контекстуальних даних та аналізу ризиків [11].

Основними методами адаптивної автентифікації, які наразі успішно використовуються у фінансовому секторі, є: поведінкова біометрія, контекстуальна автентифікація, біометрична автентифікація, аналіз аномалій та машинне навчання (ML), Автентифікація на основі знань (KBA).

Поведінкова біометрія аналізує унікальні характеристики поведінки користувача під час взаємодії з пристроєм чи додатком. Динаміка натискання клавіш дає змогу вимірювати часові інтервали між натисканнями клавіш та тривалість утримання клавіш. Кожен користувач має унікальний «почерк» друкування. Патерни руху мишею дозволяють аналізувати характеристики руху курсора, швидкості, прискорення та траєкторії. Також за допомогою динаміки дотиків до екрану можна проаналізувати силу, швидкість, положення та інші характеристики взаємодії з сенсорним екраном. Гіроскопічні патерни тримання пристрою використовують для аналізу того, як користувач тримає та маніпулює мобільним пристроєм. Ключова перевага поведінкової біометрії полягає в тому, що вона здатна безперервно верифікувати особу користувача протягом усієї сесії без додаткових дій з його боку (так звана «пасивна автентифікація») [12].

Контекстуальна автентифікація аналізує сукупність обставин, у яких відбувається спроба доступу. Визначається геолокація для перевірки місця розташування, з якого здійснюється доступ. Система відстежує аномальні

географічні переміщення, наприклад, якщо користувач увійшов у систему з України, а через годину намагається отримати доступ з іншої країни. Також аналізується часу доступу, щоб здійснити оцінку того, чи відповідає спроба доступу звичайному часовому шаблону користувача. До даного типу автентифікації також відноситься процес ідентифікації пристрою для того, щоб перевірити перелік пристроїв, які зазвичай використовує користувач. Аналіз мережі застосовується для оцінки безпечності мережі, через яку здійснюється доступ чи це публічний Wi-Fi або корпоративна мережа тощо. Контекстуальна автентифікація ефективно виявляє аномалії у звичних патернах доступу і може активувати додаткові рівні перевірки при виявленні потенційно ризикованих ситуацій.

Біометрична автентифікація за фізичними характеристиками включає: відбитки пальців, розпізнавання обличчя, сканування сітківки або райдужної оболонки ока, розпізнавання голосу. Розпізнавання відбитків пальців є одним із найпоширеніших методів біометричної автентифікації, особливо в мобільних пристроях. Під час розпізнавання обличчя відбувається аналіз геометрії обличчя, що також широкого використовується у сучасних смартфонах. Що стосується сканування сітківки або райдужної оболонки ока, то це є високонадійним методом біометричної перевірки. За допомогою розпізнавання голосу аналізується унікальність характеристик голосу користувача. Біометричні методи забезпечують високий рівень захисту, оскільки біологічні характеристики складно підробити, проте вони також вимагають спеціального обладнання та програмного забезпечення. Але дедалі частіше зловмисники використовують ШІ для підробки деяких біологічних характеристик особи.

Сучасні системи адаптивної автентифікації активно використовують алгоритми машинного навчання для виявлення підозрілої активності: виявлення аномалій (алгоритми для визначення відхилення від звичайної поведінки користувача), поведінкове профілювання (створення та постійне оновлення профілю типової поведінки користувача), прогнозування ризиків (оцінка

ймовірності шахрайських операцій на основі історичних даних), класифікація транзакцій (категоризація операцій за рівнем ризику з використанням різних алгоритмів класифікації). Ці методи дозволяють системам адаптивної автентифікації постійно вдосконалюватися, адаптуючись до нових загроз та патернів поведінки користувачів [13].

Автентифікація на основі знань використовує інформацію, якою володіє тільки справжній користувач. Статичне KBA застосовує заздалегідь визначені секретні питання та відповіді, а динамічне KBA ставить питання, що генеруються в реальному часі на основі даних про користувача з різних джерел. Також широко використовується управління знаннями про користувача: збирається та аналізується інформація про користувача для формування профілю знань.

Для ефективного впровадження системи адаптивної автентифікації важливо розуміти переваги та обмеження кожного методу (табл. 1.1).

Таблиця 1.1 – Порівняння методів адаптивної автентифікації

Метод автентифікації	Рівень безпеки	Зручність користування	Вартість впровадження	Точність розпізнавання	Імовірність обходу
Поведінкова біометрія	90–95%	Висока (пасивна автентифікація, без активних дій користувача)	$10 000–$50 000	Середня-висока	Залежить від змін у поведінці користувача
Контекстуальна автентифікація	70–85%	Висока (не потребує дій користувача)	$1 000–$5 000	Середня	Можлива підміна геолокації, IP-адреси
Фізична біометрія	95–99%	Середня (потрібна активна участь користувача)	$50 000+	Висока	Труднощі зі створенням фальшивок
Машинне навчання та аналіз аномалій	90–97%	Висока (пасивний моніторинг активностей)	$20 000–$100 000+	Висока	Залежно від складності атак
Автентифікація на основі знань	70–80%	Низька (потребує запам’ятовування та ручного введення даних)	Мінімальні витрати на реалізацію	Середня	Фішинг, підбір, соціальна інженерія

Адаптивна автентифікація є важливим елементом сучасних систем безпеки, оскільки дозволяє динамічно оцінювати ризики на основі поведінки та контексту користувача. Проведене порівняння основних методів адаптивної автентифікації (табл. 1.1) свідчить про наявність істотних відмінностей між ними за рівнем безпеки, зручністю використання, вартістю впровадження, точністю та імовірністю обходу.

Найвищий рівень безпеки забезпечують методи фізичної біометрії (відбитки пальців, розпізнавання обличчя) та рішення на основі машинного навчання й аналізу аномалій. Зокрема, фізична біометрія демонструє точність розпізнавання понад 95% при дуже низькому рівні ймовірності обходу (<1%), однак потребує значних капіталовкладень для впровадження і активної участі користувача.

Поведінкова біометрія і машинне навчання також забезпечують високу безпеку та зручність завдяки пасивному характеру автентифікації, що зменшує навантаження на користувача. Проте поведінкові методи є більш вразливими до змін у звичках користувачів та коливань у точності.

Контекстуальна автентифікація, що базується на даних про середовище користувача (геолокація, тип пристрою, IP-адреса), має помірний рівень безпеки та низькі витрати на інтеграцію. Водночас її вразливість до підміни контекстних даних обмежує надійність цього підходу.

Найменш надійною залишається автентифікація на основі знань (паролі, ПІН-коди), яка характеризується середнім рівнем точності, високою ймовірністю обходу та низькою зручністю користування. Незважаючи на незначну вартість впровадження, цей метод вважається застарілим у сучасних системах високої безпеки.

Таким чином, для забезпечення найвищого рівня захисту рекомендується використовувати багаторівневі системи автентифікації, що комбінують фізичну або поведінкову біометрію з машинним навчанням та контекстною аналітикою [14].

У фінансовому секторі широко використовуються комерційні рішення адаптивної автентифікації від провідних вендорів. Кожна з компаній має функції, які відрізняють її від конкурентів, тому розглянемо деякі з рішень які використовують компанії у фінансовому секторі. IBM Security Verify (раніше IBM Security Access Manager) пропонує комплексне рішення для адаптивної автентифікації з використанням машинного навчання для оцінки ризиків та підтримкою широкого спектру методів автентифікації. Microsoft Azure Active Directory включає функціонал умовного доступу (Conditional Access), який дозволяє налаштовувати політики автентифікації на основі різних сигналів ризику. Okta Adaptive Multi-Factor Authentication забезпечує динамічну багатофакторну автентифікацію з урахуванням контекстуальних факторів. RSA SecurID Suite пропонує рішення адаптивної автентифікації з акцентом на аналізі ризиків та бізнес-контексті.

Багато великих фінансових установ розробляють власні системи адаптивної автентифікації, адаптовані до їхніх конкретних потреб. Наприклад, Bank of America впровадила систему адаптивної автентифікації, яка аналізує понад 300 факторів для оцінки ризику та вибору методу автентифікації. JP Morgan Chase використовує поведінкову біометрію та аналіз аномалій для безперервної автентифікації користувачів онлайн-банкінгу. Barclays застосовує голосову біометрію в поєднанні з аналізом контексту для автентифікації клієнтів телефонного банкінгу.

Впровадження методів адаптивної автентифікації у фінансових установах має ряд суттєвих переваг, але також пов’язане з певними викликами та обмеженнями. *Перевагами є:* підвищений рівень безпеки, покращений користувацький досвід, відповідність регуляторним вимогам, операційна ефективність, масштабованість та гнучкість. *До недоліків та викликів* можна віднести: складність впровадження та налаштування, потреба в якісних даних, складність інтерпретації рішень системи, відповідність регуляторним вимогам, ризики помилкових спрацювань, проблеми приватності та етики.

Отже, методи адаптивної автентифікації забезпечують комплексний підхід до верифікації користувачів, який динамічно регулює рівень та методи перевірки на основі аналізу поведінкових патернів, контексту сесії та оцінки ризиків. На відміну від традиційних статичних методів, адаптивна автентифікація застосовує різні рівні перевірки залежно від поточної ситуації, забезпечуючи оптимальний баланс між безпекою та зручністю користування системою. Комбінуючи різні методи автентифікації та динамічно регулюючи рівень перевірки залежно від оцінки ризику, адаптивні системи здатні ефективно протидіяти сучасним кіберзагрозам. Успішне впровадження адаптивної автентифікації у фінансових установах вимагає комплексного підходу, що враховує технічні, нормативні та соціальні аспекти. Важливо забезпечити інтеграцію з існуючими системами, відповідність регуляторним вимогам та прийняття користувачами. Подальший розвиток адаптивної автентифікації буде спрямований на підвищення точності оцінки ризиків, розширення спектру аналізованих факторів та забезпечення безперервної верифікації користувачів. Інтеграція з технологіями штучного інтелекту та машинного навчання відкриває нові можливості для створення більш ефективних систем захисту фінансових установ.

1.3 Постановка задач для розробки вебсистеми адаптивної автентифікації користувачів для фінансових установ

Актуальність роботи. Трансформація кібербезпеки фінансового сектору вимагає впровадження адаптивних рішень автентифікації, здатних аналізувати контекст операцій та динамічно реагувати на рівень загрози. Актуальність даної проблеми підтверджується комплексом взаємопов’язаних факторів:

- вразливість фінансового сектору до кібератак;
- посилення регуляторних вимог;
- баланс між безпекою та користувацьким досвідом;
- проблеми багатоканальності;
- технологічний прогрес у сфері біометрії та штучного інтелекту;

– загострення конкуренції на фінансовому ринку.

Таким чином, розробка та впровадження ефективної системи адаптивної автентифікації є не лише технічним завданням, але й стратегічним інструментом конкурентної боротьби для українських фінансових установ в умовах цифрової трансформації галузі. Створення вітчизняних інноваційних рішень у сфері адаптивної автентифікації користувачів фінансових сервісів сприятиме як підвищенню конкурентоспроможності окремих установ, так і зміцненню загальної стійкості фінансової системи держави через мінімізацію ризиків кібератак та шахрайства.

Метою кваліфікаційної роботи бакалавра є підвищення надійності вебсистеми автентифікації за рахунок адаптивності на основі використання моделей машинного навчання.

Об’єктом роботи є процеси автентифікації та управління персонального доступом до ресурсів фінансових установ для запобігання кіберзлочинності.

Предметом роботи є методи і моделі побудови адаптивної автентифікації користувачів фінансових установ.

Для досягнення поставленої мети необхідно розв’язати наступні **завдання:**

– дослідити та описати механізми і нормативно-правову базу автентифікації у фінансових установах України та міжнародний досвід;

– проаналізувати існуючі адаптивні методи автентифікації користувачів у фінансових установах та визначити їх переваги і недоліки;

– дослідити механізми нейронних мереж для виявлення та протидії кібератакам у контексті автентифікації;

– проаналізувати метод класифікації пристроїв користувачів (UAS) з використанням моделі FastText для оцінки ризиків;

– спроєктувати архітектуру вебсистеми адаптивної автентифікації відповідно до сучасних стандартів інформаційної безпеки;

– реалізувати програмну частину вебсистеми, включаючи: клієнтську частину (Front-End) з використанням Vue.js та Socket.IO та серверну частину (Back-End) з використанням Nest.js та Spring Boot;

– створити та інтегрувати базу даних для системи з використанням PostgreSQL;

– провести тестування та оцінку ефективності розробленої системи з точки зору безпеки та user experience;

– підготувати документацію для користувачів системи, включаючи інструкції з використання.

Для виконання поставлених завдань будуть застосовані такі методи дослідження:

– методи системного аналізу для дослідження існуючих підходів до автентифікації;

– методи об’єктно-орієнтованого аналізу та проєктування;

– методи машинного навчання та інтелектуального аналізу даних;

– методи забезпечення надійності програмного забезпечення та інформаційної безпеки;

– гібридний вид методології гнучкої розробки (agile methodology);

– методи оцінки користувацького досвіду (UX).

Практичне значення роботи полягає у підвищенні надійності сервісу адаптивної автентифікації користувачів, що забезпечує вирішення проблем кібербезпеки та відповідає нормативним вимогам НБУ. Впровадження розробленого сервісу в українських фінансових установах дозволить знизити ризики фінансових втрат від кібератак та підвищити довіру клієнтів до електронних фінансових сервісів.

Висновки до розділу 1

Отже, у першому розділі описано механізми автентифікації у фінансових установах, проведено огляд існуючих адаптивних методів автентифікації,

розглянуто їх типи, переваги та недоліки кожного методу, досліджено технологічні, операційні та бізнес-аспекти автентифікації, проаналізовано нормативно-правові вимоги, які впливають на проєктування і реалізацію відповідних систем у фінансовому секторі.

Регуляторна база формує вимоги до систем автентифікації, які мають постійно оновлюватись для протидії новим загрозам, включаючи атаки з використанням штучного інтелекту. Системи автентифікації повинні відповідати вимогам національних та міжнародних стандартів, зокрема таким як PCI DSS, ISO/IEC 27001 і вимогам Директиви PSD2, що підкреслює прагнення інтегрувати кращі світові практики у національну регуляторну систему. Для досягнення максимальної ефективності необхідним є регулярний перегляд і оновлення як нормативної бази, так і технологічних рішень з урахуванням розвитку цифрових технологій та еволюції кіберзагроз.

Проведене порівняння основних методів адаптивної автентифікації свідчить про наявність істотних відмінностей між ними за рівнем безпеки, зручністю використання, вартістю впровадження, точністю та імовірністю обходу. Для забезпечення найвищого рівня захисту рекомендується використовувати багаторівневі системи автентифікації, що комбінують фізичну або поведінкову біометрію з машинним навчанням та контекстною аналітикою.

Також було розглянуто комерційні рішення адаптивної автентифікації від провідних вендорів, які широко використовуються у фінансовому секторі, а саме: IBM Security Verify, Microsoft Azure Active Directory, Okta Adaptive Multi-Factor Authentication, RSA SecurID Suite. Багато великих фінансових установ розробляють власні системи адаптивної автентифікації, адаптовані до їхніх конкретних потреб, а саме такі компанії: Bank of America, JP Morgan Chase, Barclays.

Таким чином, методи адаптивної автентифікації забезпечують комплексний підхід до верифікації користувачів, який динамічно регулює рівень та методи перевірки на основі аналізу поведінкових патернів, контексту сесії та оцінки ризиків. На відміну від традиційних статичних методів, адаптивна автентифікація

забезпечує оптимальний баланс між безпекою та зручністю користування системою. Успішне впровадження адаптивної автентифікації у фінансових установах вимагає комплексного підходу, що враховує технічні, нормативні та соціальні аспекти. Важливо забезпечити інтеграцію з існуючими системами, відповідність регуляторним вимогам та прийняття користувачами. Подальший розвиток адаптивної автентифікації буде спрямований на підвищення точності оцінки ризиків, розширення спектру аналізованих факторів та забезпечення безперервної верифікації користувачів. Інтеграція з технологіями штучного інтелекту та машинного навчання відкриває нові можливості для створення більш ефективних систем захисту фінансових установ.

2 МЕТОД КЛАСИФІКАЦІЇ UAS З ВИКОРИСТАННЯМ МОДЕЛІ FASTTEXT

2.1 Теоретичні основи та принципи функціонування алгоритмів машинного навчання в системах адаптивної автентифікації

Машинне навчання (МН) являє собою потужний інструмент для створення адаптивних систем автентифікації у фінансових вебсистемах. Основна перевага використання алгоритмів МН полягає в їхній здатності аналізувати велику кількість параметрів користувачів та виявляти приховані закономірності, які складно ідентифікувати традиційними методами.

Сучасні алгоритми машинного навчання дозволяють створювати системи, що адаптуються до поведінки користувачів і забезпечують багаторівневу автентифікацію. У контексті реалізації адаптивної автентифікації у фінансовому секторі машинне навчання відіграє важливу роль у підвищенні ефективності захисту користувацьких даних та фінансових транзакцій.

Застосування алгоритмів машинного навчання дає змогу розв’язувати низку критично важливих задач. Зокрема, системи класифікують користувацькі сесії, розрізняючи між легітимною активністю та потенційно небезпечними діями, що можуть свідчити про спробу шахрайства. Крім того, моделі прогнозування ризику дозволяють оцінити ймовірність того, що конкретна операція є підозрілою, що сприяє проактивному реагуванню на загрози.

Також машинне навчання забезпечує вибір найбільш доцільного методу автентифікації з урахуванням контексту дій користувача, зокрема місцезнаходження, типу пристрою чи характеру операції [15]. Одним із ключових аспектів є виявлення аномальної поведінки, яка може сигналізувати про компрометацію облікового запису або інші загрози інформаційній безпеці.

Робота алгоритмів машинного навчання в системах адаптивної автентифікації у фінансових установах ґрунтується на поетапному процесі обробки, аналізу та інтерпретації великого обсягу даних про поведінку

користувачів. Такий підхід забезпечує високий рівень персоналізованого контролю доступу та дозволяє в режимі реального часу виявляти потенційні загрози інформаційній безпеці.

На першому етапі – збору даних, система акумулює інформацію з множини джерел. Цей процес охоплює як традиційні, так і поведінкові характеристики користувачів. Наприклад, до біометричних показників відносяться темп і ритм введення тексту на клавіатурі, траєкторія та швидкість руху миші, що в сукупності створює унікальний «цифровий відбиток» кожного користувача.

Паралельно враховуються геолокаційні дані (наприклад, з яких країн чи міст найчастіше відбувається вхід), технічні характеристики пристрою (тип операційної системи, браузер, розширення екрану тощо), історію попередніх транзакцій та часові патерни активності, зокрема, частоту входів у систему в різні години доби чи дні тижня.

Наступним кроком є попередня обробка даних. На цьому етапі отримані дані нормалізуються, очищується від шумів, дублювань і аномальних значень. Наприклад, якщо геолокація була визначена з похибкою або транзакція зареєстрована у некоректному форматі, система усуває такі неточності.

Після цього система переходить до виділення ознак – процесу, в рамках якого автоматично або з допомогою експертів визначаються найінформативніші параметри для подальшої обробки. Наприклад, може бути виявлено, що саме частота зміни пристроїв чи раптова зміна часових патернів активності є ключовими предикторами потенційної компрометації облікового запису [16].

На етапі навчання моделі використовується попередньо зібраний масив даних для створення математичної моделі, здатної класифікувати користувацькі сесії або визначати рівень ризику транзакцій. Це здійснюється за допомогою алгоритмів на кшталт випадкових лісів, градієнтного бустингу або нейронних мереж. Модель «вчиться» розпізнавати закономірності в поведінці користувачів, які відповідають або нормальній, або підозрілій активності.

Після побудови моделі здійснюється її оцінка та валідація – тестування на нових, раніше не бачених даних. Це дозволяє оцінити точність, повноту, специфічність та інші метрики ефективності моделі. За потреби проводиться тонке налаштування параметрів, щоб мінімізувати кількість хибнопозитивних або хибнонегативних результатів.

На наступному етапі модель інтегрується в продуктивне середовище: вона починає обробляти запити в реальному часі, автоматично оцінюючи рівень ризику кожного звернення та рекомендує відповідний рівень автентифікації (наприклад, запит на біометричну перевірку або тимчасове блокування доступу).

Останній, але критично важливий компонент – постійне перенавчання моделі. У процесі експлуатації система збирає нові дані, що відображають зміну поведінкових патернів користувачів, появу нових загроз або адаптацію зловмисників до існуючих заходів безпеки. Саме здатність до динамічного оновлення моделей дозволяє підтримувати високу точність класифікації [17].

Загалом, завдяки цій багаторівневій архітектурі, машинне навчання забезпечує ефективну, гнучку й масштабовану основу для побудови адаптивних механізмів автентифікації у фінансовому середовищі.

2.2 Типи алгоритмів та критерії оцінки ефективності машинного навчання для адаптивної автентифікації

У сучасних системах адаптивної автентифікації для фінансових установ використовуються різні типи алгоритмів машинного навчання, кожен з яких має свої переваги та недоліки. У таблиці 2.1 наведено порівняльний аналіз основних алгоритмів.

З таблиці видно, що для різних завдань автентифікації оптимальними є різні алгоритми машинного навчання. Саме тому в сучасних системах часто використовується композиція алгоритмів, що дозволяє отримати найкращі результати [18].

Таблиця 2.1 – Порівняння алгоритмів машинного навчання для систем адаптивної автентифікації

Тип алгоритму	**Переваги**	**Недоліки**	**Точність (середня)**	**Області застосування в автентифікації**
Алгоритми на основі дерев рішень (Random Forest, XGBoost)	- висока інтерпретованість результатів; - стійкість до викидів; - здатність враховувати нелінійні залежності;	- схильність до перенавчання - обмежена здатність до узагальнення	92-95%	- визначення рівня ризику операції; - аналіз поведінкових патернів;
Нейронні мережі (LSTM, CNN)	- здатність виявляти складні залежності; - висока точність при наявності великого обсягу даних; - можливість роботи з різними типами даних;	- «чорна скринька»; - складність інтерпретації; - потреба у великих наборах даних для навчання; - висока обчислювальна складність;	94-98%	- аналіз біометричних даних; - виявлення аномалій у поведінці; - розпізнавання складних патернів;
Методи на основі Word Embeddings (FastText, Word2Vec)	- ефективна робота з текстовими даними; - здатність враховувати семантичний контекст; - компактне представлення даних;	- обмеженість застосування лише для текстових даних; - залежність від якості навчального корпусу;	89-93%	- аналіз текстових запитів; - виявлення фішингових атак; - обробка користувацьких команд;
Методи кластеризації (K-means, DBSCAN)	- не потребують маркованих даних; - здатність виявляти природні групи користувачів; - простота реалізації;	- чутливість до початкової ініціалізації; - складність визначення оптимальної кількості кластерів;	85-90%	- виявлення аномальної поведінки; - сегментація користувачів за рівнем ризику;
Ансамблеві методи	- висока стійкість до помилок; - краща узагальнююча здатність; - зниження ризику перенавчання.	- висока обчислювальна складність; - складність інтерпретації.	93-97%	- комплексний аналіз ризиків; - багатофакторна автентифікація.

Для оцінки ефективності алгоритмів машинного навчання в системах адаптивної автентифікації використовуються як класичні метрики, так і специфічні для фінансової сфери показники. У таблиці 2.2 наведено порівняння ключових метрик для різних алгоритмів.

Таблиця 2.2 – Порівняння ефективності алгоритмів машинного навчання за ключовими метриками

Метрика	**Random Forest**	**LSTM**	**FastText**	**XGBoost**	**Комбінований підхід**
Точність (Accuracy)	91.3%	95.8%	92.7%	93.5%	96.4%
Повнота (Recall)	88.6%	92.1%	90.4%	91.8%	94.7%
Precision	89.2%	94.5%	91.3%	92.6%	95.2%
F1-score	88.9%	93.3%	90.8%	92.2%	94.9%
AUC-ROC	0.92	0.96	0.93	0.94	0.97
FAR (False Acceptance Rate)	6.4%	3.2%	5.8%	4.7%	2.9%
FRR (False Rejection Rate)	9.7%	6.9%	8.5%	7.8%	5.3%
Час прийняття рішення (мс)	45	120	62	53	150
Обчислювальні вимоги	Низькі	Високі	Середні	Середні	Високі

Як видно з таблиці 2.2, комбінований підхід, що використовує переваги різних алгоритмів, демонструє найкращі результати за більшістю метрик, хоча і потребує більше обчислювальних ресурсів. FastText показує досить високі результати, особливо враховуючи його середні обчислювальні вимоги, що робить цей алгоритм привабливим для систем, які працюють у режимі реального часу.

2.3 Механізм роботи алгоритму FastText у контексті адаптивної автентифікації

FastText є ефективним алгоритмом для обробки природної мови, розробленим Facebook AI Research, який має особливе значення для систем

адаптивної автентифікації. В контексті фінансових установ FastText застосовується для аналізу текстових даних, пов'язаних з автентифікацією та виявленням потенційних кібератак [19].

Ключовими компонентами механізму роботи FastText є:

– представлення слів у вигляді векторів (word embeddings), які алгоритм перетворює текстові дані на числові вектори фіксованого розміру, що зберігають семантичні властивості слів;

– використання n-грам на рівні символів, які FastText аналізує за підпослідовністю символів у словах, що дозволяє ефективно працювати з рідкісними словами та морфологічно багатими мовами;

– ієрархічна класифікація, що прискорює процес навчання та класифікації, де використовується ієрархічна структура (дерево Хаффмана);

– використання підслів (subword), що дозволяє алгоритму розуміти морфологічну структуру слів;

– швидкість та ефективність, що дозволяє алгоритму бути оптимізованим для швидкого навчання на великих наборах даних.

Завдяки цим характеристикам, FastText може бути ефективно інтегрований у систему адаптивної автентифікації для аналізу текстових даних користувача (наприклад, відповідей на запитання безпеки або поведінкових моделей у текстовому спілкуванні). На основі векторного представлення тексту алгоритм дозволяє швидко класифікувати, чи є поведінка користувача типовою чи підозрілою, порівнюючи її з попередніми зразками. Роботу алгоритму FastText у системі адаптивної автентифікації можна представити у вигляді схеми (рис. 2.1).

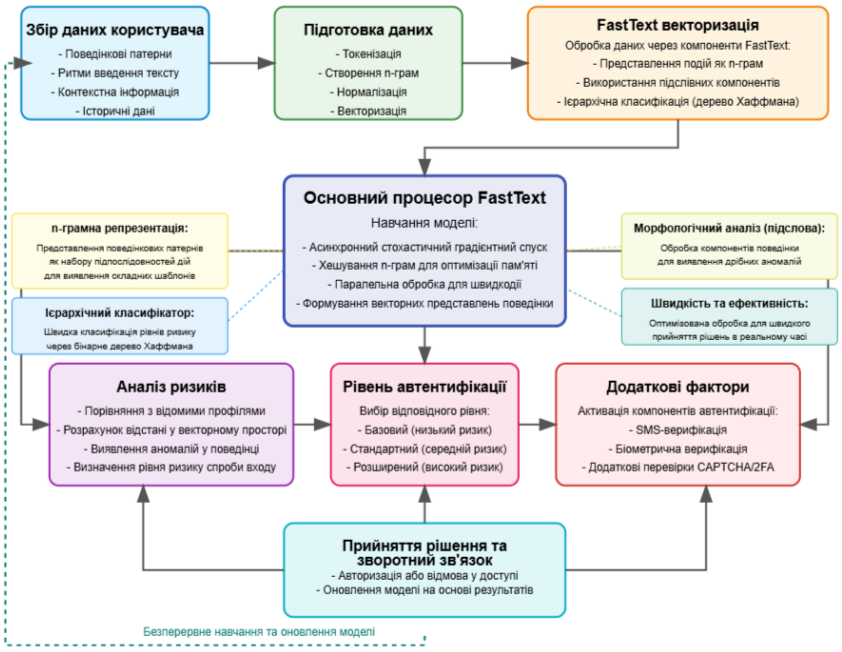


Рисунок 2.1 – Схема роботи алгоритму FastText у системі адаптивної автентифікації

Завдяки високій продуктивності FastText здатний в реальному часі обробляти великі обсяги даних, що надходять від користувачів, і приймати рішення щодо подальших кроків автентифікації, наприклад, підвищення рівня безпеки (використання біометричних даних, двофакторної автентифікації) у разі виявлення підозрілої активності [20].

2.4 Навчання моделі FastText для систем адаптивної автентифікації

Процес навчання моделі FastText для систем адаптивної автентифікації є критично важливим етапом, що визначає якість та ефективність подальшого функціонування системи безпеки. FastText пропонує два основних методи навчання, кожен з яких має свої переваги у специфічних сценаріях використання: skip-gram та CBOW (Continuous Bag of Words).

Метод CBOW базується на прогнозуванні цільового слова на основі його контексту – оточуючих слів у заданому вікні. Під час навчання модель отримує набір контекстних слів і намагається передбачити центральне слово між ними. Цей підхід демонструє високу ефективність у завданнях, де важливо розуміти загальний

контекст повідомлення, що робить його особливо цінним для аналізу фішингових атак та підозрілих запитів у фінансових системах. CBOW навчається швидше порівняно з методом skip-gram і показує кращі результати для частотних слів, що важливо для обробки стандартизованих фінансових комунікацій.

На противагу цьому, метод skip-gram працює у зворотному напрямку – модель прогнозує контекстні слова на основі цільового слова. Під час навчання модель отримує одне слово і намагається передбачити слова, що його оточують у контексті. Skip-gram показує вищу точність для рідкісних слів і загалом забезпечує кращу якість векторних представлень для семантичних завдань. Цей метод особливо корисний для виявлення аномальної поведінки користувачів та нестандартних патернів комунікації, що можуть сигналізувати про компрометацію облікового запису.

Таблиця 2.3 – Порівняння ефективності алгоритмів машинного навчання за ключовими метриками

Характеристика	**Skip-gram**	**CBOW**
Навчальна стратегія	За одним словом передбачає сусідні слова	За словами контексту передбачає центральне слово
Швидкість навчання	Повільніше (скалюється лінійно з розміром даних)	Швидше (більш ефективний на великих даних)
Чутливість до даних	Краще для невеликих/рідкісних даних	Потребує більше даних для стабільних результатів
Обробка рідкісних слів	Краща (зв’язок через n-грамні підслова)	Гірша, якщо слово малопоширене чи нове
Сценарії застосування	Виявлення аномалій, класифікація невеликих корпусів	Створення загальних моделей класифікації

Отже, Skip-gram орієнтований на предикцію контексту з одного слова, повільніше навчається, але показує кращу якість на малих датасетах і добре працює з рідкісними словами. У FastText Skip-gram вважається потужнішим при обмеженій кількості даних. CBOW швидше навчається (через паралельне передбачення слів з

контексту) і добре підходить для великих корпусів, але при малій кількості навчальних прикладів його точність може бути нижчою.

У контексті систем адаптивної автентифікації часто використовується гібридний підхід, коли базова модель навчається за допомогою skip-gram для забезпечення високої якості векторних представлень, а потім доповнюється класифікаторами, навченими за методом CBOW для конкретних завдань виявлення загроз. Тобто у задачах автентифікації вибір моделі може залежати від обсягу і характеру даних.

Для систем автентифікації у фінансовому секторі процес навчання має включати специфічні домени текстів: банківську термінологію, фінансові звіти, користувацькі запити щодо типових банківських операцій, а також зразки фішингових повідомлень та шахрайських комунікацій. Це забезпечує чутливість моделі до нюансів, характерних саме для фінансової сфери.

Не менш важливим аспектом є безперервне навчання та оновлення моделі на нових даних, що дозволяє системі адаптуватися до еволюції мовних патернів користувачів та нових видів загроз. Цей процес відбувається у режимі інкрементального навчання, коли модель оновлюється без необхідності повного перенавчання з нуля [21].

Короткого розглянемо безперервну модель skip-gram, на основі якої побудована представлена модель. Припустимо, що маємо словниковий запас розміру W, де кожне слово ідентифікується індексом $w \in \{1, \ldots, W\}$. Головна мета полягає у тому, щоб навчити векторне представлення для кожного слова w. На основі дистрибутивної гіпотези, модель навчає представлення слів таким чином, щоб вони добре передбачали слова, що з’являються в його контексті.

Формально, маючи великий навчальний корпус, представлений як послідовність слів $w_1, \ldots, w_T$, мета моделі skip-gram полягає у максимізації наступної логарифмічної ймовірності:

$$\sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c \mid w_t), \tag{3.1}$$

де t – індекс таргетного слова в наборі даних;

C_t – множина індексів слів, що оточують слово w_t;

w_t – таргетне слово/слово-мішень (target word);

w_c – слово з контексту (context word).

Ймовірність спостереження контекстного слова w_c за умови слова w_t параметризується за допомогою векторів слів. У даній формулі ми підсумовуємо для кожного таргетного слова w_t і для кожного контекстного слова w_c у контексті цього таргетного слова логарифм ймовірності того, що слово w_c зустрічається поруч із w_t. Метою є максимізувати цю суму, щоб навчити модель правильно передбачати контекст для кожного слова.

Даний вираз можна побачити, наприклад, у тренуванні моделей на основі контексту, таких як Word2Vec або GloVe, де мета – це побудувати векторні представлення слів, які максимально добре описують контекстне оточення цих слів у тексті.

Нехай s – це функція, що відображає пари (слово, контекст) у скалярні значення з R. Одним із варіантів визначення ймовірності контекстного слова є математична функція *softmax*, яка часто використовується в машинному навчанні, зокрема в задачах, де потрібно працювати з ймовірностями для кількох класів. Вона перетворює вектор сирих оцінок (так званих логітів) у ймовірності. Кожна оцінка перетворюється на ймовірність, що лежить між 0 та 1, і всі ймовірності для всіх класів (або варіантів) в сумі дають 1. А саме:

$$p(w_c \mid w_t) = \frac{e^{s(w_t, w_c)}}{\sum_{j=1}^{W} e^{s(w_t, j)}}, \tag{3.2}$$

де w_t – таргетне слово/слово-мішень (target word);

w_c – слово з контексту (context word);

W – розмір словника.

Однак така модель не підходить для нашого випадку, оскільки вона передбачає, що для заданого слова w_t передбачається лише одне контекстне слово w_c.

Задачу передбачення контекстних слів можна замість цього сформулювати як набір незалежних задач бінарної класифікації. Тоді метою є незалежно передбачити наявність (або відсутність) контекстних слів.

Натомість цю задачу для слова на позиції t можна розглядати як набір незалежних задач бінарної класифікації: усі контекстні слова – позитивні приклади, а негативні вибираються випадково зі словника. Використовуючи логістичну функцію втрат для обраної контекстної позиції c, отримаємо негативну логарифмічну ймовірність:

$$\log\left(1 + e^{-s(w_t,w_c)}\right) + \sum_{n\in N_{t,c}} \log\left(1 + e^{s(w_t,n)}\right), \quad (3.3)$$

де $s(w_t,w_c)$ – функція оцінки;

$N_{t,c}$ – множина негативних прикладів зі словника.

Позначивши логістичну функцію втрат як $l : x \mapsto \log 1 + e^{-x}$, перепишемо цільову функцію так:

$$\sum_{t=1}^{T}\left[\sum_{c\in C_t} l(s(w_t, w_c)) + \sum_{n\in N_{t,c}} l(-s(w_t, n))\right], \quad (3.4)$$

де T – загальна кількість слів у корпусі;

w_t – таргетне слово/слово-мішень (target word) на позиції t;

C_t множина контекстних позицій для слова w_t;

$N_{t,c}$ – набір негативних слів, вибраних випадково для кожної пари (w_t,w_c).

Функцію s можна реалізувати через вектори слів. Природною параметризацією для функції оцінки s між словом w_t і контекстним словом w_c є використання векторів слів. Для кожного слова w у словнику визначаються два вектори u_w і v_w у $\mathbb{R}^d$. Ці два вектори іноді називають, відповідно, вхідним та вихідним векторами. Зокрема, ми маємо вектори u_{wt} та v_{wc}, які відповідають словам w_t і w_c. Тоді: $s(w_t, w_c) = u_{w_t}^T v_{w_c}$. Це і є модель skip-gram з негативною вибіркою, що є ефективним методом навчання векторних представлень слів, який формулює задачу прогнозування контекстних слів як набір незалежних бінарних

класифікацій. Основна ідея полягає в тому, щоб, маючи задане слово-мішень w_t, передбачити, які слова з'являються поруч у контексті. Для кожного такого позитивного прикладу (пари (w_t, w_c)), модель додатково вибирає кілька негативних прикладів (w_t, n), де n – слово, яке не зустрічається в контексті w_t і вибране випадковим чином зі словника.

Такий підхід суттєво знижує обчислювальні витрати у порівнянні з повною *softmax*-нормалізацією по всьому словнику, зберігаючи при цьому здатність моделі виявляти латентні семантичні та синтаксичні зв'язки між словами [22].

Підслівна модель. Використання окремих векторів для кожного слова не враховує їхню морфологічну структуру. Тому пропонується взяти нову функцію *s*, яка враховує внутрішню будову слів. Кожне слово *w* представляється як мішок символьних *n-грам*. Додаємо символи межі < and > на початку та в кінці слова, щоб розрізняти префікси й суфікси. Також саме слово *w* включається до множини *n-грам*.

Наприклад, для слова «where» і n = 3 отримаємо: <wh, whe, her, ere, re>. Розглядаємо всі *n-грами* довжиною від 3 до 6. Припустимо, що маємо словник *n-грам* розміру *G*. Позначимо $G_w \subset \{1, \ldots, G\}$ як множину *n-грам* у слові *w*. Кожній *n-грам*і *g* відповідає вектор z_g.

Слово представляється як сума векторів його *n-грам*. Функція оцінювання виглядає так:

$$s(w, c) = \sum_{g \in G_w} Z_g^T v_c, \tag{3.5}$$

де *s (w, c)* – оцінка схожості між словом *w* та контекстом *c*;

w – цільове слово (target word);

c – контекстне слово (context word);

G_w – множина елементів, які репрезентують слово *w*.

Ця модель дозволяє ділитися представленнями між словами й добре працює для рідкісних слів. Щоб обмежити використання пам'яті, застосовуємо *хеш-функцію (FNV-1a)*, яка відображає *n-грам*и в індекси від 1 до *K* (встановлено *K* =

$2 \cdot 10^6$). Зрештою, слово представляється своїм індексом у словнику та набором захешованих *n-грам*.

Математично модель FastText можна описати наступним чином: для вхідного тексту T, що складається з послідовності слів $w_1, w_2, \ldots, w_n$, модель обчислює ймовірність класу c за формулою:

$$P(c \mid T) = softmax\left(A \cdot \left[\frac{1}{n}\sum_{i=1}^{n} x_{w_i}\right]\right), \quad (3.6)$$

де x_{w_i} – векторне представлення слова w_i;

A – матриця параметрів класифікатора;

softmax – функція нормалізації для отримання розподілу ймовірностей.

У контексті адаптивної автентифікації модель FastText навчається на наборі даних, що містить як легітимні запити користувачів, так і приклади шкідливих запитів. Після навчання модель здатна класифікувати нові запити та визначати рівень ризику в режимі реального часу [23].

Однією з найбільш значущих переваг FastText для сучасних фінансових установ є його виняткова мультимовна підтримка, що охоплює понад 150 мов світу. Це робить алгоритм ідеальним рішенням для міжнародних банків та фінансових організацій, що обслуговують клієнтів з різних країн та культурних середовищ. У контексті фінансової безпеки мультимовність також відіграє ключову роль у запобіганні відмиванню коштів та фінансуванню тероризму. FastText дозволяє аналізувати текстові компоненти транзакцій (коментарі, призначення платежів, супровідну документацію) на наявність підозрілих патернів незалежно від мови, якою вони написані. Це суттєво підвищує ефективність систем AML (Anti-Money Laundering) у міжнародних фінансових операціях.

Для організацій, що працюють на ринках з кількома офіційними мовами (наприклад, Швейцарія, Бельгія, Канада), FastText пропонує можливість безшовної інтеграції різних мовних просторів у єдину систему безпеки. Це особливо важливо для клієнтів, які регулярно використовують кілька мов у своїх фінансових комунікаціях.

Також FastText досягає значного прискорення при збереженні якості моделі: навчання на корпусі з 1 мільярда слів займає приблизно 5-10 хвилин на багатоядерних системах, тоді як еквівалентні за якістю нейромережеві моделі потребують від 2 до 24 годин на аналогічному обладнанні.

2.5 Інтеграція FastText у системи адаптивної автентифікації фінансових установ

Впровадження алгоритмів машинного навчання в систему автентифікації фінансових установ потребує комплексного підходу та врахування специфіки фінансового сектору, зокрема дотримання регуляторних вимог, таких як Постанова 58 Національного банку України [2].

Процес інтеграції FastText у системи адаптивної автентифікації передбачає кілька ключових етапів. Насамперед, відбувається збір та попередня обробка текстових даних користувача, включаючи історію пошукових запитів, повідомлень у системах обміну інформацією, коментарів та інших текстових взаємодій. Ці дані формують так званий «текстовий відбиток» користувача – унікальний профіль мовної поведінки, характерний для конкретної особи.

Після створення текстового профілю FastText здійснює векторне представлення цих даних у багатовимірному просторі, де семантично близькі слова та фрази розташовуються поруч. Це дозволяє системі ідентифікувати характерні патерни використання мови, властиві конкретному користувачу – включаючи специфічні терміни, жаргонізми, типові помилки написання та синтаксичні конструкції.

Під час активної сесії користувача система постійно аналізує нові текстові взаємодії, порівнюючи їх із встановленим профілем. Суттєві відхилення від звичайної моделі мовної поведінки можуть свідчити про потенційний несанкціонований доступ. Наприклад, раптова зміна лексики, використання незвичних синтаксичних конструкцій або звертання до тем, що раніше не були

характерні для конкретного користувача, можуть викликати сигнал тривоги в системі безпеки.

Особливо важливою є здатність FastText аналізувати контекстуальні зв'язки у текстових даних. Це дозволяє виявляти витончені спроби соціальної інженерії, які часто супроводжуються лінгвістичними маркерами маніпуляції – використанням термінів, що створюють відчуття терміновості, апеляцією до авторитетів або специфічними переконуючими конструкціями [24].

Для фінансових установ ця функціональність має критичне значення, оскільки дозволяє забезпечити додатковий рівень захисту від фішингових атак та спроб несанкціонованого доступу до банківських рахунків. При інтеграції з іншими методами біометричної автентифікації, такими як аналіз клавіатурного почерку або поведінкова біометрія, FastText створює комплексну систему захисту, що безперервно верифікує автентичність користувача протягом усієї сесії.

Алгоритм FastText пропонує низку суттєвих переваг для систем кібербезпеки, особливо у сфері виявлення та запобігання текстово-орієнтованим загрозам. На відміну від традиційних підходів, що базуються на статичних правилах та сигнатурах, FastText забезпечує глибинне семантичне розуміння текстового контенту, що дозволяє виявляти навіть невідомі раніше типи атак.

Одна з найважливіших переваг FastText полягає у здатності обробляти великі обсяги неструктурованих текстових даних у режимі реального часу. Для сучасних фінансових установ, які щодня обробляють мільйони текстових взаємодій з клієнтами через різноманітні канали комунікації, ця швидкість аналізу є критично важливою. Архітектура FastText, що базується на ефективному представленні слів як суми n-грам символів, дозволяє досягти вражаючої продуктивності навіть при обмежених обчислювальних ресурсах.

Важливою особливістю FastText є його стійкість до орфографічних помилок та опечаток – проблеми, характерної для повсякденної комунікації. Це дозволяє алгоритму коректно обробляти навіть недосконалі текстові дані, зберігаючи високу точність класифікації та аналізу. Ця властивість має особливе значення для

виявлення фішингових атак, які часто містять навмисні опечатки або заміни символів для уникнення детектування.

FastText також демонструє високу ефективність у роботі з рідкісними та невідомими словами, використовуючи підслівний підхід до представлення тексту. Це дозволяє системам безпеки ефективно виявляти обфускацію – навмисне спотворення слів та фраз для обходу традиційних систем фільтрації. Наприклад, зловмисники часто використовують заміну літер (наприклад, «p@ssw0rd» замість «password») або вставку додаткових символів для приховування ключових слів у фішингових повідомленнях. FastText здатен розпізнавати такі модифікації завдяки роботі з підсловами.

У сфері виявлення соціальної інженерії FastText пропонує унікальні можливості для аналізу психолінгвістичних маркерів маніпулятивного впливу. Алгоритм здатен виявляти дискурсивні стратегії, характерні для соціальних інженерів – створення відчуття терміновості, апеляція до страху втрати, використання удаваного авторитету чи формування довірливих відносин. Аналіз векторних представлень тексту дозволяє визначити ці патерни навіть у випадках, коли вони реалізовані новими, раніше невідомими лінгвістичними конструкціями.

Ще однією перевагою FastText є низький поріг входу та відносна простота інтерпретації результатів. На відміну від більш складних моделей глибокого навчання, FastText пропонує прозорий механізм класифікації та кластеризації, що спрощує аудит рішень системи безпеки та відповідність регуляторним вимогам фінансового сектору щодо пояснюваності алгоритмів прийняття рішень.

Застосування FastText у фінансовому секторі характеризується низкою галузевих особливостей, які вимагають специфічних підходів до імплементації та використання цієї технології. Фінансові установи функціонують у середовищі з підвищеними вимогами до безпеки, суворими регуляторними обмеженнями та високою вартістю помилок, що створює унікальний контекст для впровадження систем автентифікації на основі аналізу природної мови.

Одним із ключових аспектів застосування FastText у фінансовому секторі є аналіз текстових компонентів транзакцій – призначень платежів, коментарів до операцій, супровідної документації. Ці текстові дані містять цінну інформацію для виявлення потенційно шахрайських операцій. Наприклад, FastText може ідентифікувати нехарактерні формулювання в призначеннях платежів, які відрізняються від звичайних патернів конкретного користувача, або виявляти лінгвістичні маркери, типові для шахрайських схем – особливо у випадках соціальної інженерії, коли клієнта змушують здійснити переказ коштів під надуманим приводом.

Особливу цінність у фінансовому секторі представляє здатність FastText працювати з банківською термінологією та професійним жаргоном. Фінансові комунікації насичені специфічними термінами, абревіатурами та професіоналізмами, що вимагає додаткового навчання моделі на спеціалізованих корпусах текстів. При цьому важливо забезпечити баланс між спеціалізацією моделі на фінансовій лексиці та збереженням здатності працювати зі звичайною мовою клієнтів, яка може суттєво відрізнятися від професійної термінології [25].

Регуляторні вимоги до фінансових установ створюють додатковий контекст для впровадження FastText. Зокрема, важливою є прозорість та інтерпретованість рішень системи безпеки. На відміну від багатьох «чорноскриньових» алгоритмів машинного навчання, FastText пропонує відносно прозорий механізм класифікації, що дозволяє пояснити прийняті рішення у зрозумілих для регуляторів термінах. Це особливо важливо в контексті європейського регулювання GDPR та подібних нормативних актів, що встановлюють «право на пояснення» для автоматизованих рішень, які впливають на користувачів.

Імплементація FastText у фінансових установах також повинна враховувати необхідність інтеграції з існуючими системами захисту та відповідності внутрішнім політикам безпеки. Це включає розробку механізмів безпечного доступу до текстових даних, шифрування векторних представлень користувацьких профілів та встановлення чітких протоколів реагування на виявлені аномалії.

Важливим аспектом застосування FastText у фінансовому секторі є калібрування порогових значень для виявлення аномалій. Фінансові установи повинні знайти оптимальний баланс між безпекою та зручністю користування. Занадто чутлива система створюватиме надмірну кількість хибних спрацьовувань, що призводить до «втоми від тривог» у операторів безпеки та незручностей для клієнтів. З іншого боку, недостатня чутливість може пропустити реальні загрози.

Для міжнародних фінансових груп важливою є здатність FastText забезпечувати єдині стандарти безпеки у різних регіонах з урахуванням локальних особливостей. Це вимагає розробки багаторівневої архітектури, де базова модель доповнюється регіонально-специфічними компонентами, що враховують особливості місцевого фінансового сленгу, регуляторні вимоги та типові для регіону шахрайські схеми.

Унікальним аспектом застосування FastText у фінансовому секторі є можливість інтеграції з системами протидії відмиванню коштів (AML) та фінансуванню тероризму (CFT). Аналіз текстових даних дозволяє виявляти лінгвістичні патерни, характерні для підозрілих операцій, та встановлювати семантичні зв’язки між транзакціями, що на перший погляд виглядають незалежними, але насправді є частиною складної схеми відмивання коштів.

Висновки до розділу 2

У даному розділі було розглянуто механізм роботи алгоритмів машинного навчання в контексті адаптивної автентифікації користувачів у фінансових установах. Алгоритми машинного навчання дозволяють створювати адаптивні системи автентифікації, які пристосовуються до змін у поведінці користувачів та нових загроз. Різні типи алгоритмів мають свої переваги і недоліки, тому оптимальним є комбінований підхід, що дозволяє максимізувати ефективність за ключовими метриками. FastText демонструє високу ефективність при роботі з текстовими даними та аналізі користувацьких запитів, що робить його перспективним для використання в системах адаптивної автентифікації. Але

процес навчання та валідації моделей потребує ретельної підготовки даних та комплексної оцінки за множиною метрик. Інтеграція алгоритмів машинного навчання в системи автентифікації фінансових установ дозволяє значно підвищити рівень безпеки та зменшити операційні витрати.

3 ПРОЄКТУВАННЯ ВЕБСИСТЕМИ АДАПРИВНОЇ АВТЕНТИФІКАЦІЇ

3.1 Діаграма варіантів використання

Діаграма варіантів використання (use case diagram) є ключовим інструментом на етапі аналізу вимог до програмного забезпечення. За допомогою даної діаграми можна візуалізувати функціональність системи з точки зору користувача. Use case diagram чітко визначає межі системи, показуючи, що входить до її складу, а що – ні. З її допомогою легко ідентифікувати акторів, які взаємодіють із системою. Діаграма варіантів використання сприяє кращому розумінню потреб користувачів та їх очікувань від системи, визначає її основні функціональні можливості. Use case diagram є основою для подальшого проєктування та розробки програмного забезпечення [26]. Також вона полегшує комунікацію між замовниками, розробниками та іншими зацікавленими сторонами (рис. 3.1).

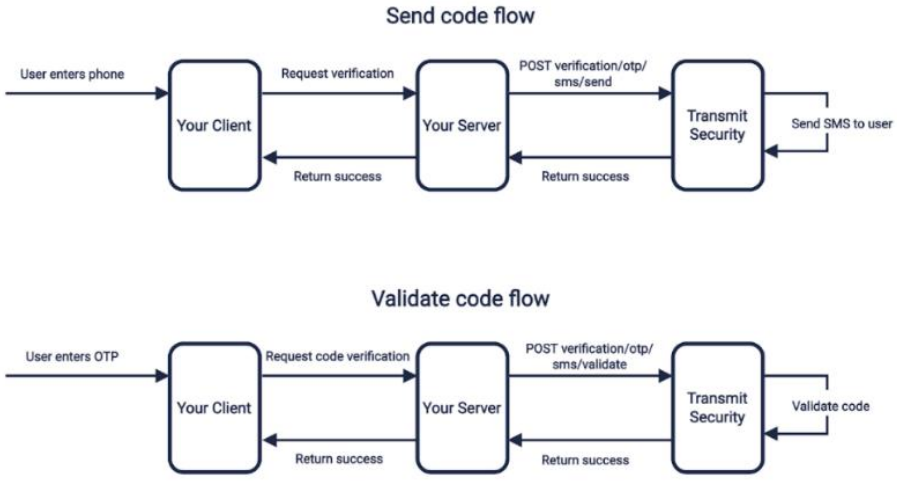


Рисунок 3.1 – Загальна структурна діаграма OTP процесу

Діаграма варіантів використання виявляє можливі ризики та проблеми на ранніх етапах розробки, та застосовується для оцінки обсягу робіт, необхідних для реалізації проєкту. Use case diagram дозволяє структурувати та організувати великий обсяг інформації про систему і є надійним інструментом для документування вимог до програмного забезпечення. Діаграма варіантів використання допомагає забезпечити відповідність розробленої системи потребам

користувачів і може бути використана для тестування ПЗ, визначаючи, які функції необхідно перевірити. Use case diagram сприяє покращенню якості програмного забезпечення, що робить його більш зручним та корисним для користувачів. З її допомогою можна легко визначити залежності між різними функціями системи. Діаграма варіантів використання полегшує процес внесення змін та доповнень до системи в майбутньому. Вона допомагає уникнути непорозумінь та конфліктів між різними членами команди розробників. Use case diagram сприяє більш ефективному управлінню проектом розробки програмного забезпечення (рис. 3.2). Вона є важливим елементом успішної розробки програмного забезпечення, яке відповідає потребам користувачів та вимогам замовника.

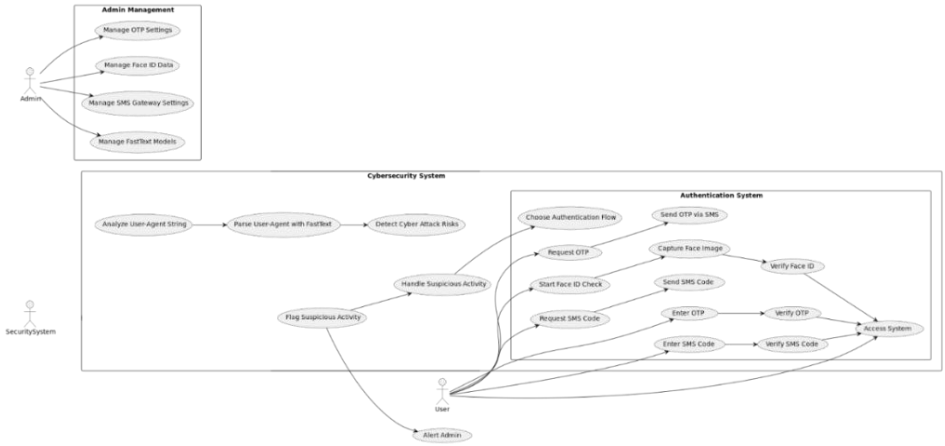


Рисунок 3.2 – Діаграма варіантів використання вебсистеми адаптивної автентифікації

У варіанті використання Analyze User-Agent String система отримує User-Agent рядок від користувача або пристрою. SecuritySystem ініціює процес аналізу цього рядка. Система виділяє ключові характеристики пристрою або браузера. Проводиться попередній розбір на основі шаблонів і ознак. Аналіз допомагає визначити аномальні або незвичні значення. Цей процес є першим етапом виявлення можливих атак. Аналізовані дані передаються для подальшої обробки. У разі підозрілих знахідок система продовжує детальнішу перевірку.

У варіанті використання Parse User-Agent with FastText після первинного аналізу система використовує модель FastText. FastText допомагає краще

класифікувати та інтерпретувати User-Agent рядок. Використовуються попередньо навчені мовні моделі. Завдяки цьому підвищується точність визначення типу пристрою чи браузера. Система враховує можливі маніпуляції або підробки User-Agent. Результати парсингу передаються на етап оцінки ризиків. Обробка відбувається швидко завдяки оптимізації FastText. Система готує структуровані дані для наступного етапу [27].

У варіанті використання Detect Cyber Attack Risks після парсингу User-Agent система оцінює рівень ризику. Використовуються внутрішні правила і алгоритми виявлення атак. Підозрілі шаблони або поведінка фіксуються окремо. Визначаються потенційні загрози, пов’язані із фішингом або ботами. У разі виявлення ризику система змінює поведінку автентифікації. Наявність загроз реєструється в журналі подій. Якщо ризик високий, активуються додаткові заходи безпеки. Цей процес допомагає попередити компрометацію системи.

Під час варіанту використання Flag Suspicious Activity якщо виявлено загрозу, система позначає подію як підозрілу. Подія заноситься до бази інцидентів безпеки. Система може обмежити або змінити доступ користувача. Відзначається профіль користувача для подальшого моніторингу. Інформація про підозрілу активність відправляється адміністраторам. Це дозволяє швидко реагувати на потенційні атаки. Одночасно користувачеві може бути запропоновано пройти додаткову перевірку. Процес є невід’ємною частиною загальної системи захисту.

Після позначення активності система визначає подальші дії. У варіанті використання Handle Suspicious Activity може бути активовано вибір альтернативного шляху автентифікації. Користувачу можуть запропонувати багатоетапну перевірку особи. Система може тимчасово заблокувати сесію. Адміністратори отримують повідомлення для подальшого аналізу. Обробка підозрілої активності інтегрована в основний захист. Цей етап є важливим для забезпечення надійності системи. Всі дії документуються для подальшого аудиту.

Під час варіанта використання Request OTP користувач надсилає запит на отримання одноразового пароля (OTP). Система генерує унікальний код для

користувача. Використовується безпечний алгоритм створення OTP. Код надсилається через SMS або інший канал. Система перевіряє правильність номеру телефону користувача. Згенерований OTP має обмежений строк дії. Процес захищає від несанкціонованого доступу. Користувач переходить до введення отриманого коду.

Варіант використання Send OTP via SMS відображає сценарій, коли система надсилає одноразовий пароль на вказаний номер телефону. Для надсилання використовується налаштований SMS-шлюз. Повідомлення містить тільки необхідну інформацію. Використовується шифрування для передачі даних. Система реєструє факт надсилання коду. У разі помилки повторне надсилання можливе за запитом. SMS код повинен бути доставлений протягом декількох секунд. Надісланий код використовується для подальшої верифікації.

У варіанті використання Enter OTP користувач вводить отриманий одноразовий пароль у відповідному полі. Система приймає введене значення для перевірки. Перевіряється правильність введення та відповідність згенерованому коду. При помилці користувачу пропонується нова спроба. Після декількох невдалих спроб доступ може бути обмежено. Успішне введення дозволяє перейти до наступного кроку. Код перевіряється з урахуванням обмеження часу дії. Процес є частиною двофакторної автентифікації.

Варіант використання Verify OTP описує сценарій, коли система порівнює введений код з тим, що був надісланий. Перевіряється актуальність і правильність OTP. Успішна верифікація дозволяє авторизувати користувача. Якщо код прострочений або неправильний, система відхиляє запит. Всі спроби верифікації фіксуються у журналі безпеки. Перевірка коду може супроводжуватись додатковими заходами захисту. У разі невдачі система пропонує новий код. Успішна перевірка відкриває доступ до системи.

Під час варіанта використання Start Face ID Check користувач ініціює перевірку за допомогою Face ID. Система активує камеру пристрою користувача. Запускається процес зчитування біометричних даних обличчя. Перевірка

проводиться у захищеному середовищі. Дані шифруються і не зберігаються без потреби. Процес забезпечує високий рівень безпеки. Система аналізує зображення для створення цифрового відбитка. Після захоплення зображення починається верифікація.

У варіанті використання Capture Face Image під час перевірки система захоплює зображення обличчя користувача. Зображення обробляється локально або у захищеному середовищі. Визначаються ключові біометричні характеристики. Якість зображення перевіряється автоматично. У разі поганої якості робиться нова спроба. Зображення готується для подальшого порівняння. Усі дані обробляються відповідно до політики конфіденційності. Процес є важливою частиною Face ID перевірки.

Варіант використання Verify Face ID необхідний для того, щоб система порівнювала захоплене зображення з еталонними даними. Перевіряється відповідність рис обличчя. При збігу користувач отримує доступ до системи. У разі розбіжностей доступ блокується. Використовується алгоритм машинного навчання для точності. Усі операції проводяться в режимі підвищеної безпеки. Перевірка відбувається в реальному часі. Після верифікації система переходить до надання доступу.

Під час варіанта використання Request SMS Code користувач надсилає запит на отримання SMS коду. Система генерує новий унікальний код. Код призначений для одноразового використання. Використовується безпечний канал для передачі коду. Номер телефону користувача перевіряється на валідність. Процес є частиною багатофакторної автентифікації. Система реєструє запит у журналі подій. Користувач очікує надходження коду.

У варіанті використання Send SMS Code система надсилає згенерований код через SMS. Передача коду виконується через безпечний шлюз. Повідомлення містить тільки потрібний код. Система відслідковує успішність доставки. Користувач отримує повідомлення протягом декількох секунд. Якщо доставка не

відбулась, можливе повторне надсилання. Надісланий код має обмежений час дії. Система готується до перевірки введеного коду.

Під час варіанта використання Enter SMS Code користувач вводить отриманий SMS код у відповідному полі. Система зчитує введене значення для перевірки. Відбувається валідація на правильність та строк дії. У разі помилки користувач може спробувати ще раз. Невірні спроби обмежуються політикою безпеки. Введення коду є обов'язковим для подальшого доступу. Система фіксує всі спроби введення. Процес є частиною перевірки ідентичності.

Варіант використання Verify SMS Code необхідний для того, щоб система перевіряла правильність введеного коду. Успішна верифікація дозволяє користувачу продовжити роботу. Якщо код недійсний або прострочений, доступ блокується. Процес перевірки фіксується в системних логах. Для перевірки використовуються захищені алгоритми. У разі помилки користувачу пропонується повторити спробу. При успішній верифікації відкривається доступ до системи. Цей етап забезпечує додатковий рівень безпеки.

Під час варіанта використання Access System після успішної перевірки користувач отримує доступ до системи. Система підтверджує авторизацію користувача. Доступ надається згідно з рівнем прав користувача. Успішний вхід реєструється для аудиту. Користувач отримує інтерфейс системи відповідно до своєї ролі. У разі помилок авторизації доступ обмежується. Перевіряються додаткові фактори автентифікації. Процес забезпечує безпечний доступ до внутрішніх ресурсів.

У варіанті використання Choose Authentication Flow система вибирає відповідний шлях автентифікації для користувача. Вибір залежить від рівня ризику та налаштувань безпеки. Може бути запропонована автентифікація через OTP, Face ID або SMS код. У випадку підозрілої активності вибирається посилений сценарій. Процес автоматизований і не потребує втручання користувача. Вибір шляху фіксується у журналі подій. Система забезпечує баланс між зручністю і безпекою. Обраний шлях визначає подальші кроки авторизації.

Варіант використання Manage OTP Settings необхідний для того, щоб адміністратор мав доступ до налаштувань OTP. Він може змінювати параметри генерації та строку дії кодів. Система дозволяє керувати політикою повторних надсилань. Адміністратор має змогу обмежувати кількість спроб введення коду. Налаштування мають відповідати внутрішнім стандартам безпеки. Зміни фіксуються у системних логах. Адміністратор може переглядати статистику використання OTP. Управління налаштуваннями впливає на загальну безпеку системи.

У варіанті використання Manage Face ID Data адміністратор керує даними для Face ID автентифікації. Він може оновлювати або видаляти біометричні шаблони користувачів. Доступ до даних здійснюється через захищений інтерфейс. Зміни фіксуються для аудиту безпеки. Адміністратор контролює політику обробки біометричних даних. Усі операції повинні відповідати нормам захисту даних. Управління даними дозволяє підтримувати актуальність Face ID перевірок. Процес є критичним для підтримання точності автентифікації.

Manage SMS Gateway Settings, як варіант використання, необхідний для того, щоб адміністратор керував налаштуваннями SMS шлюзу. Він може змінювати постачальника послуг або параметри підключення. Налаштування включають тестування доставки повідомлень. Адміністратор контролює якість і швидкість доставки SMS. Зміни зберігаються у системних логах для перевірки. Важливо дотримуватися політики безпеки при зміні налаштувань. Адміністратор аналізує статистику успішних та невдалих доставок. Управління шлюзом впливає на стабільність роботи автентифікації.

У варіанті використання Manage FastText Models адміністратор має змогу оновлювати або замінювати моделі FastText. Нові моделі можуть бути треновані для покращення точності аналізу. Адміністратор завантажує моделі через захищений інтерфейс. Проводиться тестування нових моделей перед їх впровадженням. Управління моделями дозволяє адаптувати систему до нових загроз. Зміни фіксуються в журналі безпеки. Регулярне оновлення моделей

підвищує рівень захисту. Процес є частиною стратегічного управління кібербезпекою.

3.2 Проєктування Front-End частини

Інструменти для візуального графу залежностей модулів допомагають швидко представити структуру проєкту та взаємозв'язки між його частинами. Це особливо корисно на етапах аналізу та проєктування, де важливо виявити надмірні зв'язки чи циклічні залежності. Подібні інструменти спрощують onboarding нових розробників, які можуть швидше зорієнтуватися у складній архітектурі. Граф, який представлений на рис. 3.3, полегшує рефакторинг, даючи змогу виявити «вузькі місця» чи надмірно зв'язані компоненти. Інструменти візуалізації залежностей модулів також корисні для виявлення прихованих залежностей, які можуть призводити до помилок під час змін у коді. Візуалізація залежностей підтримує процес документування ПЗ і полегшує комунікацію між командами.

Одним з недоліків є перевантаження інформацією у великих проєктах, де граф може стати нечітким і важким для аналізу.

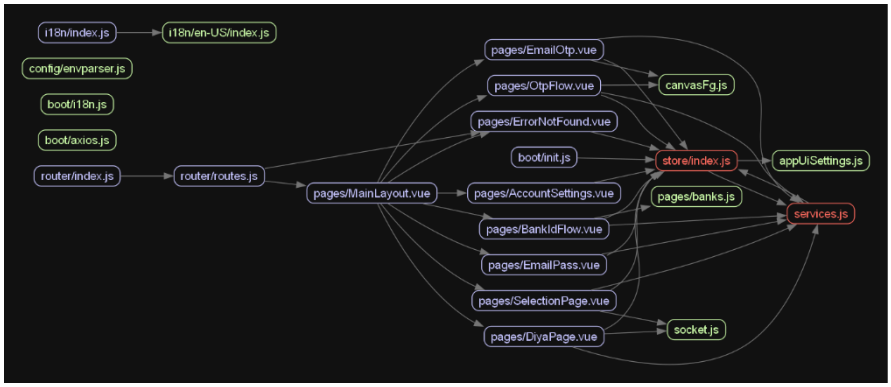


Рисунок 3.3 – Діаграма зв'язків компонентів модуля OTP widget вебсистеми адаптивної автентифікації

Автоматично згенеровані графи іноді можуть показувати лише поверхневі зв'язки, не враховуючи динамічну поведінку модулів. Інтеграція з CI/CD або IDE може вимагати додаткових налаштувань, що збільшує час впровадження. Також

існує ризик надмірної довіри до візуалізації, що може відволікати від аналізу реального коду й логіки програми.

Компонент EmailOtp (рис. 3.4) відповідає за двофакторну автентифікацію користувача за допомогою email і одноразового коду. На першому етапі користувач вводить свою email-адресу, яка перевіряється та надсилається на сервер для генерації OTP.

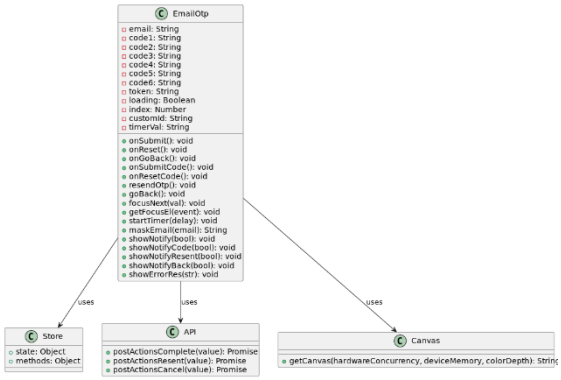


Рисунок 3.4 – Діаграма класів компонента EmailOtp.vue

Після успішної перевірки email, компонент переходить до другої сторінки, де відображається форма введення 6-значного коду підтвердження. Кожне поле вводу коду пов’язане з відповідною змінною (code1-code6), а при введенні фокусується наступне поле автоматично. Якщо код введено неправильно, користувачу виводиться повідомлення про помилку, і він може повторити спробу. За потреби користувач може надіслати код повторно, але лише після завершення таймера, який відображає час до повторної відправки. Кнопка «Назад» дозволяє скасувати процес і повернутися до попереднього кроку, очищаючи всі поля. Повідомлення про помилки або успішні дії відображаються через сповіщення, що викликаються методом notify з бібліотеки Quasar.

Компонент BankId (рис. 3.5) містить логіку автентифікації користувача через BankID. Клас управляє внутрішнім станом, таким як завантаження, токен, номер телефону та коди OTP.

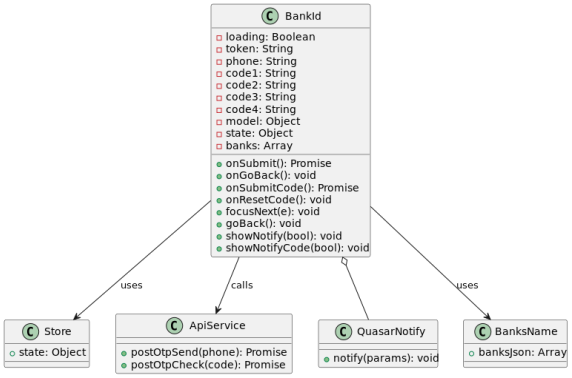


Рисунок 3.5 – Класів компонента BankId.vue

Даний клас містить методи для відправки номера телефону (onSubmit), перевірки коду (onSubmitCode), обробки повернення назад (onGoBack, goBack), фокусування між інпутами (focusNext) та скидання коду (onResetCode). Компонент взаємодіє з класом ApiService, який відповідає за HTTP-запити до серверної частини для надсилання та перевірки OTP. Клас QuasarNotify використовується для виводу повідомлень користувачеві про помилки чи успіх операцій. Також використовується клас Store, що містить глобальний стан застосунку, зокрема такі властивості як state.returnUrl чи state.urlFromCiam. Клас BanksName надає список банків, які використовуються як опції для q-select. Усі ці класи спільно забезпечують повний цикл авторизації користувача через BankID. Вони також сприяють розділенню обов’язків: інтерфейс, логіка, стан і дані чітко відокремлені.

Компонент OptFlow (рис. 3.6) відповідає за двоетапну автентифікацію користувача через телефон. Спочатку він відображає форму введення номера телефону, і при його успішному введенні надсилає запит до API для ініціалізації сесії. Якщо запит проходить успішно, на інтерфейсі з’являється форма введення одноразового коду (OTP).

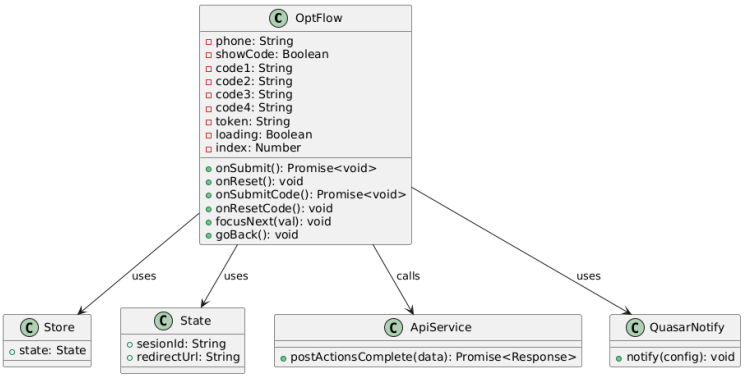


Рисунок 3.6 – Класів компонента OptFlow.vue

Введений код перевіряється шляхом повторного запиту до API з цим кодом та sessionId. Компонент використовує локальні змінні (phone, code1–4, loading, showCode) для управління станом. Для UX покращення застосовуються Quasar-компоненти q-input, q-btn, q-toggle і q-form. Повідомлення про помилки або успішні дії відображаються через $q.notify. Користувач може повернутися назад з форми OTP, натиснувши кнопку «Назад».

Компонент DiyaPage (рис. 3.7) Vue-компонент DiyaPage відповідає за генерацію та обробку QR-коду для авторизації через застосунок Дія. Він ініціалізує сокет-з’єднання для обробки callback-ів від сервера після сканування QR-коду користувачем. При монтуванні компонента запускається таймер зворотного відліку, який оновлює інтерфейс до завершення сесії. Якщо користувач натискає кнопку «Оновити QR-код», попереднє сокет-з’єднання розривається, генерується новий deepLink, і створюється нове сокет-з’єднання.

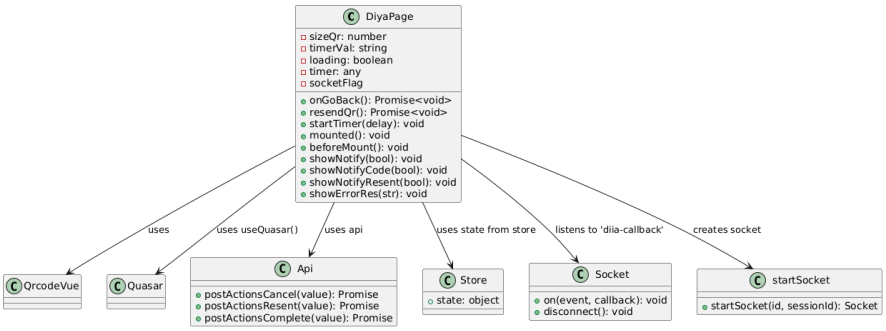


Рисунок 3.7 – Класів компонента DiyaPage.vue

Компонент підтримує адаптивність: для десктопних і мобільних пристроїв відображається відповідний варіант QR-коду або посилання. У методі onGoBack реалізована логіка скасування підпроцесу авторизації та повернення користувача назад. При отриманні повідомлення diia-callback з позитивним статусом, відбувається завершення кроку авторизації через API і редірект користувача. Якщо авторизація неуспішна, виводяться відповідні повідомлення про помилки. Для інформування користувача компонент використовує систему сповіщень Quasar. Додатково в компоненті є кілька методів для виведення сповіщень про статус дій, таких як успішна відправка або помилка [28].

3.3 Діаграма послідовностей Front-End частини

Sequence diagrams – це потужний інструмент, який дозволяє розробникам візуалізувати порядок взаємодії об’єктів у межах певного сценарію, що значно спрощує розуміння логіки системи. Вони допомагають уніфікувати бачення між учасниками команди: розробниками, аналітиками, дизайнерами та тестувальниками.

Завдяки послідовним діаграмам легше виявити логічні помилки або недоліки у взаємодії компонентів ще до початку реалізації. Вони сприяють плануванню архітектури і дають змогу задокументувати поведінку системи для майбутньої підтримки. Sequence diagrams особливо корисні в мікросервісній архітектурі або при складній взаємодії між клієнтом і сервером, де важливо показати обмін повідомленнями, черговість викликів та можливі умови переходу.

Попри переваги, використання sequence diagrams має і свої недоліки. Для складних систем такі діаграми можуть стати перевантаженими, важкими для читання та підтримки. Ручне оновлення діаграм при кожній зміні логіки забирає багато часу, що робить їх неактуальними, якщо не підтримувати дисципліну в команді.

Також діаграма не відображає деталі реалізації або структуру об’єктів, тож її потрібно поєднувати з іншими видами UML-діаграм (наприклад, class або

component diagrams). Вона може вводити в оману, якщо сценарій на діаграмі не охоплює всі можливі варіанти поведінки, наприклад, обробку помилок або паралельну роботу потоків. Крім того, для нових членів команди, які не знайомі з нотацією UML, такі діаграми можуть вимагати додаткового навчання [29].

Розглянемо детальніше діаграму послідовностей взаємодії класів компонентів у модулі OTP widget вебсистеми адаптивної автентифікації (рис. 3.8).

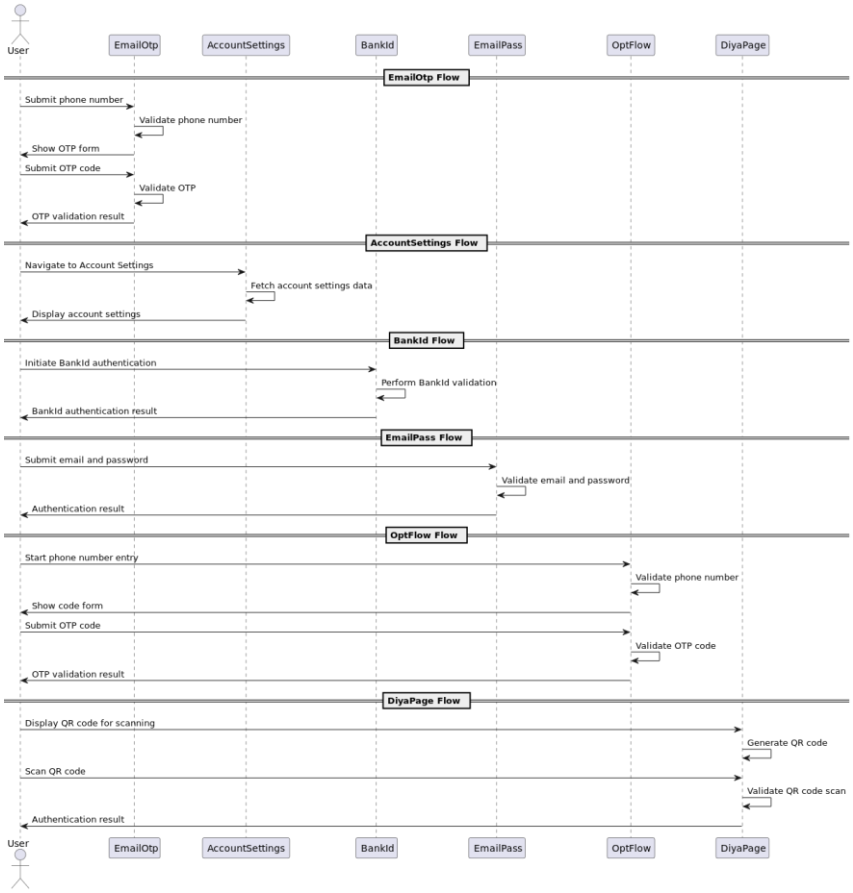


Рисунок 3.8 – Діаграма послідовностей взаємодії класів компонентів у модулі OTP widget вебсистеми адаптивної автентифікації

EmailOtp відповідає за перевірку одноразового коду, який надсилається на електронну пошту користувача. Користувач вводить отриманий код, і компонент надсилає його на сервер для валідації. У разі успішної перевірки користувач переходить до наступного кроку. AccountSettings відображає налаштування облікового запису після успішної авторизації. Компонент запитує дані користувача

з сервера і дозволяє редагування певних полів. Він також може бути кінцевою точкою деяких потоків авторизації.

BankId ініціює процес входу через BankID, переадресовуючи користувача на зовнішній сервіс банку. Після підтвердження особи банк повертає токен авторизації. Компонент приймає цей токен, перевіряє його на сервері й оновлює стан сесії. EmailPass відповідає за класичну аутентифікацію через логін і пароль. Після введення даних користувача виконується запит до API на підтвердження облікових даних. У разі успіху користувач авторизується в системі.OptFlow починає процес ідентифікації за номером телефону, надсилаючи OTP через SMS. Користувач вводить код, який перевіряється сервером на дійсність. Успішна валідація переводить користувача до наступного етапу або надає доступ.

DiyaPage генерує QR-код, який потрібно сканувати в застосунку Дія для входу. Після сканування додаток викликає callback, що активує перевірку даних на сервері. Успішна перевірка завершує автентифікацію, і користувача перенаправляють на потрібну сторінку.

3.4 Діаграма класів Back-End частини

Після проєктування Front-End частини переходимо до проєктування серверної частини. Необхідно визначити основні абстракції під час використання архітектурних патернів у різних Back-End фреймворках (рис. 3.9).

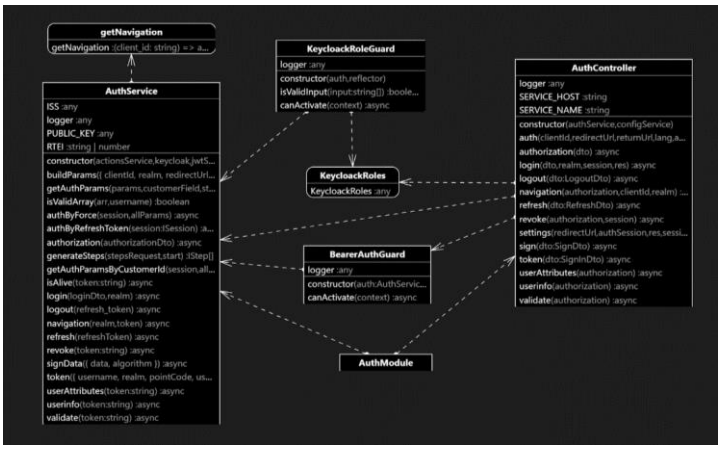


Рисунок 3.9 – Діаграма класів AuthModule вебсистеми адаптивної автентифікації

Клас AuthService містить методи, що реалізують основну бізнес-логіку аутентифікації та авторизації користувачів. Сюди входять методи для побудови параметрів запиту (buildParams), отримання параметрів аутентифікації (getAuthParams, getAuthParamsByCustomerId), перевірки валідності масивів (isValidArray), а також різні сценарії входу або через примусову аутентифікацію (authByForce), refresh token (authByRefreshToken) або через стандартний логін (login). Додатково сервіс дозволяє виконувати авторизацію (authorization), генерувати кроки автентифікації (generateSteps), перевіряти стан токена (isAlive), виходити з системи (logout), оновлювати токени (refresh), відкликати їх (revoke), підписувати дані (signData), отримувати токени (token), атрибути користувача (userAttributes), інформацію про користувача (userinfo) та валідувати токени (validate).

Клас AuthController відповідає за обробку HTTP-запитів, пов’язаних із аутентифікацією та авторизацією. Він надає методи для ініціалізації аутентифікації (auth), логіну (login), виходу з системи (logout), отримання навігаційних даних (navigation), оновлення токенів (refresh), відкликання сесій чи токенів (revoke), отримання налаштувань (settings), підпису даних (sign), отримання токенів (token), атрибутів користувача (userAttributes), інформації про користувача (userinfo) і валідації авторизації (validate).

Гарди KeycloakRoleGuard і BearerAuthGuard забезпечують захист маршрутів відповідно до ролей користувача та наявності валідного токена. KeycloakRoleGuard містить методи для перевірки валідності вхідних ролей (isValidInput) і визначення, чи може користувач отримати доступ до ресурсу (canActivate), використовуючи інформацію про ролі (KeycloakRoles). BearerAuthGuard перевіряє наявність і дійсність токена за допомогою методу canActivate, використовуючи функціонал AuthService для авторизації запитів.

Сервіс getNavigation надає допоміжний метод getNavigation, який повертає навігаційні дані для певного клієнта за його ідентифікатором. Всі ці компоненти

об’єднуються в модулі AuthModule, що дозволяє централізовано керувати залежностями, маршрутами та логікою аутентифікації у всій системі.

Нижче наведено діаграму класів, що демонструє модуль дій (ActionsModule), який координує взаємодію між різними сервісами, відповідальними за обробку дій користувача, аутентифікацію та OTP (одноразові паролі) (рис. 3.10).

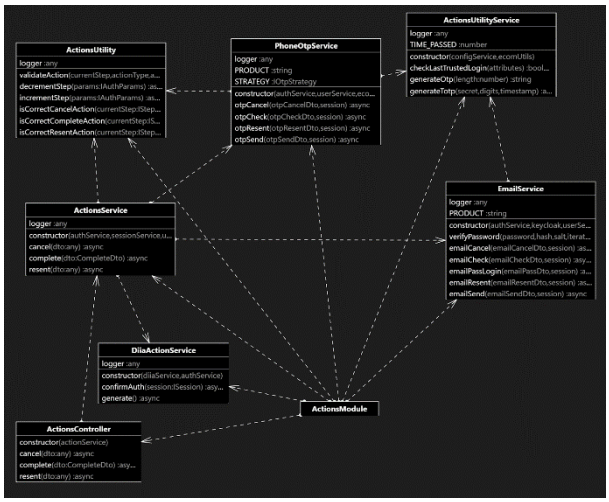


Рисунок 3.10 – Діаграма класів ActionsModule вебсистеми адаптивної автентифікації

Клас ActionsController виступає як точка входу для зовнішніх запитів і викликає відповідні методи сервісу ActionsService, що реалізує бізнес-логіку для скасування (cancel), завершення (complete) та повторної відправки дії (resent). ActionsService взаємодіє з іншими службами, як-от PhoneOtpService, EmailService, DiiaActionService для виконання конкретних операцій, таких як відправка OTP по телефону або електронній пошті. Клас ActionsUtility використовується для перевірки правильності кроків виконання дій, зокрема, чи можна скасувати, завершити або повторити дію на поточному етапі.

Клас PhoneOtpService реалізує стратегію для OTP по телефону (згідно з інтерфейсом IOtpStrategy) і надає методи для надсилання, перевірки, повторного надсилання або скасування OTP. EmailService, зі свого боку, виконує аналогічну роль, але для обробки OTP через email дозволяє перевіряти паролі, надсилати або скасовувати OTP, перевіряти введені коди тощо. Клас DiiaActionService має справу

з інтеграцією з державним застосунком «Дія», а саме реалізує підтвердження дії та генерацію відповідних даних. Клас ActionsUtilityService надає допоміжні функції, зокрема, генерацію OTP-кодів та перевірку останніх надійних входів користувача. Усі сервіси використовують логер для журналювання операцій, що полегшує відстеження та діагностику.

Клас ActionsModule слугує як модуль для зв’язку між цими компонентами в межах одного функціонального блоку. Він імпортує всі потрібні сервіси, включно з ActionsUtility, ActionsUtilityService, PhoneOtpService, EmailService, DiiaActionService, та об’єднує їх для централізованої роботи. Основна логіка полягає в делегуванні задач відповідним сервісам згідно з типом дії та каналом комунікації (телефон, email або застосунок «Дія»). Наприклад, при спробі користувача скасувати OTP, ActionsService перевіряє правильність поточного кроку через ActionsUtility, після чого викликає PhoneOtpService.otpCancel або EmailService.emailCancel, залежно від каналу. Ця структура модульна, розширювана і підтримує принципи розділення відповідальності, що забезпечує легкість підтримки та масштабування системи.

Діаграма класів описує модуль CommunicationModule, який забезпечує відправку повідомлень і електронних листів через зовнішні сервіси (рис. 3.11).

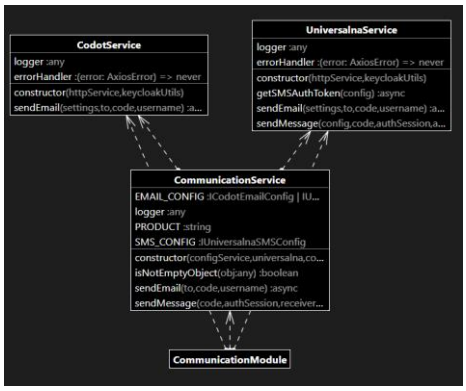


Рисунок 3.11 – Діаграма класів CommunicationModule вебсистеми адаптивної автентифікації

Центральним елементом є клас CommunicationService, який інкапсулює логіку взаємодії з цими зовнішніми системами через методи sendEmail та

sendMessage. Сервіс має конфігураційні властивості EMAIL_CONFIG і SMS_CONFIG, що визначають, через який провайдер буде виконуватись надсилання. Метод isNotEmptyObject використовується для перевірки валідності вхідних об’єктів перед передачею у відповідні сервіси. Логер та властивість PRODUCT дозволяють відстежувати процеси та налаштовувати поведінку в залежності від контексту продукту.

Клас CodotService відповідає за надсилання електронної пошти через сервіс Codot і містить метод sendEmail, який приймає налаштування, адресу одержувача, код і ім’я користувача. Він також має обробник помилок errorHandler, що реагує на помилки типу AxiosError – типові для HTTP-запитів. UniversalnaService, в свою чергу, надає ширший спектр функціональності: крім sendEmail, він реалізує метод sendMessage, що дозволяє надсилати SMS, а також getSMSAuthToken, який використовується для автентифікації перед відправкою SMS. Обидва сервіси підключаються до CommunicationService, який делегує їм реальні дії в залежності від конфігурації. Це дозволяє централізовано управляти всіма повідомленнями через один сервіс, не турбуючись про особливості конкретних провайдерів.

Уся архітектура об’єднується у модулі CommunicationModule, який слугує точкою з’єднання між залежностями та основним сервісом. Завдяки цьому модулю CommunicationService отримує необхідні сервіси (CodotService, UniversalnaService) та конфігурацію для коректного функціонування. Логіка організована так, щоб можна забезпечити максимальну гнучкість та легко змінювати провайдерів або додавати нові без суттєвих змін у коді CommunicationService. Це також дозволяє реалізувати fallback-механізми, коли при відмові одного сервісу, інший може його замінити. Подібна структура відповідає принципам інверсії залежностей і відкритості/закритості (SOLID), що робить систему масштабованою та зручною в обслуговуванні. Таким чином, система чітко поділена на шари відповідальності з мінімальним зв’язуванням, що спрощує її розширення та підтримку [30].

Основним класом який реалізує базові методи для моніторингу стану системи є HealthController, зокрема, check() та testEndpoint(). Клас містить дві важливі

константи IDENTITY_SERVICE_PORT та SERVICE_NAME, що дозволяють ідентифікувати сервіс серед інших мікросервісів у системі. Конструктор приймає кілька залежностей, пов’язаних із моніторингом: наприклад, health, http, memory, disk, тощо. Дані сервіси дозволяють виконати комплексну оцінку стану та перевіряється доступність HTTP-з’єднань, використання пам’яті та стан файлової системи.

Метод check() викликає всі доступні перевірки для визначення, чи система здорова (healthy), і повертає агрегований результат. Цей метод може використовуватись сторонніми сервісами або інструментами моніторингу (наприклад, Prometheus або Kubernetes liveness probes). Метод testEndpoint() виконує простішу перевірку: повертає статус 200 OK для підтвердження, що сервіс працює і приймає запити. Це дозволяє розділити повну перевірку і базову, а саме: одна для внутрішньої діагностики, а інша для зовнішніх health-чеків. Такий підхід є стандартом у сучасних мікросервісних архітектурах.

Модуль HealthModule інкапсулює логіку контролера HealthController, забезпечуючи його залежностями та дозволяючи інтеграцію з іншими модулями системи. Його легко імпортувати у будь-який інший модуль, щоб додати базову діагностику сервісу. Подібна реалізація дозволяє централізовано слідкувати за здоров’ям усієї системи. Вона також зменшує ризик непомічених збоїв, оскільки дозволяє виявляти проблеми на рівні ресурсів ще до критичного падіння. Крім того, вона корисна для DevOps-команд під час CI/CD-процесів. Таким чином, HealthModule є фундаментальним елементом надійності та стабільності системи.

На діаграмі класів зображено взаємодію між модулями та класами в системі, яка реалізує електронну ідентифікацію через сервіси «Дія» та BankID (рис. 3.12).

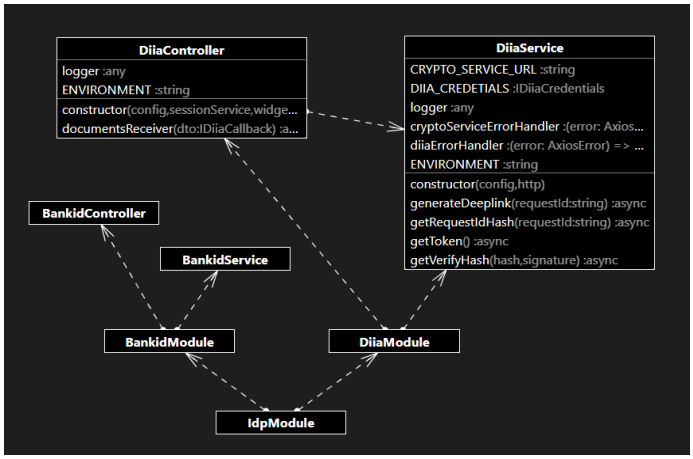


Рисунок 3.12 – Діаграма класів IdpModule вебсистеми адаптивної автентифікації

Клас DiiaController відповідає за прийом даних від клієнтів і виклик відповідних сервісних методів. Метод documentsReceiver(dto: IDiiaCallback) приймає DTO (Data Transfer Object), який містить дані, отримані після авторизації користувача через сервіс Дія. У цьому методі відбувається перевірка підпису, розшифрування даних та їх обробка або збереження. Конструктор класу ініціалізує необхідні сервіси, зокрема, sessionService і widgetService, що використовуються для збереження сесій або показу інтерактивних елементів. Залежність від DiiaService дозволяє контролеру викликати функції глибокого посилання або верифікації. Змінна logger використовується для логування подій і помилок, а ENVIRONMENT для визначення конфігурацій (наприклад, продакшн чи тестове середовище). Контролер виступає точкою входу для зовнішніх запитів у модулі DiiaModule, що входить до загального IdpModule (модуля ідентифікації). Таким чином, DiiaController концентрує логіку маршрутизації запитів і делегування відповідним сервісам.

Клас DiiaService є ядром обробки логіки, пов’язаної із сервісом «Дія». Метод generateDeeplink створює URL-посилання, яке клієнт може використовувати для запуску ідентифікації в застосунку «Дія», передаючи унікальний requestId. Метод getRequestIdHash створює хеш для запиту, що забезпечує перевірку автентичності клієнтського запиту. Метод getToken повертає токен для доступу до захищених даних або API, а getVerifyHash перевіряє хеш-підпис, забезпечуючи

криптографічну перевірку достовірності даних. Усі методи є асинхронними, тобто підтримують неблокуючі виклики до зовнішніх API, зокрема HTTP-запити до «Дії». CryptoServiceErrorHandler та diiaErrorHandler – це функції обробки помилок, які гарантують, що система адекватно реагує на збої у зв'язку або невірні відповіді. Конструктор ініціалізує конфігурацію та HTTP-клієнт, який взаємодіє з зовнішніми сервісами. Змінні CRYPTO_SERVICE_URL і DIIA_CREDENTIALS містять необхідну інформацію для авторизації та взаємодії з API. Цей сервіс є ключовим компонентом модуля DiiaModule, який у свою чергу забезпечує сервісну логіку для контролера та інших частин системи.

На відміну від модуля DiiaModule, BankidModule реалізує логіку для автентифікації через BankID. BankidController і BankidService працюють подібно до компонентів Diia, але для іншого провайдера ідентифікації. Обидва модулі (Diia і Bankid) інкапсулюються всередині IdpModule, який є головним інтерфейсом для автентифікації користувачів у системі. Це дозволяє системі бути універсальною і гнучкою, тобто вона може обробляти запити як від користувачів із застосунком «Дія», так і від тих, хто використовує BankID. Взаємодія між модулями реалізована через залежності, які зображено пунктирними стрілками. Загалом, така архітектура забезпечує масштабованість, безпеку та розширюваність системи, дозволяючи при необхідності додати нові провайдери ідентифікації. Вона дотримується принципів розділення обов'язків: контролери обробляють запити, сервіси – бізнес-логіку, а модулі – структурну організацію. Завдяки цьому код залишається зручним для читання, легко тестується і підтримується.

На діаграмі класів модуля WidgetModule показана організація логіки взаємодії з віджетами: невеликими інтерактивними компонентами частиною інтерфейсу або сесій користувача (рис. 3.13).

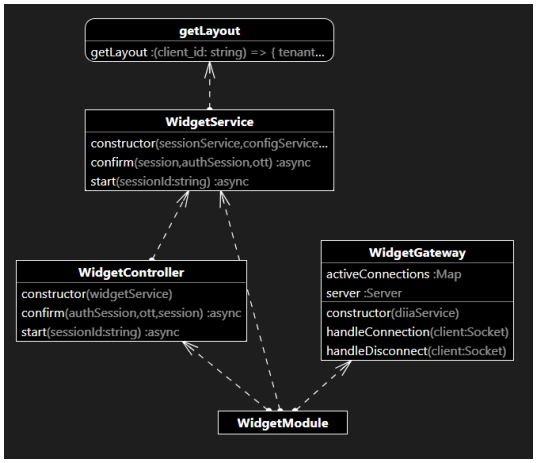


Рисунок 3.13 – Діаграма класів WidgetModule вебсистеми адаптивної автентифікації

Клас WidgetService є центральною ланкою бізнес-логіки, що координує запуск віджета (start(sessionId: string)) і підтвердження сесії (confirm(session, authSession, ott)). Конструктор цього сервісу приймає кілька залежностей, зокрема сервіси конфігурацій і сесій, що вказує на широке використання в інфраструктурі. Його асинхронні методи обробляють взаємодії в рамках сесії, використовуючи авторизаційні токени або тимчасові коди OTT. Цей сервіс також використовує метод getLayout, що отримує конфігурацію або зовнішній вигляд віджета для певного клієнта. Таким чином, WidgetService виконує роль посередника між сесією, зовнішнім виглядом і підтвердженням ідентичності. Його дизайн відображає гнучкість і підтримку багатьох потоків взаємодії. Це дозволяє системі адаптуватися до різних сценаріїв, таких як мобільні інтерфейси або інтеграція із зовнішніми клієнтами.

WidgetController – це компонент, який відповідає за виклик публічних точок доступу для ініціалізації та підтвердження віджетів. Він викликає відповідні методи WidgetService у відповідь на зовнішні запити, зокрема start(sessionId) для запуску віджета та confirm(authSession, ott, session) для перевірки автентичності сесії. Контролер ізолює HTTP або WebSocket запити від логіки обробки даних, що дозволяє чітко розмежувати рівні архітектури. Його єдиною залежністю є WidgetService, що робить його простим для тестування та розширення. Таке

розмежування дозволяє централізовано керувати потоком автентифікації користувача, починаючи з отримання OTT до повного запуску віджета. Таким чином, контролер обслуговує зовнішній API або маршрути системи. Його робота фокусується на отриманні та перевірці запитів, делегуючи бізнес-логіку в сервіс. Це відповідає принципам REST/GraphQL або Socket API, де контролер лише обслуговує запити й відповіді. В результаті цей клас забезпечує чисту й логічну взаємодію з зовнішнім середовищем.

WidgetGateway відповідає за обробку WebSocket-з'єднань, тобто двосторонньої комунікації між клієнтом і сервером. Він керує підключенням (handleConnection(client: Socket)) і відключенням клієнтів (`handleDisconnect(client: Socket)), а також має змінну activeConnections, яка зберігає поточні з'єднання. Це особливо важливо для динамічних віджетів, які повинні реагувати на дії користувача в реальному часі. Крім того, конструктор класу приймає залежність diiaService, що вказує на можливу інтеграцію з платформою державних послуг «Дія», наприклад, для автентифікації або передачі документів. Gateway дозволяє легко розширювати взаємодію через події, без потреби в ручному контролі сесій. Архітектура враховує живу взаємодію через сокети й асоційовану логіку керування підключеннями. Таким чином, WidgetGateway – це транспортний рівень комунікації в реальному часі. Його інтеграція з іншими компонентами модуля гарантує, що дані користувача залишаються синхронізованими й захищеними. Це критично для сервісів, що потребують миттєвого зворотного зв'язку або взаємодії, таких як електронні кабінети або віджети цифрової ідентифікації.

3.5 Діаграма послідовностей Back-End частини

AuthModule є центральним компонентом системи, що відповідає за автентифікацію та авторизацію користувачів. Після успішної аутентифікації через AuthController, токен користувача передається до інших модулів для підтвердження доступу. AuthService здійснює валідацію токенів, забезпечуючи надійний захист доступу до маршрутів. Гарди BearerAuthGuard та

KeycloakRoleGuard інтегруються з іншими модулями для перевірки прав доступу. Таким чином, AuthModule виконує роль шлюзу безпеки для всієї системи [31].

ActionsModule взаємодіє з AuthModule для перевірки повноважень користувача перед виконанням дій. Наприклад, перед запуском певної користувацької дії, ActionsModule перевіряє, чи має користувач відповідні ролі або дозволи. Для цього використовується валідований токен, який перевіряється через AuthService або відповідні гарди. Це дозволяє гарантувати, що дії можуть виконувати лише авторизовані користувачі. Таким чином, ActionsModule покладається на AuthModule як на механізм контролю доступу.

CommunicationModule використовує AuthModule для аутентифікації відправника повідомлення та підтвердження його прав. Коли ініціюється дія в ActionsModule, вона може викликати надсилання повідомлення через CommunicationModule. Наприклад, після успішної дії система може надіслати push-повідомлення або email користувачеві. CommunicationModule взаємодіє з AuthService для отримання ідентифікатора користувача і підтвердження автентичності. Цей модуль підтримує зворотний зв’язок і підвищує інтерактивність системи.

HealthModule працює незалежно, але може використовуватись для перевірки стану як AuthModule, так і інших модулів. Через метод check() HealthController може ініціювати перевірку доступності зовнішніх сервісів, у тому числі AuthService чи CommunicationService. Якщо один з модулів недоступний, HealthModule повідомить про деградацію сервісу. Він також може інтегруватися з ActionsModule, щоб призупинити виконання критичних дій у разі виявлення проблем. Таким чином, HealthModule забезпечує стабільність і надійність взаємодії між модулями.

На рисунку 3.14 зображено діаграму послідовності взаємодії основних модулів серверної частина вебсистеми, що розробляється.

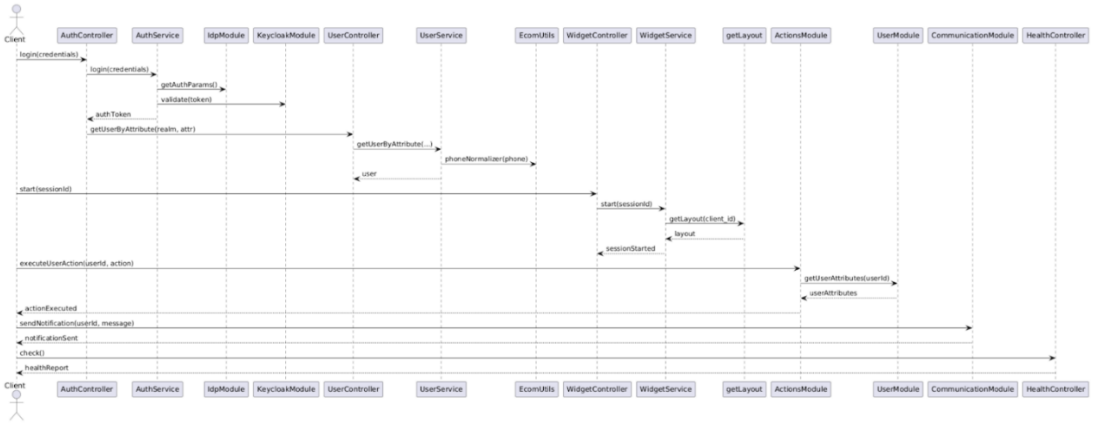


Рисунок 3.14 – Діаграма послідовності взаємодії основних модулів серверної частина вебсистеми адаптивної автентифікації

IdpModule (Identity Provider Module) виступає як проміжна ланка між зовнішніми провайдерами ідентифікації та внутрішніми модулями, зокрема KeycloakModule і UserModule. Його основна роль координувати процес автентифікації користувачів через зовнішні IdP (як-от BankID, Дія, тощо). Після успішної автентифікації через зовнішній IdP, IdpModule передає отримані ідентифікаційні дані в KeycloakModule для створення або оновлення облікового запису. Він також може передавати цю інформацію до UserModule для подальшої обробки атрибутів користувача. Таким чином, IdpModule забезпечує гнучке підключення до сторонніх джерел автентифікації.

KeycloakModule інтегрується з Keycloak-сервером для керування сесіями, токенами та ролями користувачів. Після отримання даних від IdpModule, KeycloakModule створює нового користувача або оновлює існуючого в Keycloak. Він також взаємодіє з UserModule через Keycloak SDK для витягування атрибутів користувачів, груп, ролей та іншої інформації. KeycloakModule надає централізований доступ до сервісів автентифікації та авторизації. Його функціональність є критичною для забезпечення безпеки в усьому додатку.

WidgetModule забезпечує взаємодію з кінцевими користувачами через віджети, що можуть запускати сесії, підтвердження чи інші дії. Він використовує дані з UserModule для персоналізації або перевірки доступу користувачів до функціоналу. Наприклад, перед початком сесії WidgetService може звернутися до

UserService для перевірки, чи має користувач відповідні атрибути або чи він автентифікований. У разі використання зовнішнього IdP, WidgetModule через посередництво IdpModule та KeycloakModule підтверджує справжність користувача. Таким чином, WidgetModule залежить від надійної інтеграції з усіма іншими модулями для реалізації повноцінного функціоналу.

3.6 Діаграма розгортання

UML діаграми розгортання моделюють фізичну архітектуру системи. Діаграми розгортання показують взаємозв'язки між програмним і апаратним компонентами в системі та фізичним розподілом обробки.

Діаграми розгортання, які ви зазвичай готуєте на етапі реалізації розробки, показують фізичне розташування вузлів у розподіленій системі, артефакти, які зберігаються на кожному вузлі, а також компоненти та інші елементи, які реалізують артефакти (рис. 3.15). Вузли представляють апаратні пристрої, такі як комп'ютери, датчики та принтери, а також інші пристрої, які підтримують середовище виконання системи. Шляхи зв'язку та зв'язки розгортання моделюють зв'язки в системі [32].

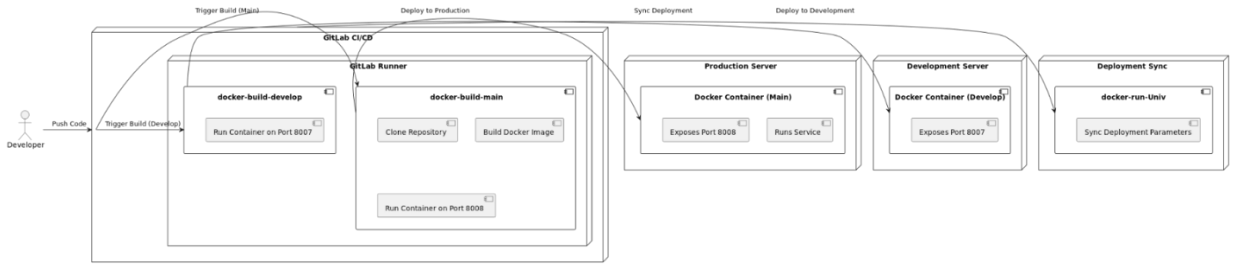


Рисунок 3.15 – Діаграма розгортання для вебсистеми адаптивної автентифікації

У діаграмі розгортання описано процес побудови та розгортання Docker-контейнерів у рамках CI/CD-архітектури GitLab. Конфігурація визначає змінну GIT_STRATEGY: clone, яка гарантує, що кожна джоба завжди виконується на свіжій копії репозиторію. Це дозволяє уникнути проблем, пов'язаних із кешованими змінами або залишками попередніх виконань. Основна логіка побудови реалізована у двох джобах: docker-build-main і docker-build-develop, які

відповідають за продакшн та девелопмент-середовища відповідно. Усі джоби виконуються на runner-і з тегом gitrunner.codot.pro, що може вказувати на власний self-hosted runner, налаштований на компанійному сервері. Docker-build-main спрацьовує виключно при зміні в гілці main, тобто орієнтований на production-збірку. У джобі спочатку визначаються ключові параметри, зокрема, зовнішні порти для HTTP (8008) і WebSocket (5008), а також динамічна назва контейнера на основі змінних середовища CI/CD ($CI_PROJECT_NAME, $CI_ENVIRONMENT_NAME, $CI_PROJECT_ID). Перед побудовою нової версії сервісу система перевіряє, чи існує контейнер з такою ж назвою, і, якщо так, видаляє його через docker rm -f, щоб уникнути конфлікту імен або портів. Далі копіюються змінні конфігурації та сертифікати (змінна SETTINGS, приватний та публічний ключі) в робочу директорію контейнера. Після цього запускається команда docker build з аргументами, що задають ім’я сервісу і хост, а результатом є створений контейнер, який далі запускається з відповідними змінними середовища за допомогою docker run.

Аналогічно до docker-build-main, джоб docker-build-develop виконує майже ті ж самі кроки, але в іншому середовищі – develop, з іншими портами (8007 і 5007). Це дозволяє одночасно підтримувати та тестувати декілька версій сервісу, не створюючи конфліктів між середовищами. Назва контейнера також формується динамічно на основі гілки і проєкту, забезпечуючи унікальність. Таким чином, кожна гілка має власне ізольоване середовище, що відповідає найкращим практикам розробки, тестування і релізу. Механізм копіювання конфігурацій та ключів передбачається як стандарт безпеки і гнучкості, тобто зміна сертифікатів або конфігурацій може виконуватись без втручання в кодову базу.

Висновки до розділу 3

Проєктування є першим етапом процесу розробки програмних проєктів. За допомогою графу залежностей, діаграм класів та діаграм послідовностей. UML проєктування надає значні переваги при розробці модулів, таких як AuthModule,

ActionsModule, CommunicationModule, HealthModule, IdpModule, KeycloakModule, UserModule та WidgetModule. Завдяки діаграмам класів, компонентів та взаємодій розробники можуть наочно бачити структуру та зв'язки між модулями. Це полегшує планування інтеграцій, наприклад, як AuthModule взаємодіє з KeycloakModule або IdpModule. UML допомагає виявити потенційні проблеми на ранньому етапі, зменшуючи ризики помилок у логіці або інтерфейсах. Завдяки стандартній нотації UML покращується комунікація між командами, особливо коли в проєкті беруть участь аналітики, тестувальники та розробники. Документування за допомогою UML полегшує супровід і масштабування системи. Діаграми станів та послідовностей корисні для моделювання поведінки таких модулів, як UserModule та CommunicationModule. UML також сприяє кращому розумінню бізнес-логіки, наприклад, у ActionsModule чи WidgetModule. У модулі HealthModule UML може показати критичні точки моніторингу та зв'язки з іншими сервісами. Загалом, UML-аналіз допомагає створити узгоджену, масштабовану й підтримувану архітектуру.

Для покращення надійності та масштабованості CI-конфігурації доцільно внести кілька оптимізацій. По-перше, варто додати окрему стадію test перед build, у якій будуть запускатися юніт-тести або лінтери, що дозволить виявити помилки до створення Docker-образу. По-друге, замість ручного видалення контейнерів через docker ps/grep, краще використовувати docker container prune або мітки з автоматичним очищенням, щоб уникнути конфліктів при одночасному запуску кількох джоб. По-третє, зберігати створені Docker-образи в Docker Registry (наприклад, GitLab Container Registry), що дозволить використовувати їх повторно без повторної побудови. Для цього потрібно додати крок docker push після docker build, а в джобах – налаштувати логін до реєстру через змінні CI (CI_REGISTRY, CI_REGISTRY_USER). Також варто розділити змінні середовища у .gitlab-ci.yml на глобальні та середовищеві (environment: develop, environment: prod) для зручності підтримки. Замість копіювання файлів сертифікатів у кожній джобі, їх можна зберігати у GitLab CI Variables як Base64 і розпаковувати всередині

контейнера, що підвищить безпеку. Бажано також винести повторювану логіку побудови контейнера в окремий шаблон YAML (include), щоб уникнути дублювання між main і develop. Також можна використовувати артефакти або кешування, щоб зберігати результати проміжних кроків. Для зручності дебагу доцільно логувати версію іміджу (наприклад, git commit hash або timestamp) під час побудови. Нарешті, додавання умов when: manual або allow_failure: true до нестабільних джоб допомогає зменшити кількість зупинених пайплайнів через незначні помилки.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис програмної реалізації

Під час взаємодії користувача з вебсистемою, браузер надсилає HTTP-запити на сервер для отримання необхідного вмісту, відправлення даних або виконання інших дій. Такі запити зазвичай містять кілька заголовків – це пари «ключ-значення», що описують параметри запиту. User-Agent string (скорочено UAS) є HTTP-заголовком, який описує програмне забезпечення, яке діє від імені користувача (рис. 4.1).

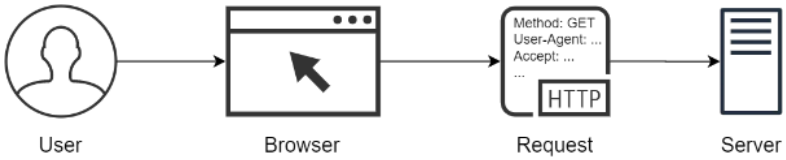


Рисунок 4.1 – UAS описує властивості браузера

Початкове призначення UAS полягало в узгодженні контенту, тобто в механізмі визначення найкращого контенту для користувача залежно від інформації у відповідному UAS (наприклад, формат зображення, мова документа, текстове кодування тощо). Ця інформація зазвичай містить дані про середовище, в якому працює браузер (пристрій, операційна система та її версія, регіональні налаштування), движок браузера та розкладки та їх версії та інше (рис. 4.2).

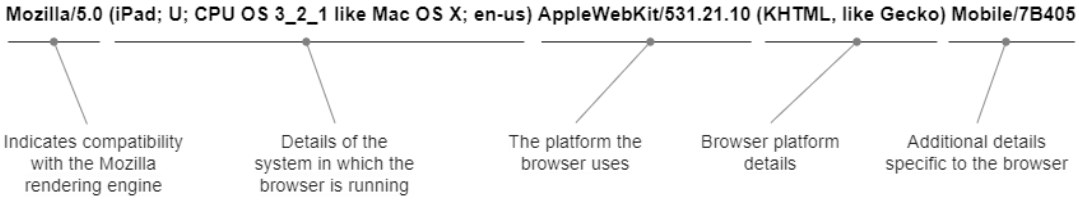


Рисунок 4.2 – Приклад типового UAS та його елементів

UAS має багато практичних застосувань. Найпоширенішим є вебаналітика, тобто звітність про склад трафіку для оптимізації ефективності вебсайту. Інше застосування – управління вебтрафіком, що включає блокування небажаних ботів, зменшення навантаження на сайт від непотрібних відвідувачів, запобігання клікфорду (шахрайські кліки) та інші подібні завдання. Завдяки великій кількості

інформації, яку вони містять, UAS також можуть слугувати джерелом даних для застосування методів машинного навчання.

Елементи User-Agent String (UAS) можуть слугувати корисними проксі-ознаками характеристик користувача, таких як стиль життя, технічна обізнаність або навіть рівень добробуту. Наприклад, користувач, який зазвичай відвідує вебсайт із преміального мобільного пристрою, ймовірно, відрізняється від користувача, який заходить на той самий сайт за допомогою браузера Internet Explorer на настільному комп'ютері під керуванням Windows XP. Наявність проксі-ознак таких характеристик, отриманих на основі UAS, є особливо цінною у випадках, коли відсутня будь-яка інша демографічна інформація про користувача (наприклад, коли сайт відвідує нова, неідентифікована особа).

Відносно простим способом створення ознак для алгоритмів машинного навчання з UAS є застосування парсера, виділення окремих елементів та подальше one-hot кодування. Такий підхід може бути ефективним у простих сценаріях, коли потрібно трансформувати лише найвищі рівні та легко впізнавані елементи UAS у машинозчитувані ознаки. Наприклад, досить просто визначити тип пристрою користувача (мобільний, ПК, сервер тощо). Для такого інженерінгу ознак існує низка високоякісних парсерів, що базуються на регулярних виразах (наприклад, проєкт ua-parser та його реалізації для різних мов програмування).

Однак вищезгаданий підхід швидко стає непридатним, якщо виникає потреба враховувати усі елементи UAS для максимального вилучення інформації. Це пов'язано з двома ключовими причинами. Першою причиною є те, що існуючі рекомендації щодо форматування заголовків User-Agent не є обов'язковими до виконання, і в реальних умовах можна зіткнутися з великою різноманітністю специфікацій UAS. Унаслідок цього забезпечити узгоджений парсинг таких рядків надзвичайно складно. Більше того, щодня з'являються нові пристрої, версії операційних систем та браузерів, що ускладнює підтримку високоякісних парсерів. Другою причиною є те, що кількість можливих елементів UAS і їхніх комбінацій є астрономічно великою. Навіть якщо їх вдалося б закодувати у one-hot форматі,

утворена матриця даних була б надзвичайно розрідженою (sparse) та надто великою для обробки на комп'ютерах, які зазвичай використовують фахівці з аналізу даних.

Для подолання цих викликів можна застосувати метод зниження розмірності (dimensionality reduction) та представити UAS у вигляді векторів фіксованої довжини, мінімізуючи втрати інформації. Ця ідея не є новою, і оскільки UAS – це, по суті, текстові рядки, таке представлення можна реалізувати за допомогою методів обробки природної мови (NLP). Алгоритм fastText, розроблений дослідниками з Facebook, забезпечує особливо корисні результати. Цей метод має кілька практичних переваг:

– не потребує багато даних, тобто якісна модель може бути навчена навіть на кількох тисячах прикладів;

– добре працює з короткими та структурованими документами, такими як UAS;

– швидко навчається;

– ефективно обробляє «слова поза словником», тобто здатний створювати осмислені векторні подання навіть для тих рядків, які не зустрічались під час навчання моделі.

Офіційна реалізація алгоритму fastText доступна у вигляді окремої бібліотеки мовою C++ та обгортки для Python. Обидві версії добре задокументовані, легко встановлюються та використовуються. Крім того, популярна бібліотека Python gensim має власну реалізацію цього алгоритму. Нижче опишемо навчання моделі fastText у середовищі R.

Існує декілька обгорток для мови R, що працюють з C++ бібліотекою fastText (зокрема, fastText, fastrtext та fastTextR). Втім, найпростішим способом навчання і використання моделей fastText у R, безперечно, є використання офіційної Python-реалізації через пакет reticulate. Імпортувати Python-модуль fasttext у середовище R можна наступним чином (рис. 4.3):

```
# Install `fasttext` first
# (see https://fasttext.cc/docs/en/support.html)

# Load the `reticulate` package
# (install first, if needed):
require(reticulate)

# Make sure `fasttext` is available to R:
py_module_available("fasttext")
## [1] TRUE

# Import `fasttext`:
ft <- import("fasttext")

# Then call the required methods using
# the standard `$` notation,
# e.g.: `ft$train_supervised()`
```

Рисунок 4.3 – Імпортування Python-модуля fasttext у середовище R

Навчання моделі базуються на вибірці з 200,000 унікальних рядків User-Agent з бази даних whatismybrowser.com (рис. 4.4).

```
mozilla/5.0 (iphone; cpu iphone os 10_3 like mac os x) applewebkit/602.1.50 (khtml, like gecko) crios/56.0.2924.75 mobile/14e5239e
safari/602.1 ruxitsynthetic/1.0 v256748272 t209530018064740643 6
mozilla/5.0 (iphone; cpu iphone os 12_4_3 like mac os x) applewebkit/605.1.15 (khtml, like gecko) mobile/15e148 instagram 123.0.0.24.115
(iphone6,2; ios 12_4_3; nl_nl; nl-nl; scale=2.00; 640x1136; 188362626) nw/1
mozilla/5.0 (iphone; cpu iphone os 10_2_1 like mac os x) applewebkit/602.4.6 (khtml, like gecko) mobile/14d27
[fban/fbios;fbav/84.0.0.36.37;fbbv/52496792;fbdv/iphone7,1;fbmd/iphone;fbsn/ios;fbsv/10.2.1;fbss/3;fbcr/3ireland;fbid/phone;fblc/en_gb;fbop/5
;fbrv/0]
mozilla/5.0 (linux; u; android 8.1.0; fr-fr; redmi s2 build/opm1.171019.011) applewebkit/537.36 (khtml, like gecko) version/4.0
chrome/61.0.3163.128 mobile safari/537.36 xiaomi/miuibrowser/10.8.3-g
mozilla/5.0 (ipad; cpu os 10_2_1 like mac os x) applewebkit/602.4.6 (khtml, like gecko) mobile/14d27
[fban/fbios;fbav/88.0.0.60.70;fbbv/55239788;fbdv/ipad6,4;fbmd/ipad;fbsn/ios;fbsv/10.2.1;fbss/2;fbcr/sprint;fbid/tablet;fblc/en_us;fbop/5;fbrv
/0]
mozilla/5.0 (ipad; cpu os 9_0 like mac os x) applewebkit/601.1.46 (khtml, like gecko) mobile/13a344
[fban/fbios;fbav/8.0.0.28.18;fbbv/1665515;fbdv/ipad2,1;fbmd/ipad;fbsn/iphone os;fbsv/9.0;fbss/1; fbcr/;fbid/tablet;fblc/en_us;fbop/1]
mozilla/5.0 (linux; android 9; oneplus a6000 build/pkq1.180716.001; wv) applewebkit/537.36 (khtml, like gecko) version/4.0
chrome/76.0.3809.111 mobile safari/537.36 [fb_iab/fb4a;fbav/233.0.0.36.117;]
mozilla/5.0 (linux; android 5.1.1; build/lmy49i; wv) applewebkit/537.36 (khtml, like gecko) version/4.0 chrome/59.0.3071.125 safari/537.36
mozilla/5.0 (linux; android 7.0; sm-g925f build/nrd90m; wv) applewebkit/537.36 (khtml, like gecko) version/4.0 chrome/74.0.3729.136 mobile
safari/537.36 jssdk/2 topbuzz/11.2.3 nettype/4g
mozilla/5.0 (macintosh; intel mac os x 10.11; rv:46.0) gecko/20100101 firefox/46.0 | hardenize/v1.1168.1 (https://www.hardenize.com)
```

Рисунок 4.4 – Приклад підмножин UAS

Дані зберігаються у текстовому файлі у форматі *plain text*, де кожен рядок містить один UAS, нормалізований до нижнього регістру, жодної іншої попередньої обробки не застосовується.

Навчання некерованої (unsupervised) моделі fastText в R є надзвичайно простим, достатньо викликати команду (рис. 4.5).

```
m_unsup <- ft$train_unsupervised(
   input = "./data/train_data_unsup.txt",
   model = "skipgram",
   lr = 0.05,
   dim = 32L, # vector dimension
   ws = 3L,
   minCount = 1L,
   minn = 2L,
   maxn = 6L,
   neg = 3L,
   wordNgrams = 2L,
   loss = "ns",
   epoch = 100L,
   thread = 10L
)
```

Рисунок 4.5 – Навчання некерованої моделі fastText

Аргумент dim у наведеній команді визначає розмірність простору векторних подань. Кожен UAS трансформується у вектор розмірності 32. Інші параметри

керують процесом навчання моделі (lr – швидкість навчання, loss – функція втрат, epoch – кількість ітерацій у процесі тренування тощо). Повна інформація щодо параметрів міститься у офіційній документації fastText..

Після того як модель навчена, обчислюються векторні подання для нових випадків з набору потенційних кібератак на фінансові установи. Нижче показано використання ключової функції – m_unsup$get_sentence_vector(), яка повертає середнє значення векторів для всіх елементів, що складають заданий UAS (рис. 4.6).

```
test_data <- readLines("./data/test_data_unsup.txt")

emb_unsup <- test_data %>%
  lapply(., function(x) {
    m_unsup$get_sentence_vector(text = x) %>%
      t(.) %>% as.data.frame(.)
  }) %>%
  bind_rows(.) %>%
  setNames(., paste0("f", 1:32))

# Printing out the first 5 values
# of the vectors (of size 32)
# that represent the first 3 UAS
# from the test set:

emb_unsup[1:3, 1:5]
##      f1       f2    f3    f4      f5
## 1 0.197 -0.03726 0.147 0.153  0.0423
## 2 0.182  0.00307 0.147 0.101  0.0326
## 3 0.101 -0.28220 0.189 0.202 -0.1623
```

Рисунок 4.6 – Використання функції, що повертає середнє значення векторів для елементів

Для того, щоб перевірити чи добре працює дана модель необхідно використовувати отримані векторні подання UAS у подальших задачах машинного навчання та оцінити якість отриманого результату Втім, перед переходом до побудови моделей для конкретних задач доцільно візуально оцінити якість векторних уявлень fastText. Для цього можна використовувати відомі графіки tSNE, які дозволяють знизити розмірність векторів для їх візуалізації у 2D або 3D просторі. На зображенні (рис. 4.7) наведено тривимірну tSNE-візуалізацію векторів, обчислених для UAS із вибірки потенційних кібератак на фінансові установи за допомогою моделі fastText. Модель змогла згенерувати векторні подання, які відображають важливі характеристики оригінальних UAS. Таким чином, спостерігається чітке розділення точок за видами кібератак.

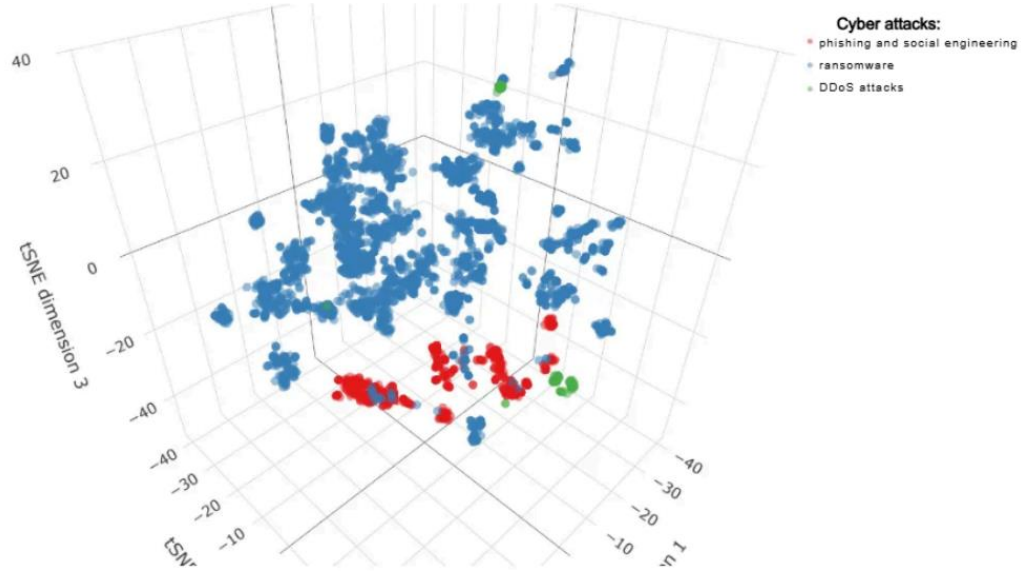


Рисунок 4.7 – 3D візуалізація tSNE векторів, отриманих за допомогою некерованої моделі fastText

Навчання керованої (supervised) моделі fastText вимагає наявності розмічених даних. У цьому контексті парсери UAS можуть бути дуже корисними, адже дозволяють отримати мітки для тисяч прикладів. Алгоритм fastText підтримує як багатокласову, так і мультиетикетну класифікацію. Стандартний формат міток: __label__<значення>. Такі мітки мають передувати кожному документу у навчальному наборі (у випадку багатьох міток – розділені пробілами).

Оскільки ми створюємо векторні подання, які відображають різницю між UAS за типом кібератак на фінансові установи, нижче показано (рис.4.8) розмічені дані, які можна використати для навчання відповідної моделі.

```
__label__server mozilla/5.0 (iphone; cpu iphone os 10_3 like mac os x) applewebkit/602.1.50 (khtml, like gecko) crios/56.0.2924.75
mobile/14e5239e safari/602.1 ruxitsynthetic/1.0 v256748272 t2095300180647406436
__label__mobile mozilla/5.0 (iphone; cpu iphone os 12_4_3 like mac os x) applewebkit/605.1.15 (khtml, like gecko) mobile/15e148 instagram
123.0.0.24.115 (iphone6,2; ios 12_4_3; nl_nl; nl-nl; scale=2.00; 640x1136; 188362626) nw/1
__label__mobile mozilla/5.0 (iphone; cpu iphone os 10_2_1 like mac os x) applewebkit/602.4.6 (khtml, like gecko) mobile/14d27
[fban/fbios;fbav/84.0.0.36.37;fbbv/52496792;fbdv/iphone7,1;fbmd/iphone;fbsn/ios;fbsv/10.2.1;fbss/3;fbcr/3ireland;fbid/phone;fblc/en_gb;fbop/5
;fbrv/0]
__label__mobile mozilla/5.0 (linux; u; android 8.1.0; fr-fr; redmi s2 build/opm1.171019.011) applewebkit/537.36 (khtml, like gecko)
version/4.0 chrome/61.0.3163.128 mobile safari/537.36 xiaomi/miuibrowser/10.8.3-g
__label__mobile mozilla/5.0 (ipad; cpu os 10_2_1 like mac os x) applewebkit/602.4.6 (khtml, like gecko) mobile/14d27
[fban/fbios;fbav/88.0.0.60.70;fbbv/55239788;fbdv/ipad6,4;fbmd/ipad;fbsn/ios;fbsv/10.2.1;fbss/2;fbcr/sprint;fbid/tablet;fblc/en_us;fbop/5;fbrv
/0]
__label__mobile mozilla/5.0 (ipad; cpu os 9_0 like mac os x) applewebkit/601.1.46 (khtml, like gecko) mobile/13a344
[fban/fbios;fbav/8.0.0.28.18;fbbv/1665515;fbdv/ipad2,1;fbmd/ipad;fbsn/iphone os;fbsv/9.0;fbss/1; fbcr/;fbid/tablet;fblc/en_us;fbop/1]
__label__mobile mozilla/5.0 (linux; android 9; oneplus a6000 build/pkq1.180716.001; wv) applewebkit/537.36 (khtml, like gecko) version/4.0
chrome/76.0.3809.111 mobile safari/537.36 [fb_iab/fb4a;fbav/233.0.0.36.117;]
__label__mobile mozilla/5.0 (linux; android 5.1.1; build/lmy49i; wv) applewebkit/537.36 (khtml, like gecko) version/4.0 chrome/59.0.3071.125
safari/537.36
__label__mobile mozilla/5.0 (linux; android 7.0; sm-g925f build/nrd90m; wv) applewebkit/537.36 (khtml, like gecko) version/4.0
chrome/74.0.3729.136 mobile safari/537.36 jssdk/2 topbuzz/11.2.3 nettype/4g
__label__computer mozilla/5.0 (macintosh; intel mac os x 10.11; rv:46.0) gecko/20100101 firefox/46.0 | hardenize/v1.1168.1
(https://www.hardenize.com)
```

Рисунок 4.8 – Приклад розмічених даних, що підходять для навчання контрольованої моделі fastText

Команда R для навчання контрольованої моделі fastText на таких ромічених даних схожа на попередню (рис. 4.9).

```
m_sup <- ft$train_supervised(
    input = "./data/train_data_sup.txt",
    lr = 0.05,
    dim = 32L, # vector dimension
    ws = 3L,
    minCount = 1L,
    minCountLabel = 10L, # min label occurence
    minn = 2L,
    maxn = 6L,
    neg = 3L,
    wordNgrams = 2L,
    loss = "softmax", # loss function
    epoch = 100L,
    thread = 10L
)
```

Рисунок 4.9 – Навчання керованої моделі fastText

Оцінити якість отриманої керованої моделі можна за допомогою розрахунку таких метрик, як точність (precision), повнота (recall) та F1-міра на вибірці. Ці показники можуть бути обчислені як у середньому по всіх мітках, так і для окремих класів. Нижче наведено метрики якості для UAS із міткою «ransomware» (рис. 4.10).

```
 metrics <- m_sup$test_label("./data/test_data_sup.txt")
metrics["__label__mobile"]
## $`__label__ransomware`
## $`__label__ransomware`$precision
## [1] 0.998351
##
## $`__label__ransomware`$recall
## [1] 0.9981159
##
## $`__label__ransomware`$f1score
## [1] 0.9982334
```

Рисунок 4.10 – Метрики якості для UAS із міткою «ransomware»

Візуальна оцінка tSNE-проєкції векторів для даної контрольованої моделі підтверджує її високу якість: спостерігається чітке розділення навчальних прикладів за типом потенційних кібератак на фінансові установи (рис. 4.11). При навчанні контрольованої моделі алгоритм отримує додаткову інформацію, що дозволяє йому формувати вектори, орієнтовані на конкретне завдання.

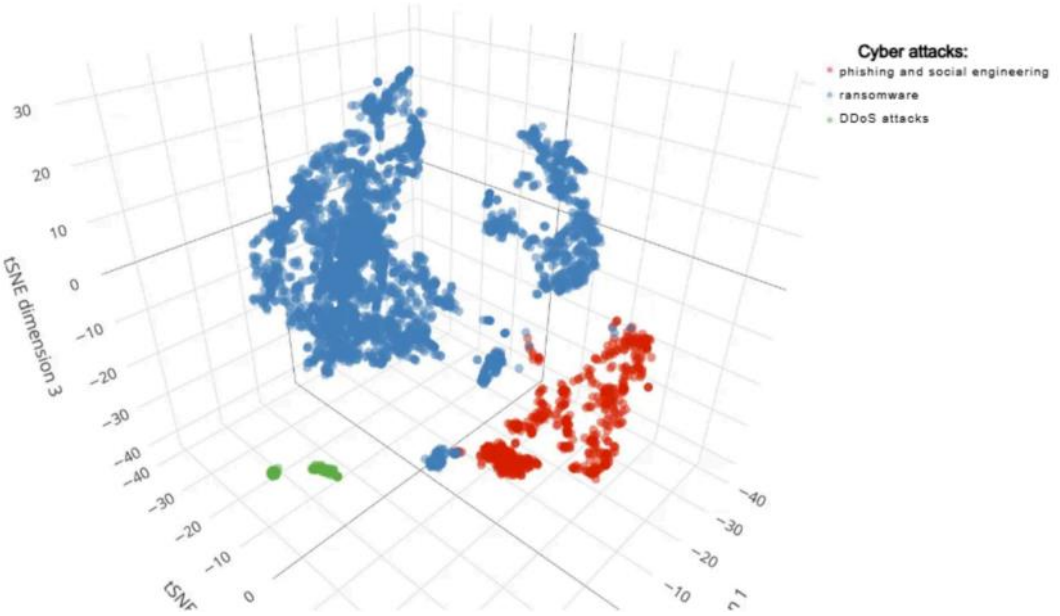


Рисунок 4.11 – 3D візуалізація tSNE векторів, отриманих за допомогою контрольованої моделі fastText з мітками, що відповідають типам кібератак

Отже, інформація, яка міститься у рядках UAS, може бути ефективно представлена у вигляді векторних подань низької розмірності. Моделі для створення таких векторів можна навчати як у неконтрольованому, так і в контрольованому режимах. Неконтрольовані вектори є універсальними та можуть бути використані у будь-яких подальших задачах машинного навчання. Таким чином, можна дійти висновку, що найефективніше навчати моделі на конкретних завданнях. Одним із найбільш ефективних алгоритмів для векторизації UAS є fastText, реалізації якого доступні всіма основними мовами програмування, що використовуються у машинному навчанні.

4.2 Розроблення структури бази даних

Розробка програмних проєктів має свої етапи, зокрема початковими є обрання основного функціоналу та визначення функціональних вимог до розробленого вебсервісу. Наступною частиною є формування структури БД.

PostgreSQL – це система управління реляційними базами даних з відкритим кодом (ORDBMS), яка відома своєю надійністю, масштабованістю та потужною підтримкою розширених функцій SQL.

У контексті управління ідентифікацією та доступом (IAM) PostgreSQL відіграє ключову роль як бекенд-система для зберігання ідентифікацій користувачів, дозволів та даних аутентифікації. Системи IAM відповідають за управління користувачами та їх доступом до різних ресурсів в IT-інфраструктурі організації. Здатність PostgreSQL ефективно обробляти великі та складні набори даних робить її ідеальним вибором для зберігання даних користувачів, ролей і політик доступу. Багато рішень IAM, таких як Keycloak або OpenAM, використовують реляційні бази даних на кшталт PostgreSQL для збереження даних аутентифікації, таких як імена користувачів, паролі (захешовані), метадані користувачів та інформація про сесії. Крім того, підтримка PostgreSQL розширених функцій безпеки, таких як рівнева безпека рядків і шифрування, забезпечує захист чутливих даних, таких як облікові дані користувачів. У системах IAM PostgreSQL також може зберігати детальні журнали та сліди аудиту, які відстежують активність користувачів, що є важливим для дотримання вимог безпеки та регулювання. Підтримка складних об’єднань і зв’язків у PostgreSQL дозволяє IAM-системам зберігати складні мапування користувачів та ролей, а також ієрархії дозволів, що необхідні для реалізації політик доступу за принципом найменших привілеїв. Іншою важливою функцією PostgreSQL є її здатність обробляти одночасні транзакції, що важливо для IAM-систем, які повинні одночасно аутентифікувати багатьох користувачів. PostgreSQL також можна інтегрувати з зовнішніми протоколами аутентифікації, такими як LDAP або OAuth, і зберігати необхідну інформацію для підтримки цих протоколів (рис. 4.12). Нарешті, її надійність і продуктивність роблять PostgreSQL відмінним вибором для забезпечення високої доступності та швидкої роботи систем IAM навіть під високим навантаженням.

Гнучкість і продуктивність PostgreSQL роблять її основою багатьох рішень IAM, де вона сприяє управлінню користувачами, ролями та дозволами в різноманітних додатках. Наприклад, коли користувач намагається отримати доступ до системи, PostgreSQL зберігає облікові дані користувача і виконує

аутентифікацію, порівнюючи введені дані з тими, що зберігаються. Це включає як аутентифікацію на основі пароля (де зазвичай паролі захешовані і зберігаються), так і більш складні методи, такі як багатофакторна аутентифікація (MFA), коли додаткові атрибути користувача або зовнішні токени перевіряються. PostgreSQL також відіграє важливу роль в управлінні доступом на основі ролей (RBAC), де користувачі призначаються певним ролям, кожна з яких має свій набір дозволів.

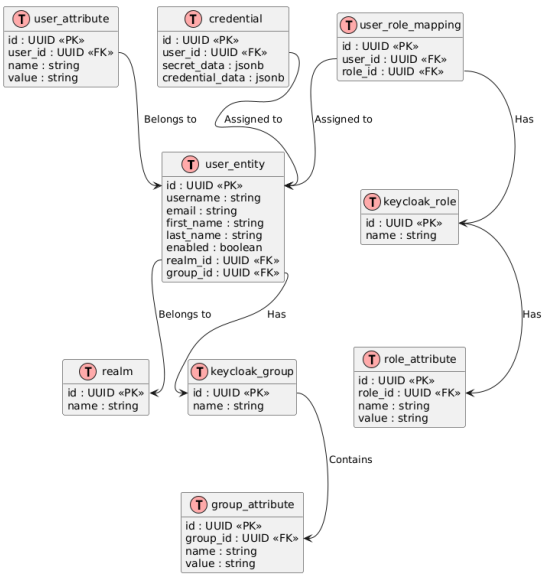


Рисунок 4.12 – Діаграма відношень між сутностями в БД

Завдяки PostgreSQL IAM-системи можуть ефективно запитувати ролі користувачів і пов’язані з ними привілеї, що забезпечує надання або відмову в доступі відповідно до визначених політик. Крім того, IAM-системи часто використовують PostgreSQL для управління сесіями користувачів, гарантуючи, що користувачі аутентифіковані та авторизовані протягом часу їхнього доступу.

Здатність PostgreSQL інтегруватися з іншими постачальниками ідентифікації та федеративними системами ідентифікації робить її важливим компонентом сучасної інфраструктури IAM. Крім того, рішення IAM часто потребують масштабування, і можливості реплікації та кластеризації PostgreSQL дозволяють забезпечити високу доступність і горизонтальне масштабування, що дозволяє IAM-сервісу обробляти зростаючу кількість користувачів і запитів. Журнали аудиту та транзакційні можливості PostgreSQL також підтримують вимоги IAM щодо

детального відстеження подій та моніторингу, що є важливим для безпеки та дотримання нормативних вимог.

Таблиця user_entity зберігає основну інформацію про користувачів у системі. Вона містить унікальний ідентифікатор користувача (UUID), що є первинним ключем. Кожен користувач має унікальне ім’я користувача, адресу електронної пошти, ім’я та прізвище. Поле enabled вказує, чи активований обліковий запис користувача. Користувач прив'язаний до певного домену (realm), що реалізується через зовнішній ключ realm_id. Також користувач може належати до певної групи, що задається полем group_id. Ця таблиця слугує центральним вузлом для зв’язків з іншими таблицями, такими як атрибути, ролі та облікові дані.

Таблиця user_attribute дозволяє зберігати додаткові атрибути для користувачів. Вона містить унікальний ідентифікатор, а також зовнішній ключ user_id, який посилається на user_entity. Кожен запис вказує назву атрибута (name) та його значення (value). Це дозволяє зберігати гнучкі метадані про користувача, що не входять до основного профілю. Наприклад, це можуть бути уподобання, налаштування або бізнес-атрибути. Зв’язок з таблицею користувачів забезпечує однозначне визначення приналежності атрибутів. Така модель дозволяє розширювати профіль користувача без зміни основної схеми.

Таблиця realm зберігає інформацію про домени (реальми), які визначають ізольовані контексти автентифікації. Кожен запис має унікальний ідентифікатор та назву домену. Реальм використовується для логічного поділу користувачів, конфігурацій і політик доступу. Користувачі з таблиці user_entity мають зовнішній ключ realm_id, що вказує на належність до певного реальму. Це дозволяє управляти багатодоменними середовищами з відокремленими політиками безпеки. Реальми особливо корисні для SaaS-сервісів або великих організацій із сегментованими підрозділами. Таблиця є фундаментальною для побудови багатодоменними середовища.

Таблиця credential відповідає за зберігання облікових даних користувачів. Вона має унікальний ідентифікатор та зовнішній ключ user_id, який вказує на

відповідного користувача. Поля secret_data та credential_data використовують тип jsonb для зберігання зашифрованої інформації та метаданих. Це дозволяє гнучко обробляти різні типи автентифікаційних механізмів, включаючи паролі, ключі або токени. Таблиця підтримує безпечне зберігання облікових даних із можливістю розширення. Зв'язок із таблицею користувачів забезпечує індивідуальне призначення облікових даних. Такий підхід дозволяє управляти безпекою на granularному рівні.

Таблиця keycloak_group зберігає групи, до яких можуть належати користувачі. Вона містить унікальний ідентифікатор і назву групи. Групи дозволяють логічно об'єднувати користувачів для спрощення управління правами доступу. Кожен користувач у таблиці user_entity може бути асоційований із групою через поле group_id. Таблиця є корисною для реалізації ролей, доступів або організаційної структури. Через таблицю group_attribute можна додати метаінформацію до кожної групи. Це забезпечує розширюваність і динамічність групових налаштувань.

Таблиця group_attribute дозволяє додавати атрибути до груп користувачів. Вона має унікальний ідентифікатор і зовнішній ключ group_id, що пов'язує запис із відповідною групою. Кожен атрибут має назву (name) та значення (value), які описують додаткові характеристики групи. Це можуть бути політики, мітки або службова інформація. Таблиця підтримує динамічне розширення функціоналу без зміни структури основної таблиці груп. Вона корисна для гнучкої адаптації групових параметрів під бізнес-вимоги. Таким чином, атрибути можуть відображати ієрархії, області застосування або налаштування доступу.

Таблиця keycloak_role зберігає ролі, що використовуються для контролю доступу в системі. Вона містить унікальний ідентифікатор та назву ролі. Ролі визначають рівень доступу або функціональність, яку має користувач. Вони є ключовим елементом у політиках авторизації. Через таблицю user_role_mapping ролі призначаються конкретним користувачам. Додаткові атрибути до ролей

можуть зберігатися у таблиці role_attribute. Така структура дозволяє створювати гнучкі, деталізовані моделі доступу.

Таблиця role_attribute додає можливість зберігати додаткові атрибути для кожної ролі. Вона містить унікальний ідентифікатор і зовнішній ключ role_id, що зв'язує запис із відповідною роллю. Атрибути мають назву (name) та значення (value), які розширюють опис ролі. Це можуть бути рівні доступу, описові теги або інші політики. Таблиця дозволяє деталізувати параметри ролей без порушення основної схеми. Це зручно для побудови гнучких та конфігурованих політик авторизації. Вона також забезпечує масштабованість моделі керування правами.

Таблиця user_role_mapping зберігає відповідність між користувачами та ролями. Вона має унікальний ідентифікатор і два зовнішні ключі: user_id та role_id. Це дозволяє прив'язати користувача до однієї або кількох ролей у системі. Таблиця реалізує many-to-many зв'язок між користувачами та ролями. Це критично для реалізації політик RBAC (Role-Based Access Control). Такий підхід забезпечує гнучке керування дозволами на основі ролей. Зв'язок з таблицями user_entity та keycloak_role робить цю таблицю ключовим елементом у схемі авторизації.

4.3 Розроблення серверної частини

Мікросервісна архітектура передбачає розподілену структуру, де кожен сервіс виконує окрему бізнес-функцію та взаємодіє з іншими через мережу. У такій системі питання безпеки передачі даних стає критично важливим, адже мікросервіси часто обмінюються конфіденційною інформацією, яка може бути перехоплена чи змінена зловмисниками. Для захисту цього обміну необхідно впроваджувати мікросервіси, відповідальні за шифрування протоколів передачі даних, які гарантують цілісність і конфіденційність інформації. Без належного шифрування існує ризик несанкціонованого доступу до даних, особливо в умовах масштабованих і динамічних середовищ, де кількість точок взаємодії постійно зростає.

Використання різних протоколів шифрування в мікросервісній архітектурі дозволяє адаптувати рівень захисту до специфіки кожного каналу зв’язку. Наприклад, для REST API стандартом є застосування HTTPS із SSL/TLS-сертифікатами, які забезпечують шифрування трафіку та автентифікацію сторін. Для інших протоколів, таких як gRPC чи AMQP, також існують власні механізми шифрування, що дозволяють захистити передані дані незалежно від технології. Впровадження спеціалізованих мікросервісів для шифрування дає змогу централізовано керувати політиками безпеки, оновлювати криптографічні алгоритми та швидко реагувати на нові загрози.

Ключовою перевагою виділення окремих мікросервісів для шифрування є можливість стандартизувати підходи до захисту даних у всій системі. Це дозволяє уникнути ситуацій, коли різні команди розробників впроваджують власні, не завжди сумісні або достатньо надійні рішення. Такі мікросервіси можуть підтримувати сучасні протоколи TLS 1.2/1.3 із рекомендованими наборами шифрів, наприклад TLS_AES_256_GCM_SHA384 чи TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, що відповідає сучасним вимогам безпеки. Централізація шифрування також спрощує аудит, моніторинг та оновлення сертифікатів, підвищуючи загальний рівень захисту мікросервісної екосистеми.

Окрім шифрування, такі мікросервіси можуть інтегрувати додаткові механізми безпеки: автентифікацію, авторизацію та журналювання подій. Наприклад, використання JWT для автентифікації між сервісами чи впровадження API Gateway, який контролює всі зовнішні запити, дозволяє забезпечити багаторівневий захист. Безперервний моніторинг та аудит дій допомагають вчасно виявляти і усувати потенційні вразливості. Завдяки цьому мікросервісна архітектура з окремими сервісами для шифрування стає більш стійкою до сучасних кіберзагроз, забезпечуючи надійний захист даних незалежно від використовуваних протоколів і технологій (рис. 4.13).

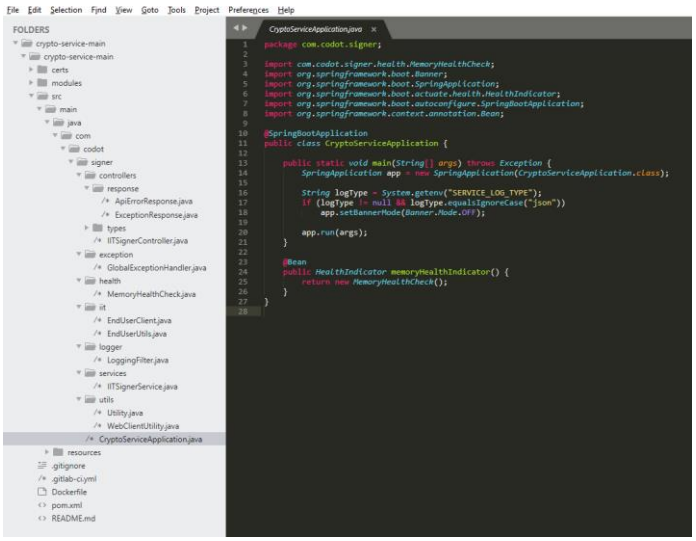


Рисунок 4.13 – Структура проєкту мікросервісу для організації шифрування для обміну даними

Код класу CryptoServiceApplication є типовим для програми на Spring Boot. Розберемо його по частинах. Спочатку йде оголошення пакета package com.codot.signer, до якого належить цей клас. Пакети використовуються для організації класів і уникнення конфліктів імен. Потім імпортуються інші класи, які використовуються в цьому файлі. MemoryHealthCheck це користувацький клас для перевірки стану пам’яті. Клас org.springframework.boot.Banner необхідний для керування банером, який відображається при запуску Spring Boot. Клас org.springframework.boot.SpringApplication використовується для запуску Spring Boot застосунків. Інтерфейс Spring Boot Actuator org.springframework.boot.actuate.health.HealthIndicator слугує для індикації стану здоров’я програми. Бачимо, що org.springframework.boot.autoconfigure.SpringBootApplication це зручна анотація, яка комбінує @Configuration, @EnableAutoConfiguration та @ComponentScan. Анотація org.springframework.context.annotation.Bean, яка вказує, що метод повертає «бін» (об’єкт), який має бути керований Spring IoC-контейнером. .

Назва CryptoServiceApplication вказує на те, що це, ймовірно, застосунок, пов’язаний з криптографічними сервісами. Рядок public static void main(String[] args) throws Exception { … } це стандартна точка входу для будь-якої Java-програми. Коли ви запускаєте цю програму, виконання починається з цього методу.

У рядку SpringApplication app = new SpringApplication(CryptoServiceApplication.class);: Створюється екземпляр SpringApplication, який відповідає за bootstrapping (початкове завантаження) і запуск Spring Boot додатка. Клас CryptoServiceApplication.class передається для вказівки головного класу додатка. У рядку String logType = System.getenv(«SERVICE_LOG_TYPE»); отримуємо значення змінної середовища з назвою «SERVICE_LOG_TYPE». Якщо змінна середовища «SERVICE_LOG_TYPE» існує і її значення (без урахування регістру) є «json», то банер Spring Boot (графічне зображення або текст, що відображається при запуску) буде вимкнено. Це може бути корисним для середовищ, де логи збираються в JSON-форматі, і візуальний банер може заважати. Метод app.run(args); запускає Spring Boot додаток. Він виконує такі завдання, як завантаження контексту Spring, конфігурація вбудованого вебсервера (якщо це вебзастосунок) та інші початкові налаштування.

Анотація @Bean вказує Spring, що метод memoryHealthIndicator() створює і повертає об’єкт, який має бути зареєстрований як «бін» у контексті Spring. Метод public HealthIndicator memoryHealthIndicator() повертає об’єкт типу HealthIndicator. Рядок return new MemoryHealthCheck(); створює новий екземпляр класу MemoryHealthCheck і повертає його. Оскільки MemoryHealthCheck реалізує інтерфейс HealthIndicator (або є його підкласом), Spring Actuator буде використовувати цей бін для перевірки стану пам’яті застосунку.

Цей код є початковою точкою для Spring Boot сервісу. Він налаштовує базові функціональні можливості Spring Boot, включаючи можливість вимкнення банера на основі змінної середовища та інтеграцію перевірки стану здоров’я пам’яті за допомогою Spring Boot Actuator. Додаток надає криптографічні сервіси представлені на рис. 4.14 програмний код реалізує утиліту для здійснення HTTP-запитів за допомогою Spring WebClient. Клас WebClientUtility знаходиться в пакеті com.codot.signer.utils і призначений для спрощення взаємодії з зовнішніми сервісами через HTTP. Він використовує реактивний WebClient від Spring Framework, що робить його ефективним для неблокуючих операцій вводу/виводу.

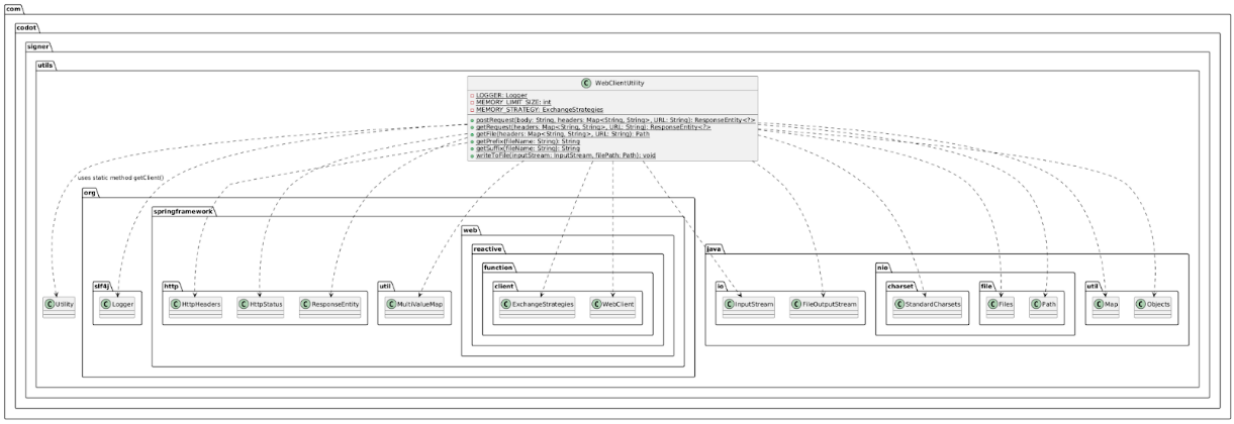


Рисунок 4.14 – Діаграма класів файлу WebUtilityClient.java

Метод getClient() надає попередньо налаштований HTTP-клієнт або його конектор. Аналогічно static *.*.http.MediaType.APPLICATION_JSON;, дозволяє використовувати APPLICATION_JSON без вказівки MediaType. Представлено публічний клас WebClientUtility. Та статичний та фінальний об’єкт Logger для логування подій та помилок в цьому класі. Використовується SLF4J (Simple Logging Facade for Java). Статична та фінальна константа int MEMORY_LIMIT_SIZE = 256 * 1024 * 1024; визначає максимальний розмір буфера пам’яті для обробки тіл відповідей у WebClient. Тут встановлено 256 мегабайт. Присутня статична та фінальна стратегія ExchangeStrategies MEMORY_STRATEGY = ExchangeStrategies.builder() … .build(); для обміну даними для WebClient. Вона налаштована на використання MEMORY_LIMIT_SIZE як максимального розміру в пам’яті для кодеків, що дозволяє обробляти великі відповіді без помилок «DataBufferLimitException».

У класі WebUtilityClient також визначені статичні методи для виконання HTTP-запитів (POST, GET) та отримання файлів. Після цього exchangeStrategies(MEMORY_STRATEGY) застосовує визначену раніше стратегію для контролю розміру пам’яті, clientConnector(getClient()) використовує клієнт-конектор, отриманий від методу Utility.getClient(). Це може бути, наприклад, конектор для Netty або Apache HttpClient. У процесі роботи виконується POST-запит з параметрами (.uri(URL): встановлює цільовий URL, .contentType(APPLICATION_JSON): встановлює тип вмісту запиту як JSON,

.bodyValue(body): додає тіло запиту, .exchangeToMono(…): обробляє відповідь). Запит виконується на основі реактивний підходу. Таким чином clientResponse.statusCode().value() отримує числовий код стану відповіді, а clientResponse.headers().asHttpHeaders() отримує заголовки відповіді, тоді як clientResponse.bodyToMono(String.class) конвертує тіло відповіді в Mono<String>. При цьому .doOnError(…) обробляє помилки, що виникають під час виконання запиту, логуючи їх. Метод .block() блокує потік виконання, поки не буде отримана відповідь. У реактивному програмуванні зазвичай уникають block(), але в утилітних методах це може бути виправдано для простоти.

Призначення WebClientUtility полягає у тому, що клас є централізованою утилітою для виконання HTTP-запитів у додатку. Він інкапсулює логіку налаштування WebClient, обробку відповідей (включаючи помилки), а також спеціалізовану логіку для завантаження файлів. Використання статичних методів робить його легкодоступним з будь-якої частини програми без необхідності створення екземпляра класу WebClientUtility. Це типовий приклад «сервісного» або «утилітного» класу у Spring Boot застосунках. Хоча WebClient є реактивним, використання .block() в цих статичних методах перетворює їх на блокуючі виклики. Це може бути прийнятно для деяких сценаріїв, де проста синхронна поведінка бажана, але в суто реактивних додатках це вважається антипатерном. Помилки логуються, але не завжди повторно кидаються (крім getFile у випадку відсутності Content-Disposition). Залежно від вимог, можливо, потрібно додати більш детальну обробку помилок або перетворити їх на кастомні винятки. Кожен виклик postRequest, getRequest або getFile створює новий екземпляр WebClient. Хоча WebClient є потоково-безпечним, іноді більш ефективно використовувати єдиний екземпляр WebClient або кешувати його для повторного використання, особливо якщо багато запитів виконуються. Однак, у даному випадку, завдяки тому, що WebClient.builder() використовує вже налаштований ExchangeStrategies та ClientConnector (з Utility.getClient()), це зменшує накладні витрати на кожне створення. Розглянемо код класу Utility (рис. 4. 15). Цей клас, як випливає з його

назви та вмісту, є набором допоміжних (утилітних) методів для різних функцій, таких як обробка HTTP-заголовків, налаштування HTTP-клієнтів та робота з датами.

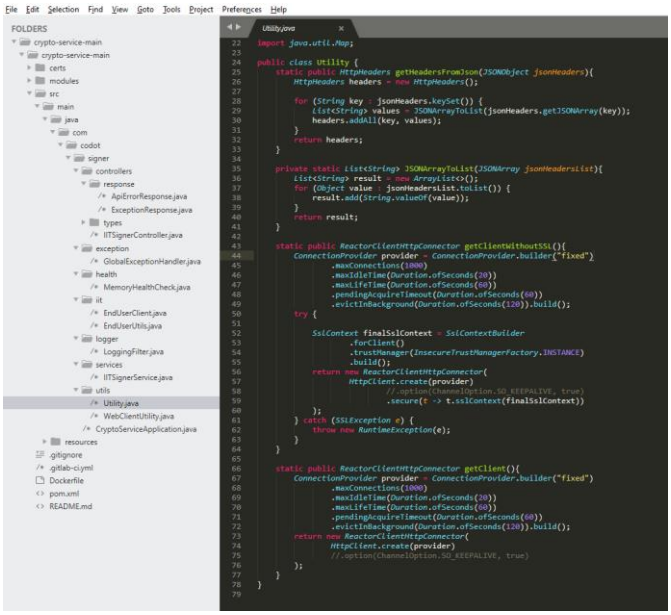


Рисунок 4.15 – Файл Utility.java

Пакет package com.codot.signer.utils; вказує, що цей клас належить до пакету com.codot.signer.utils.

Методи повертають ReactorClientHttpConnector, який є реалізацією ClientHttpConnector для WebClient, що використовує Reactor Netty як базовий HTTP-engine. Обидва методи налаштовують ConnectionProvider (провайдер з’єднань), який керує пулом HTTP-з’єднань.

Метод ConnectionProvider.builder(«fixed») створює провайдер з’єднань з фіксованим розміром пулу. Метод .maxConnections(1000): задає максимальну кількість активних з’єднань у пулі. Метод .maxIdleTime(Duration.ofSeconds(20)) встанолює максимальний час, протягом якого з’єднання може залишатися неактивним у пулі, перш ніж буде закрито.

Метод .maxLifeTime(Duration.ofSeconds(60)) встановлює аксимальний час життя з’єднання у пулі, після якого воно буде закрито незалежно від активності. Метод pendingAcquireTimeout(Duration.ofSeconds(60)) задає максимальний час очікування для отримання з’єднання з пулу, якщо всі з’єднання зайняті.

Метод .evictInBackground(Duration.ofSeconds(120)) встановлює фонове видалення неактивних або прострочених з'єднань кожні 120 секунд. Метод static public ReactorClientHttpConnector getClientWithoutSSL() повертає ReactorClientHttpConnector, який налаштований на ігнорування перевірки SSL-сертифікатів. Використання InsecureTrustManagerFactory.INSTANCE є небезпечним у виробничому середовищі, оскільки це відключає перевірку сертифікатів SSL/TLS. Це означає, що клієнт довірятиме будь-якому сертифікату, навіть самопідписаним або недійсним, що робить його вразливим до атак типу «людина посередині» (man-in-the-middle).

Цей підхід може бути виправданий лише для локального розроблення або тестування, де безпека не є пріоритетом. Метод static public ReactorClientHttpConnector getClient() метод повертає ReactorClientHttpConnector зі стандартною конфігурацією пулу з'єднань, але без спеціальної конфігурації SSL. Це означає, що за замовчуванням він буде виконувати стандартну перевірку сертифікатів, довіряючи сертифікатам з JVM TrustStore (якщо не налаштовано інше).

Розглянемо наданий код класу IITSignerService. Клас розташований у пакеті com.codot.signer.services і, судячи з його назви та вмісту, є сервісом, що надає функціональність для взаємодії з криптографічною бібліотекою, ймовірно, для роботи з електронним цифровим підписом (ЕЦП) або схожими криптографічними операціями (рис. 4. 16).

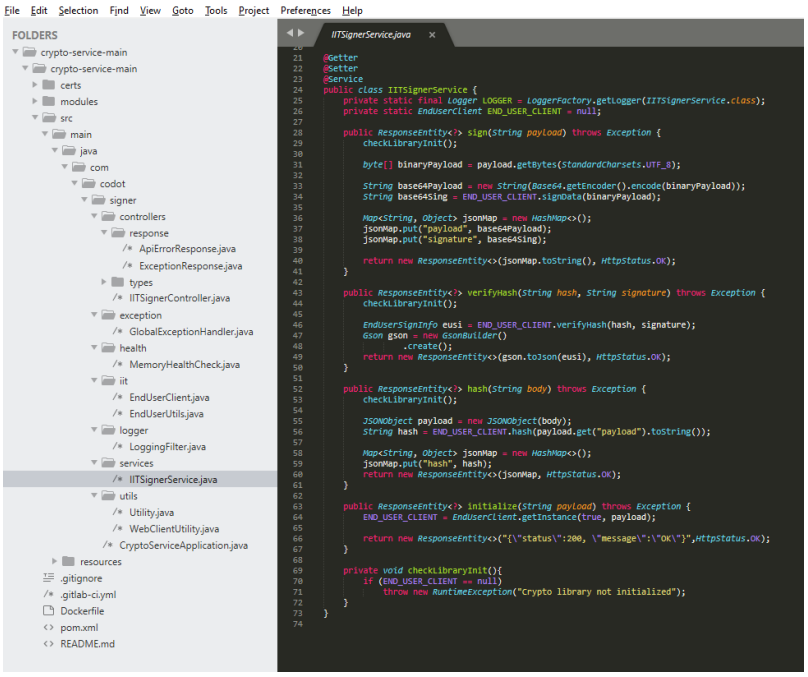


Рисунок 4.16 – IITSignerService.java

Назва «IIT» вказує на конкретну українську компанію «Інститут Інформаційних Технологій» (IIT), яка займається розробкою криптографічних засобів. Оголошення пакету package com.codot.signer.services; вказує пакет класу. При цьому пакет містить класи для роботи з JSON (Gson, JSONObject), логуванням (slf4j), Spring (ResponseEntity, HttpStatus, @Service), а також спеціалізовані класи для криптографії (EndUserClient, EndUserSignInfo) та обробки даних (Base64, StandardCharsets). Lombok використовується для скорочення бойлерплейт-коду. Анотації @Getter та @Setter (Lombok) від Lombok автоматично генерують стандартні методи геттерів і сеттерів для всіх нестатичних полів класу під час компіляції. У даному випадку, для полів класу IITSignerService немає нестатичних полів, тому ці анотації можуть бути зайвими або призначені для полів, які могли бути раніше або з’являться в майбутньому. Анотація @Service (Spring) позначає клас як «сервісний компонент» у Spring-додатку. Це дозволяє Spring-контейнеру автоматично виявляти, створювати та керувати життєвим циклом цього класу (IOC – Inversion of Control), а також використовувати його для впровадження залежностей (DI – Dependency Injection). Поле класу private static final Logger LOGGER це статичний фінальний об’єкт Logger для логування повідомлень та

помилок, пов’язаних з цим сервісом. Поле класу private static EndUserClient END_USER_CLIENT = null; це статичне поле, яке зберігатиме єдиний екземпляр EndUserClient. Оскільки це статичне поле, воно буде спільним для всіх екземплярів IITSignerService (або єдиного екземпляра, якщо Spring використовує синглтон, що є типовим для @Service). Ініціалізується null і буде встановлено методом initialize.

Клас IITSignerService є обгорткою (або фасадом) над зовнішньою криптографічною бібліотекою (представленою EndUserClient). Він надає зручний HTTP-орієнтований інтерфейс для виконання таких криптографічних операцій, як підписання, верифікація підпису/хешу та хешування даних. Він також керує життєвим циклом EndUserClient, гарантуючи, що його ініціалізація відбувається коректно перед використанням. Клієнт відправляє запит на ендпоінт «ініціалізувати» (наприклад, /iit/initialize) з необхідними параметрами. Після успішної ініціалізації клієнт може відправляти запити на «підписати» (/iit/sign), «хешувати» (/iit/hash) або «перевірити хеш» (/iit/verifyHash). Синглтон EndUserClient: Використання статичного поля END_USER_CLIENT і методу getInstance вказує на шаблон «синглтон» для EndUserClient. Це поширений підхід для криптографічних бібліотек, які можуть мати складну ініціалізацію або керувати ресурсами (наприклад, апаратними токенами). Управління станом: оскільки END_USER_CLIENT є статичним, стан ініціалізації є глобальним для всього додатку. Якщо додаток працює в багатопотоковому середовищі, це вимагає, щоб EndUserClient був потоково-безпечним.

Клас EndUserClient у пакеті com.codot.signer.iit є ключовим компонентом для взаємодії з низькорівневою криптографічною бібліотекою від компанії ІІТ (, яка відома своїми криптографічними продуктами. Цей клас (рис. 4.17) діє як синглтон, гарантуючи, що в будь-який момент часу активний лише один екземпляр клієнта (і, отже, базової криптографічної бібліотеки). Управління синглтоном забезпечує єдиний екземпляр EndUserClient. Під час ініціалізації криптографічної бібліотеки конфігурується EndUser (основний клієнт бібліотеки ІІТ) з різними налаштуваннями (режим інтерфейсу користувача, мова, кодування, параметри

виконання, тип підпису CAdES, налаштування проксі, OCSP, TSP, LDAP, CMP). У процесі управління сертифікатами та ключами зчитуються сертифікати ЦСК, сертифікати користувачів та приватні ключі з файлів або інших носіїв ключів. В процесі верифікації перевіряються цифрові підписи та хеші. Під час хешування обчислюються криптографічні хеші даних.

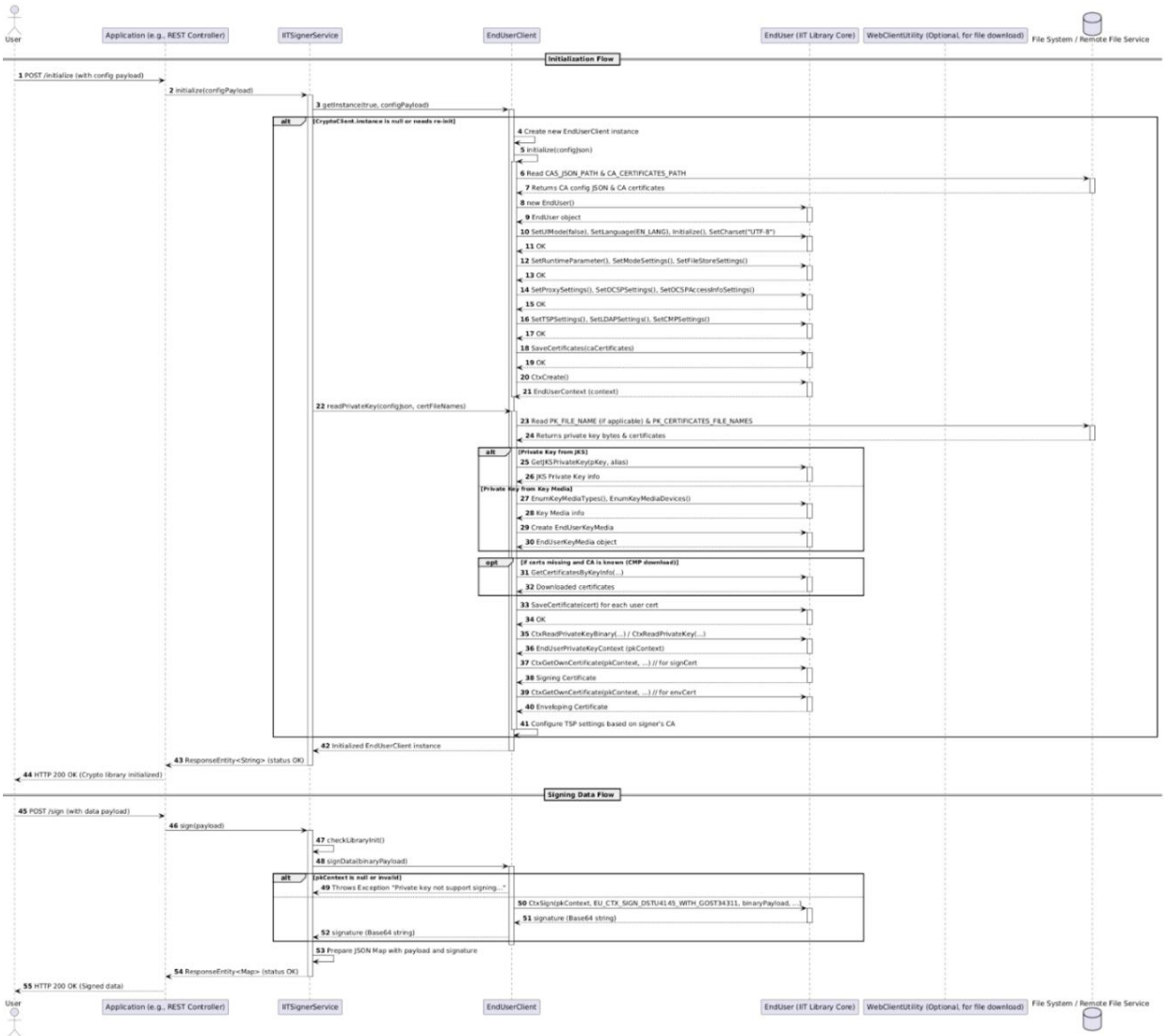


Рисунок 4.17 – Діаграма послідовності для бізнес-логіки EndUserClient.java

Пакет com.iit.certificateAuthority.endUser.libraries.signJava.* – це основні класи з Java-бібліотеки криптографії ІІТ, що надають низькорівневі функції для підписання, верифікації та керування ключами. Пакет com.codot.signer.utils.WebClientUtility вказує на зовнішню залежність від службового класу в тому ж проєкті для виконання вебзапитів (наприклад, для

завантаження файлів конфігурації). Пакет JSONObject, JSONArray широко використовуються для розбору конфігурації, наданої у вигляді JSON-рядків. Поля класу EndUserClient: LOGGER для логування, FILE_SERVICE_URL якщо програмі потрібно отримувати файли конфігурації з віддаленого сервісу, використовується цей URL. Статичний, volatile синглтон-екземпляр instance, в якому volatile має вирішальне значення для правильної реалізації синглтона в багатопотоковому середовищі, endUser це головний об’єкт з бібліотеки com.iit, через який виконуються всі криптографічні операції, CAs – масив для зберігання конфігурацій центрів сертифікації, розібраних з JSON-файлу. Об’єкти контексту context та pkContext, керовані бібліотекою ІІТ для конкретних криптографічних операцій та обробки приватних ключів. Сертифікати підписання та шифрування користувача (signCert та envCert) зберігають інформацію для швидкого отримання іншими частинами програми. Метод синглтона getInstance це стандартний шаблон подвійної перевірки блокування для потокобезпечної ініціалізації синглтона. Він приймає envAsString (JSON-рядок), який містить усі параметри конфігурації для криптографічної бібліотеки, включаючи шляхи до сертифікатів, налаштування проксі та деталі приватного ключа. Викликає initialize для конфігурації основного об'єкта EndUser. Викликає readPrivateKey для завантаження приватного ключа та пов'язаних сертифікатів. Якщо під час ініціалізації виникає будь-яка помилка, він встановлює localInstance назад на null і повторно викидає виняток. Внутрішній статичний клас CA моделює конфігурацію центру сертифікації (ЦСК). Він містить поля для різних кінцевих точок сервісів ЦСК (OCSP, CMP, TSP) та загальних імен емітентів (issuerCNs). Конструктор приймає JSONObject і розбирає його поля для заповнення об'єкта CA. jsonArrayToStringArray – це допоміжний метод для перетворення JSONArray рядків у String[]. Метод initialize відповідає за налаштування об'єкта endUser (основного клієнта криптографічної бібліотеки) за допомогою безлічі параметрів конфігурації, отриманих з вхідного об'єкта json. Він зчитує шляхи до сертифікатів ЦСК та файлів конфігурації ЦСК (ймовірно, CAs.json та CACertificates.p7b). Закоментовані виклики uploadFile вказують на потенційний

механізм отримання цих файлів з віддаленого сервісу. Він розбирає CAs.json для заповнення масиву CAs. Він встановлює різні параметри для об'єкта endUser: Метод SetUIMode(false) вимикає будь-які графічні компоненти бібліотеки, що важливо для серверних програм. Метод SetLanguage(EndUser.EU_EN_LANG) встановлює мову на англійську. Метод Initialize() виконує ключовий виклик ініціалізації для бібліотеки IIT. Метод SetCharset("UTF-8") встановлює кодування символів. Метод SetRuntimeParameter(...) конфігурує поведінку під час виконання, включаючи EU_SIGN_TYPE_CADES_X_LONG для підписів CAdES-X-Long. Метод SetModeSettings(...) використовується для керування онлайн/офлайн режимами. Методи setFileStoreSettings(...) та SetPath("") налаштовують віртуальне файлове сховище в пам’яті, що означає, що бібліотека не намагатиметься читати/записувати файли на диск для сертифікатів/CBC тощо, а керуватиме ними в пам’яті. Метод SetProxySettings() конфігурує деталі проксі-сервера. Методи SetOCSPSettings(), SetOCSPAccessInfoModeSettings() та SetOCSPAccessInfoSettings() конфігурують OCSP (Online Certificate Status Protocol) сервери для перевірки статусу сертифікатів. Метод SetTSPSettings(): Конфігурує TSP (Timestamp Protocol) сервери для додавання міток часу до підписів. SetLDAPSettings(), SetCMPSettings() конфігурує налаштування LDAP та CMP (Certificate Management Protocol). Метод endUser.SaveCertificates(caCertificates) зберігає кореневі сертифікати ЦСК у внутрішньому контексті бібліотеки. Метод context = endUser.CtxCreate() створює криптографічний контекст, який, як правило, є сесійним об’єктом для виконання операцій. Метод uploadFile є private і наразі закоментований у initialize, але він показує, як клієнт міг би завантажувати файли конфігурації з віддаленого FILE_SERVICE_URL за допомогою WebClientUtility. Це гарний дизайн для гнучкого розгортання. Метод getCA є допоміжним метод для пошуку об'єкта CA за його caSubjectCN. Метод getKeyMedia перераховує доступні типи та пристрої носіїв ключів за допомогою бібліотеки EndUser, щоб створити об'єкт EndUserKeyMedia, який представляє фізичне або логічне розташування приватного ключа. Метод readPrivateKey – це один з найскладніших методів, який відповідає

за завантаження приватного ключа та розбирає численні параметри конфігурації з об’єкта json, щоб знайти та отримати доступ до приватного ключа. Він обробляє два основні сценарії завантаження приватного ключа:

– з файлу, якщо надано PK_FILE_NAME, він зчитує байти приватного ключа. Якщо присутній PK_JKS_ALIAS, це означає контейнер JKS (Java Key Store) і використовує endUser.GetJKSPrivateKey для вилучення ключа та сертифікатів;

– з носія ключа, якщо файл не вказано, він використовує getKeyMedia для пошуку ключа на пристрої.

Метод readPrivateKey завантажує пов’язані сертифікати (з PK_CERTIFICATES_FILE_NAMES). Якщо сертифікати не надано, а вказано CA Issuer CN, він намагається завантажити сертифікати з CMP (Certificate Management Protocol) сервера за допомогою методу GetCertificatesByKeyInfo бібліотеки EndUser. Метод endUser.SaveCertificate(certs.get(i)) зберігає знайдені сертифікати у внутрішньому сховищі сертифікатів бібліотеки. Рядок коду pkContext = endUser.CtxReadPrivateKeyBinary() або endUser.CtxReadPrivateKey(): Зчитує приватний ключ у контекст приватного ключа. Потім він намагається отримати сертифікати підписання (signCert) та шифрування (envCert), пов’язані з приватним ключем, з контексту. Нарешті, він оновлює налаштування TSP на основі емітента сертифіката підписання.

Методи криптографічних операцій – це публічні методи, які надають основні криптографічні послуги. Усі вони взаємодіють з об’єктом endUser та/або pkContext для виконання відповідних операцій. Вони обгортають потенційні винятки з базової бібліотеки у Exception з більш описовими повідомленнями. Метод noKeysWarn це простий допоміжний метод для логування попереджень під час ініціалізації, якщо очікувані параметри конфігурації не знайдені. Клас EndUserClient є центральним вузлом для взаємодії програми з криптографічною бібліотекою ІІТ. Він абстрагує складність конфігурації бібліотеки, управління з’єднаннями, завантаження ключів та сертифікатів, а також виконання фактичних криптографічних операцій. Реалізувавши його як синглтон, він забезпечує належне

управління ресурсами та єдину точку контролю для базової нативної бібліотеки. Цей клас є важливим для будь-якої програми, яка потребує виконання цифрового підписання, верифікації або хешування за допомогою криптографічного пакета IIT.Клас IITSignerController розташований у пакеті com.codot.signer.controllers і, судячи з назви та анотацій, є контролером Spring REST. Його основна функція полягає в отриманні HTTP-запитів від клієнтів і делегуванні їх обробки сервісу IITSignerService. Рядок коду package com.codot.signer.controllers; вказує пакет, до якого належить клас. Рядки import com.codot.signer.controllers.types.IITHashPayload; та import com.codot.signer.controllers.types.IITVerifyHashPayload; це імпорти DTO (Data Transfer Objects) (рис. 4.18). Ці класи використовуються для структурування вхідних JSON-даних від клієнта та для їх валідації. Рядок import com.codot.signer.services.IITSignerService; Контролер не створює IITSignerService самостійно, а просить Spring надати його, що робить код більш модульним та легшим для тестування.

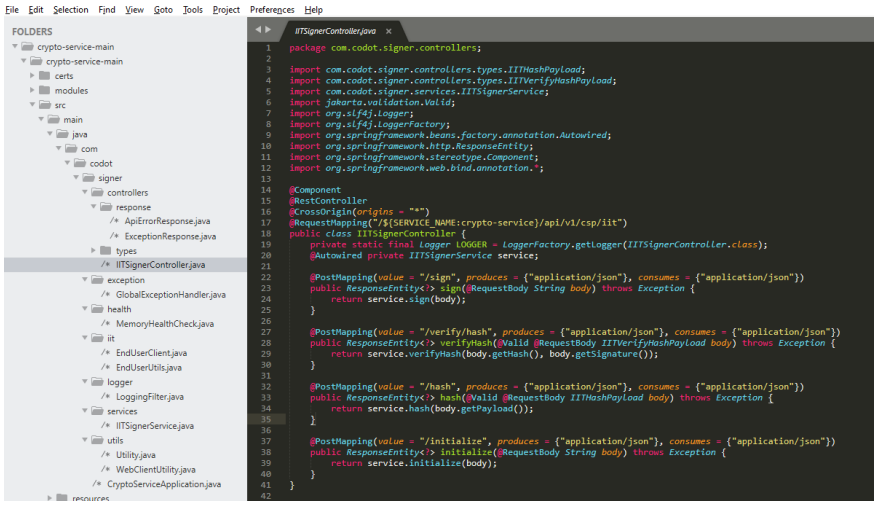


Рисунок 4.18 – Файл контролеру обробки запитів

Кожен метод у цьому контролері анотований @PostMapping, що означає, що він обробляє HTTP POST-запити. Вони також визначають:

– анотація @PostMapping(value = "/sign", ...) в якій шлях: /{SERVICE_NAME}/api/v1/csp/iit/sign має формат “вхід”: JSON-рядок у тілі запиту

(String body), “дія”: делегує виклик service.sign(body), «призначення»: надає REST-ендпоінт для підписання довільних даних;

– анотація @PostMapping(value = "/verify/hash", ...) в якій шлях: /{SERVICE_NAME}/api/v1/csp/iit/verify/hash має формат “вхід”: JSON-об’єкт, який автоматично мапується на IITVerifyHashPayload, (об’єкт очікує поля hash та signature. @Valid застосовує валідацію до IITVerifyHashPayload), «дія»: делегує виклик service.verifyHash(body.getHash(), body.getSignature()), «призначення»: надає REST-ендпоінт для перевірки відповідності хешу та підпису;

– анотація @PostMapping(value = "/hash", ...), в якій шлях: /{SERVICE_NAME}/api/v1/csp/iit/hash має формат «вхід»: JSON-об’єкт, який автоматично мапується на IITHashPayload (об’єкт очікує поле payload. @Valid застосовує валідацію до IITHashPayload), «дія»: делегує виклик service.hash(body.getPayload()), «призначення»: надає REST-ендпоінт для обчислення хешу даних;

– анотація @PostMapping(value = "/initialize", ...), в якій шлях: /{SERVICE_NAME}/api/v1/csp/iit/initialize має формат «вхід»: JSON-рядок у тілі запиту (String body) та містить конфігураційні дані для ініціалізації криптографічної бібліотеки, «дія»: делегує виклик service.initialize(body), «призначення»: надає REST-ендпоінт для початкової ініціалізації криптографічної бібліотеки, щл зазвичай має бути першим викликом до того, як будуть виконуватися інші криптографічні операції.

Клас IITSignerController є шлюзом HTTP API для криптографічних функцій, наданих IITSignerService. Він відповідає за отримання HTTP-запитів, мапування вхідних JSON-даних на Java-об’єкти (DTO), валідацію вхідних даних (завдяки @Valid), делегування фактичної бізнес-логіки (криптографічних операцій) до IITSignerService, формування HTTP-відповідей на основі результатів сервісу, налаштування базових шляхів та дозволів CORS. По суті, це типовий контролер у архітектурі Spring Boot RESTful API, який дотримується принципу «розділення відповідальності», де контролер обробляє лише вебрівень, а сервіс – бізнес-логіку.

4.4 Тестування продуктивності серверної частини

Тестування серверної частини програмного забезпечення є критично важливим етапом у життєвому циклі розробки, оскільки саме тут закладається фундамент стабільної та надійної роботи всієї системи. Без ретельного тестування бекенду, додаток може зіткнутися з непередбачуваними помилками, уповільненнями та навіть повними збоями, що призведе до невдоволення користувачів та репутаційних втрат. Це тестування охоплює перевірку баз даних, API, бізнес-логіки та інтеграції з іншими системами, забезпечуючи, що всі ці компоненти взаємодіють правильно і ефективно. Наприклад, неправильно оброблені запити до бази даних можуть призвести до втрати даних або неправильного відображення інформації. Забезпечення безпеки даних також є ключовим аспектом, адже вразливості на сервері можуть бути використані зловмисниками для несанкціонованого доступу. Ретельне тестування допомагає виявити та усунути ці загрози ще до розгортання системи. Інвестування в тестування серверної частини дозволяє уникнути значних витрат на виправлення помилок після випуску продукту. Це також сприяє підвищенню продуктивності, оскільки оптимізація коду та виправлення вузьких місць відбуваються на ранніх етапах (рис. 4.19).

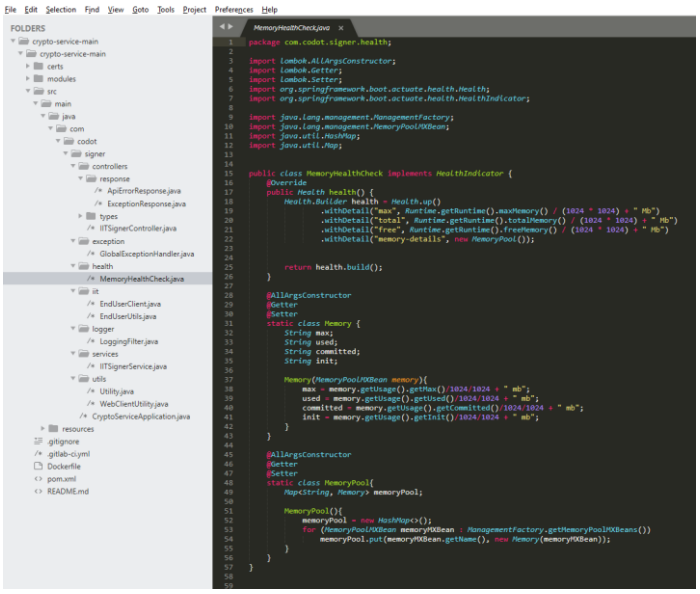


Рисунок 4.19 – Клас тестування пам’яті серверної частини в процесі роботи

Тестування серверної частини не обмежується лише пошуком помилок; воно також включає перевірку масштабованості та надійності системи під навантаженням. Проведення навантажувального тестування та стрес-тестування дозволяє визначити, як система поводиться при великій кількості одночасних користувачів або високих обсягах даних, що є особливо важливим для додатків з високим трафіком. Наприклад, електронна комерція або соціальні мережі потребують бекенду, який може витримувати пікові навантаження без зниження продуктивності. Тестування відмовостійкості гарантує, що система може відновитися після збою та мінімізувати час простою. Забезпечення високої доступності є пріоритетом, оскільки будь-який простій може мати значні фінансові наслідки. Усі ці аспекти тестування бекенду сприяють створенню стабільного, продуктивного та безпечного програмного продукту, що відповідає очікуванням користувачів і бізнесу. Комплексний підхід до тестування серверної частини допомагає створити програмне забезпечення, яке буде конкурентоспроможним на ринку та задовольнятиме потреби зростаючої аудиторії.

Клас MemoryHealthCheck у пакеті com.codot.signer.health призначений для моніторингу стану пам’яті Java-додатку за допомогою функціоналу Health Indicators (індикаторів стану) Spring Boot Actuator. Основне призначення коду на рис (Тестування пам’яті серверної частини в процесі роботи) Надати детальну інформацію про використання пам’яті JVM (Java Virtual Machine) як частину ендпоінту /health в Spring Boot Actuator. Це дозволяє системним адміністраторам або системам моніторингу швидко оцінити, чи додаток має проблеми з пам'яттю (наприклад, наближається до межі доступної пам’яті), що може бути індикатором потенційних збоїв або необхідності оптимізації. Інтерфейс HealthIndicator з Spring Boot Actuator. Будь-який клас, який імплементує цей інтерфейс, буде автоматично виявлений Actuator і його метод health() буде викликаний для збору інформації про стан. За замовчуванням, результат цього методу буде включено у відповідь на запит до ендпоінту /actuator/health (або просто /health, якщо не налаштовано management.endpoints.web.base-path). Клас Health:, який використовується для

створення об’єкта стану. Він має методи up() (для здорового стану), down() (для нездорового стану), unknown(), outOfService() та withDetail() для додавання додаткової інформації. Клас ManagementFactory та MemoryPoolMXBean з пакету java.lang.management, які надають доступ до внутрішньої інформації про JVM, включаючи управління пам'яттю, потоками, збирачем сміття тощо. Вони дозволяють отримати деталі про використання різних областей пам'яті. Клас MemoryHealthCheck імплементує HealthIndicator, що робить його кастомним індикатором стану для Spring Boot Actuator. Метод health() викликається Spring Boot Actuator для перевірки стану. Метод Health.up() початково стан встановлюється як «UP» (здоровий). Якщо під час збору даних не виникне винятків, він залишиться «UP». Метод .withDetail(«ключ», «значення») додає деталі до звіту про стан. Параметр max це максимальна кількість пам’яті, яку JVM може спробувати використовувати. Обчислюється як Runtime.getRuntime().maxMemory() і конвертується в мегабайти. Параметр total це загальна пам’ять, виділена JVM на даний момент. Обчислюється як Runtime.getRuntime().totalMemory(). Параметр free це кількість вільної пам'яті в JVM з виділеної загальної пам’яті. Обчислюється як Runtime.getRuntime().freeMemory(). Параметр memory-details замість простого рядка тут додається об’єкт new MemoryPool(). Це означає, що Actuator спробує серіалізувати цей об’єкт у JSON, надаючи більш детальну структуру інформації про пам’ять. Внутрішній статичний клас Memory моделює інформацію про використання одного конкретного пулу пам’яті JVM. У ньому поле max доя визначення аксимального розміру пулу пам’яті. Поле used визначає скільки пам’яті використовується в цьому пулі. Поле committed визначає кількість пам’яті, гарантовано доступної для JVM у цьому пулі. Поле init визначає початковий розмір пулу пам’яті. Конструктор Memory(MemoryPoolMXBean memory) приймає об’єкт MemoryPoolMXBean (який представляє один пул пам’яті) і витягує з нього відповідні значення використання пам’яті, конвертуючи їх у мегабайти.

Внутрішній статичний клас MemoryPool призначений для агрегації інформації про всі пули пам’яті в JVM. Поле memoryPool – це Map, де ключем є ім’я пулу пам’яті (наприклад, «G1 Eden Space», «G1 Old Gen»), а значенням – об’єкт Memory, що містить деталі використання для цього пулу. Конструктор MemoryPool() ініціалізує HashMap, використовує ManagementFactory.getMemoryPoolMXBeans() для отримання списку всіх MXBeans, які представляють пули пам’яті в поточній JVM, ітерує по кожному MemoryPoolMXBean і створює новий об’єкт Memory для нього, а потім додає його до мапи memoryPool з ім'ям пулу як ключем. Коли ви звернетеся до ендпоінту /actuator/health вашого Spring Boot додатка, відповідь JSON буде містити секцію даних деталі можуть відрізнятися залежно від JVM і GC. Статистика буде корисною для моніторингу продуктивності, що дозволятиме оперативно відстежувати використання пам’яті додатком. Клас дозволить виконувати виявлення витоків пам’яті. Постійно зростаюче використання used або committed пам’яті в пулах (особливо в «Old Gen») може вказувати на витік пам’яті, який потрібно дослідити та виправити. Корисність полягає також у діагностиці технічних проблем. При проблемах з продуктивністю або збоях, звіт /health надає швидку інформацію про стан пам’яті, що є першим кроком у діагностиці. Системи моніторингу (наприклад, Prometheus, Grafana, Zabbix) можуть автоматично опитувати цей ендпоінт і сповіщати, якщо використання пам’яті перевищує певні порогові значення. Контроль розподілу ресурсів допомагає зрозуміти, як JVM використовує пам’ять, і може бути корисним для налаштування параметрів JVM (наприклад, розміру купи -Xmx, -Xms) для оптимальної продуктивності. Отже, MemoryHealthCheck – це важливий компонент для забезпечення стабільності та спостережуваності (observability) Spring Boot програми, надаючи життєво важливу інформацію про стан пам’яті.

Статистика тестування продуктивності серверної частини є ключовою для розуміння того, наскільки ефективно сервер обробляє запити під різним навантаженням. Вона допомагає виявити «вузькі місця», оптимізувати

конфігурацію та переконатися, що система відповідає вимогам щодо швидкості та масштабованості.

Зазвичай, така статистика включає метрики, як-от:

– час відгуку (Response Time) – середній час, за який сервер обробляє запит і повертає відповідь. Важливо аналізувати середній час, медіану, 90-й або 95-й перцентиль (тобто, 90% або 95% запитів були оброблені швидше цього часу);

– пропускна здатність (Throughput) – кількість запитів, які сервер може обробити за одиницю часу (наприклад, запитів на секунду, транзакцій на годину);

– кількість помилок (Error Rate) – відсоток запитів, які завершилися помилкою (наприклад, HTTP 5xx помилки), тобто це критичний показник, який вказує на стабільність системи;

– використання ресурсів (Resource Utilization) – споживання ресурсів сервера (CPU, RAM, дисковий ввід/вивід, мережевий трафік) під навантаженням, що допомагає зрозуміти, чи не є якийсь ресурс «пляшковим горлечком»;

– кількість активних користувачів/запитів (Active Users/Requests) – максимальна кількість одночасних користувачів або запитів, які система може підтримувати, зберігаючи прийнятний час відгуку.

Ось приклад таблиці зі згенерованими тестовими даними для статистики тестування продуктивності серверної частини. Ці дані можуть бути результатом виконання навантажувального тесту за допомогою таких інструментів, як Apache JMeter, LoadRunner, k6, Locust тощо.

Було проведено тест навантаження API користувача (User API Load Test), результати якого наведені у табл. 4.1. Тест тривав 30 хвилин. Тестувався сценарій при якому виконувалась симуляція реєстрації, входу в систему з аналізом потенційних ризиків кібератаки та перегляду профілю користувача.

Таблиця 4.1 – Статистика тестування продуктивності серверної частини

Метрика / Параметр	Значення (100 віртуальних користувачів)	Значення (500 віртуальних користувачів)	Значення (1000 віртуальних користувачів)	Оптимальне значення / ціль
Кількість запитів, в тисячах	176	751	839	>1 000 000 (за 30 хв)
Пропускна здатність (RPS)	97	417	667	>500 RPS
Середній час відгуку	143	387	863	<500 мс
Медіана часу відгуку	119	297	703	<400 мс
90-й перцентиль часу відгуку	249	583	1504	<800 мс
95-й перцентиль часу відгуку	298	736	2204	<1000 мс
Кількість помилок (Total)	13	51	1507	0
Відсоток помилок	0	0.007	0.125	<0.1%
Використання CPU (сервер)	44	76	98	<80%
Використання RAM (сервер)	61	84	94	<90%
В/В диска (IOPS)	996	2491	3997	Згідно з вимогами
Мережевий трафік (МБ/с)	11	42	79	Залежить від функціоналу

З таблиці 4.1 видно, що при 100 віртуальних користувачах система демонструє відмінні показники: низький час відгуку, висока пропускна здатність і відсутність помилок. Використання ресурсів сервера знаходиться на прийнятному рівні. При 500 віртуальних користувачах продуктивність залишається хорошою. Час відгуку зростає, але все ще в межах прийнятних значень. З’являються поодинокі помилки, що може вказувати на початок проблем з масштабуванням. Використання CPU та RAM зростає. При 1000 віртуальних користувачах спостерігається значне погіршення показників. Час відгуку зростає в рази, 90-й та 95-й перцентилі виходять за межі оптимальних значень, вказуючи на те, що велика частина користувачів відчуває затримки. Кількість помилок зростає суттєво, що є критичним. Використання CPU досягає майже 100%, що вказує на перевантаження процесора як основну проблему.

Згідно з цими тестовими даними, серверна частина задовільно справляється з навантаженням до 500 віртуальних користувачів. При 1000 користувачах система стає нестабільною, час відгуку неприйнятний, і з'являється значна кількість помилок. Це вказує на необхідність оптимізації серверної частини, зокрема, масштабування ресурсів (CPU, RAM) або оптимізації коду/запитів до бази даних, щоб покращити продуктивність при високих навантаженнях.

4.5 Керівництво користувача

Після того, як вебсистему було протестовано необхідно розробити інструкції для користувачів, які будуть його використовувати. На рис. 4.20-а для автентифікації у вебсистемах треба приділяти увагу ризикам можливих кібератак.

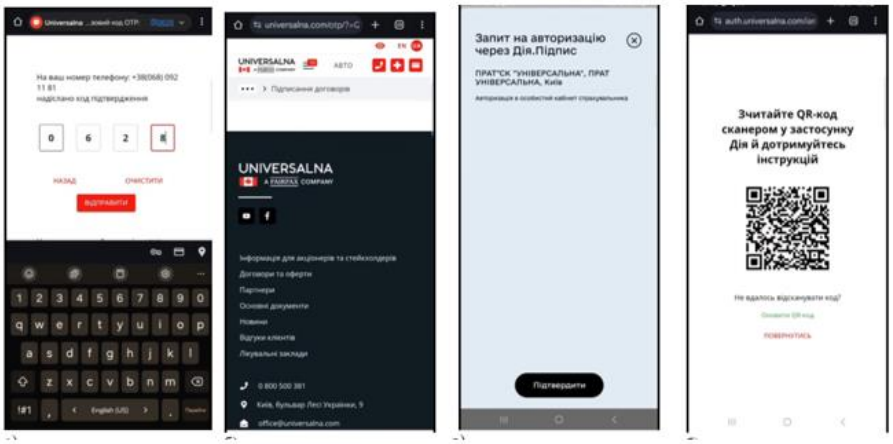


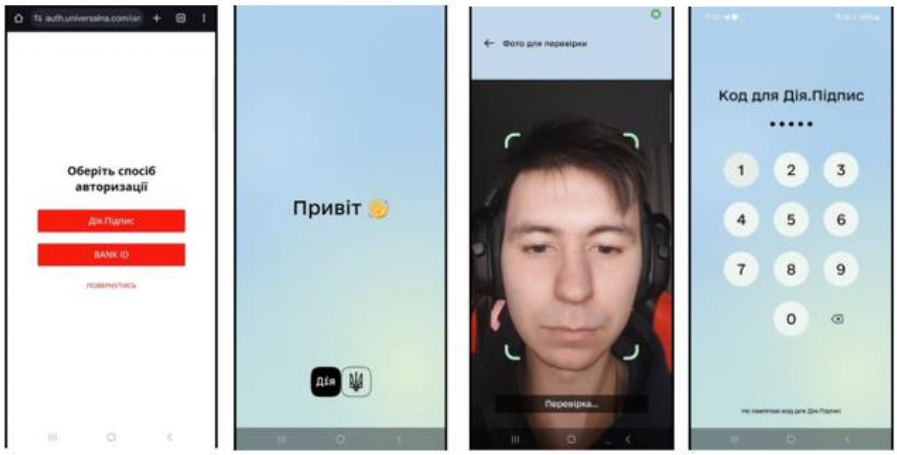
Рисунок 4.20 – а) інтерфейс для автентифікації користувача; б) успішна автентифікація; в, г) використання інтеграції «Дія» API

Відповідно до аналізу даних про користувача за допомогою алгоритму FastText був використаний механізм надсилання SMS-повідомлень з числовим кодом на смартфон на номер телефону, що закріплений за фізичною особою. На рис. 4.20-б після коректного вводу числового значення отриманого OTP коду користувач авторизується і потрапляє у вебсистему фінансової установи.

Для отримання особистих даних, які мають використовуватись у реквізитах фінансових контрактів в розробленому ПЗ використовується інтеграція з API

офіційного державного сервісу «Дія» (рис. 4.20-в). Для завершення процесу авторизації у «Дія» користувач має зісканувати QR-код та дотримуватись інструкцій (рис. 4.20-г).

Після обрання способу авторизації користувач успішно отримує повідомлення, що позначає початок використання «Дія» API в процесі адаптивної автентифікації (рис. 4.21). У подальшому сценарії відображення етапів верифікації буде виконуватись на основі результатів аналізу системних користувацьких браузерних даних.



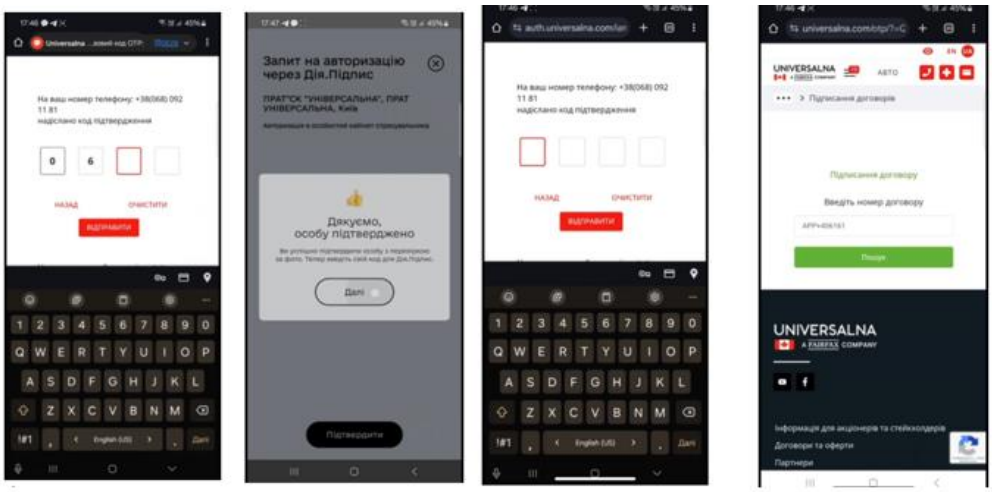
а) б) в) г)

Рисунок 4.21 – а, б) обрання способу авторизації за допомогою «Дія» API; в, г) перевірка з використанням сканування обличчя

Сканування обличчя в застосунку «Дія» – це процедура, яка використовується для верифікації особи, або для підтвердження ідентичності під час використання електронного підпису (Дія.Підпис) (рис. 4. 21-в). Вона включає зчитування даних особи за допомогою камери телефону та порівняння з фото в паспорті або інших документах, що є в системі.

Сканування обличчя використовується для підтвердження вашої ідентичності при реєстрації в застосунку, підписанні документів або інших операціях, де потрібна точна ідентифікація. Крім сканування обличчя на основі аналізу користувацьких даних «Дія» може використовувати додатковий код для Дія. Підпис (рис. 4.21-г)

При успішному підтвердження коду Дія.Підпис користувача можна додатково перевірити. На етапі перемикання між користувацькими інтерфейсами автентифікації Дія можливі сценарії виникнення загроз кібератак (рис. 4.22).



а) б) в) г)

Рисунок 4.22 – а,б) етапи адаптивної автентифікації з повторним використанням кодів підтвердження для взаємоінтегрованих систем

Підтвердження автентифікації за допомогою кодів та пошук фінансових договорів є необхідним, саме тому вебсистема адаптивної автентифікації обирає сценарій з підтвердженням на основі надісланого коду на номер телефону (рис 4.22-в).

Успішне підтвердження надає права користувачу виконати процес пошуку договору (рис. 4.23). Проте на етапі взаємодії між різними провайдерами даних фінансових установ можливі сценарії виникнення загроз кібератак. Саме тому необхідно виконувати аналіз системних користувацьких даних та застосовувати механізми адаптивної автентифікації.

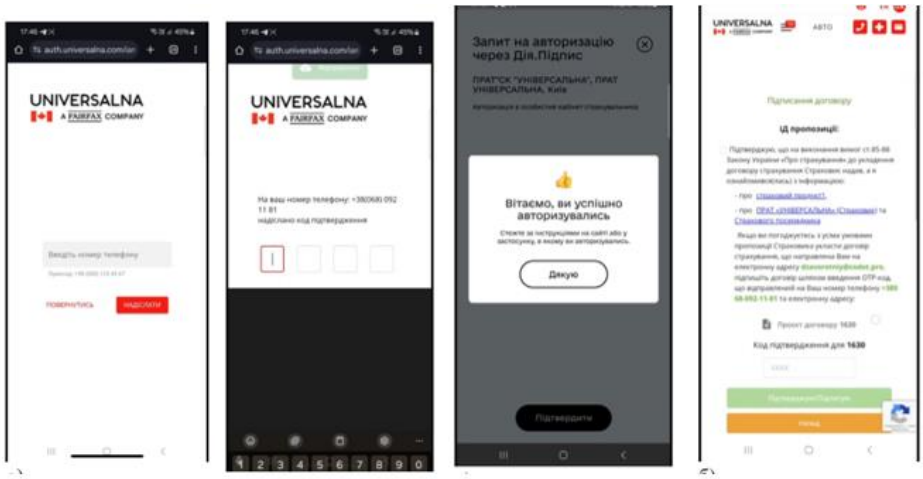


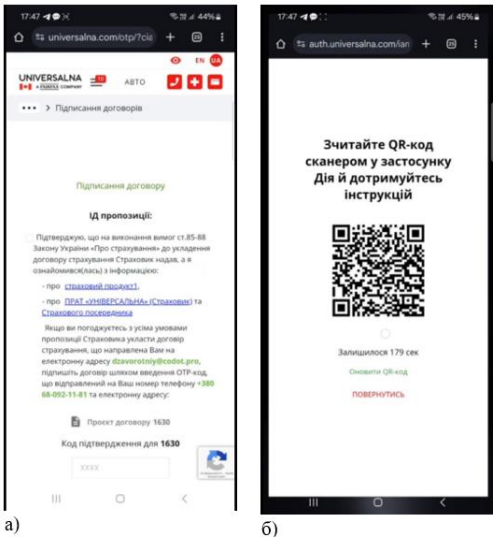
Рисунок 4.23 – Підтвердження операції пошуку фінансових договорів на основі кодів, що передаються на особистий номер телефону

Застосування адаптивної аутентифікації з авторизацією через Дія.Підпис підвищило надійність підписання договорів. Під час пошуку фінансових договорів використовуються користувацькі дані, які надаються через «Дія» API, тому що в договорах міститься інформація. З точки зору мережевої взаємодії є ймовірність загрози кібератак, тому механізми адаптивної автентифікації виконують сценарій підтвердження пошуку фінансового договору ще й через Дія.Підпис (рис. 4.23-в).

Користувач потрапляє на етап підписання договору (рис 4.23-г). Йому відображається інформація про законність дій. При цьому важливим є погодитись зі всіма умовами пропозиціями фінансових установ, щоб укласти договір. Обов’язковим атрибутом є e-mail та номер телефону. Користувач може ознайомитись з проєктом фінансового договору.

На завершальному етапі підтвердження і підписання фінансового договору також є можливість кібератаки. Тому використовуючи алгоритм FastText для класифікації рівня загрози можна вирішити проблему порушення процесу підписання фінансових договорів зловмисниками (рис. 4.24).

Вебсистема адаптивної автентифікації обирає сценарій вводу коду підтвердження через QR-код.



а) б)

Рисунок 4.24 – Підтвердження підписання фінансових договорів на основі аналізу користувацьких системних даних

Користувач відсканувавши QR–код у застосунку Дія та вклавшись у часові обмеження на виконання відповідної операції забезпечує надійність підписання фінансових договорів.

Висновки до розділу 4

Реалізація адаптивної автентифікації з використанням зазначеного стеку технологій є багатообіцяючим підходом до підвищення безпеки та зручності користувачів. Використання PostgreSQL як основної бази даних для зберігання користувацьких даних та інформації, пов'язаної з автентифікацією, забезпечує надійність, масштабованість та цілісність даних. Її реляційна структура дозволяє ефективно керувати складними зв'язками між користувачами, сесіями та параметрами безпеки. Здатність PostgreSQL обробляти великі обсяги даних і підтримувати транзакційність є критично важливою для системи автентифікації, яка постійно обробляє запити.

Spring Boot як фреймворк для розробки вебсервісу керування алгоритмами шифрування є чудовим вибором завдяки своїй простоті налаштування, широкому функціоналу та активній спільноті. Він дозволяє швидко розробляти та розгортати мікросервіси, що є ідеальним для винесеної логіки управління шифруванням.

Інтеграція Spring Security у Spring Boot значно спрощує реалізацію складних механізмів автентифікації та авторизації, включаючи адаптивні політики. Це дозволяє легко управляти ключами шифрування, алгоритмами та їхнім життєвим циклом, забезпечуючи гнучкість та безпеку системи.

Застосування комбінації Дія.Підпису та кодів з телефону для авторизації операцій є потужним багатофакторним підходом, що значно підвищує рівень безпеки. Дія.Підпис надає високий рівень довіри та юридичну значущість, що є особливо важливим для критичних фінансових або державних операцій. Додаткове підтвердження через SMS-код додає ще один незалежний фактор автентифікації, ускладнюючи несанкціонований доступ навіть у разі компрометації одного з факторів. Це створює надійний бар'єр проти фішингових атак та несанкціонованих транзакцій.

Інтеграція сканування обличчя як біометричного методу автентифікації є передовим кроком у підвищенні зручності та безпеки. Біометричні дані є унікальними для кожного користувача, що значно знижує ризик компрометації облікових записів. Хоча технології розпізнавання обличчя продовжують розвиватися, їх впровадження вимагає уваги до точності, швидкості та захисту від атак спуфінгу. Важливо також враховувати питання приватності та зберігання біометричних даних, які є особливо чутливими.

Аналіз браузерних даних користувача для адаптивної автентифікації додає контекстний шар безпеки. Цей метод дозволяє системі виявляти незвичну поведінку, таку як спроби входу з нового пристрою, незвичної геолокації або нестандартних шаблонів використання. Використання таких даних допомагає оцінити рівень ризику кожної спроби автентифікації та динамічно застосовувати додаткові перевірки, такі як вимога додаткового фактора автентифікації або блокування підозрілої активності. Це значно підвищує здатність системи адаптуватися до мінливих загроз.

Загалом, така архітектура адаптивної автентифікації створює багатошаровий захист, де кожен компонент виконує свою унікальну роль. PostgreSQL та Spring

Boot забезпечують міцну і гнучку технологічну основу. Комбінація Дія.Підпису та SMS-кодів гарантує надійну багатофакторну авторизацію, а біометрична автентифікація через сканування обличчя покращує користувацький досвід. Аналіз браузерних даних додає інтелект системі, дозволяючи їй динамічно реагувати на потенційні загрози. Це комплексна система, яка здатна забезпечити високий рівень безпеки та зручності для користувачів у сучасному цифровому середовищі. Постійний моніторинг та оновлення алгоритмів є ключовими для підтримання ефективності цієї системи.

ВИСНОВКИ

У ході виконання роботи було описано механізми автентифікації у фінансових установах, проведено огляд існуючих адаптивних методів автентифікації, розглянуто їх типи, переваги та недоліки кожного методу. Також було досліджено технологічні, операційні та бізнес-аспекти автентифікації, проаналізовано нормативно-правові вимоги, які впливають на проєктування і реалізацію відповідних систем у фінансовому секторі. Також було сформульовано актуальність, мету, об’єкт і предмет дослідження, визначено завдання для досягнення поставленої мети.

На відміну від традиційних статичних методів, адаптивна автентифікація застосовує різні рівні перевірки залежно від поточної ситуації, забезпечуючи оптимальний баланс між безпекою та зручністю користування системою. Комбінуючи різні методи автентифікації та динамічно регулюючи рівень перевірки залежно від оцінки ризику, адаптивні системи здатні ефективно протидіяти сучасним кіберзагрозам. Успішне впровадження адаптивної автентифікації у фінансових установах вимагає комплексного підходу, що враховує технічні, нормативні та соціальні аспекти. Подальший розвиток адаптивної автентифікації буде спрямований на підвищення точності оцінки ризиків, розширення спектру аналізованих факторів та забезпечення безперервної верифікації користувачів фінансових установ.

Алгоритми машинного навчання дозволяють створювати адаптивні системи автентифікації, які пристосовуються до змін у поведінці користувачів та нових загроз. Різні типи алгоритмів мають свої переваги і недоліки, тому оптимальним є комбінований підхід, що дозволяє максимізувати ефективність за ключовими метриками. FastText демонструє високу ефективність при роботі з текстовими даними та аналізі користувацьких запитів, що робить його перспективним для використання в системах адаптивної автентифікації. Але процес навчання та валідації моделей потребує ретельної підготовки даних та комплексної оцінки за множиною метрик. Інтеграція алгоритмів машинного навчання в системи

автентифікації фінансових установ дозволяє значно підвищити рівень безпеки та зменшити операційні витрати.

Реалізація адаптивної автентифікації з використанням зазначеного стеку технологій є багатообіцяючим підходом до підвищення безпеки та зручності користувачів. Використання PostgreSQL як основної бази даних для зберігання користувацьких даних та інформації, пов'язаної з автентифікацією, забезпечує надійність, масштабованість та цілісність даних. Її реляційна структура дозволяє ефективно керувати складними зв'язками між користувачами, сесіями та параметрами безпеки. Здатність PostgreSQL обробляти великі обсяги даних і підтримувати транзакційність є критично важливою для системи автентифікації, яка постійно обробляє запити.

Інтеграція Spring Security у Spring Boot значно спрощує реалізацію складних механізмів автентифікації та авторизації, включаючи адаптивні політики. Це дозволяє легко управляти ключами шифрування, алгоритмами та їхнім життєвим циклом, забезпечуючи гнучкість та безпеку системи. Застосування комбінації Дія.Підпису та кодів з телефону для авторизації операцій є потужним багатофакторним підходом, що значно підвищує рівень безпеки. Дія.Підпис надає високий рівень довіри та юридичну значущість, що є особливо важливим для критичних фінансових або державних операцій. Додаткове підтвердження через SMS-код додає ще один незалежний фактор автентифікації, ускладнюючи несанкціонований доступ навіть у разі компрометації одного з факторів.

Інтеграція сканування обличчя як біометричного методу автентифікації є передовим кроком у підвищенні зручності та безпеки. Аналіз браузерних даних користувача для адаптивної автентифікації додає контекстний шар безпеки. Метод дозволяє системі виявляти незвичну поведінку, таку як спроби входу з нового пристрою, незвичної геолокації або нестандартних шаблонів використання. Використання таких даних допомагає оцінити рівень ризику кожної спроби автентифікації та динамічно застосовувати додаткові перевірки, такі як вимога

додаткового фактора автентифікації або блокування підозрілої активності, що значно підвищує здатність системи адаптуватися до мінливих загроз.

Загалом, така архітектура адаптивної автентифікації створює багатошаровий захист, де кожен компонент виконує свою унікальну роль. PostgreSQL та Spring Boot забезпечують міцну і гнучку технологічну основу. Комбінація Дія.Підпису та SMS-кодів гарантує надійну багатофакторну авторизацію, а біометрична автентифікація через сканування обличчя покращує користувацький досвід. Було підвищено надійність вебсистеми автентифікації за рахунок адаптивності на основі використання моделей машинного навчання на менше ніж 0,1% при 1000 користувачів з 839 тис. запитів. Постійний моніторинг та оновлення алгоритмів є ключовими для підтримання ефективності цієї системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. Відомості Верховної Ради України. 2010. № 34. Ст. 481.

2. Положення про автентифікацію та застосування посиленої автентифікації на платіжному ринку : Постанова Правління Національного банку України від 03.05.2023 р. № 58. URL: https://bank.gov.ua/ua/legislation/Resolution_03052023_58 (дата звернення: 21.04.2025).

3. Про платіжні послуги : Закон України від 30.06.2021 р. № 1591-IX. Відомості Верховної Ради України. 2021. № 29. Ст. 234.

4. Положення про організацію заходів із забезпечення інформаційної безпеки в банківській системі України : Постанова Правління Національного банку України від 28.09.2017 р. № 95. URL: https://bank.gov.ua/ua/legislation/Resolution_28092017_95 (дата звернення: 21.04.2025).

5. Положення про кіберзахист та інформаційну безпеку в платіжних системах та системах розрахунків : Постанова Правління Національного банку України від 19.09.2019 р. № 122. URL: https://bank.gov.ua/ua/legislation/Resolution_19092019_122 (дата звернення: 24.04.2025).

6. Payment Card Industry Data Security Standard (PCI DSS). Version 4.0. PCI Security Standards Council, 2022. 360 p.

7. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection — Information security management systems — Requirements. International Organization for Standardization, 2022. 28 p.

8. Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market (PSD2). Official Journal of the European Union. 2015. L 337. P. 35-127.

9. What is Authentication? Auth0 : website. URL: https://auth0.com/intro-to-iam/what-is-authentication (Last accessed: 20.03.2025).

10. Shanthi Vineela. What is Authentication? – Types, Role & How it Works! Teamwin Global Technologica : website. URL: https://teamwin.in/index.php/2024/09/28/what-is-authentication-types-role-how-it-works/ (Last accessed: 20.03.2025).

11. Галета М.С., Денисюк О.В. Технології адаптивної автентифікації в інформаційних системах фінансових установ. Кібербезпека: освіта, наука, техніка. 2024. № 1(14). С. 72-85.

12. Коваленко Д.В. Поведінкова біометрія як засіб підвищення безпеки банківських систем. Інформаційні технології та комп'ютерна інженерія. 2024. № 2. С. 45-59.

13. Бойко А.І., Мельник С.П. Машинне навчання у системах автентифікації фінансового сектору. Вісник Національного банку України. 2023. № 3. С. 132-148.

14. Шевченко О.В. Багатофакторна автентифікація: сучасні підходи та виклики впровадження. Системи обробки інформації. 2023. № 4(172). С. 94-107.

15. Tiwari A., Sanyal G. A Multi-factor Security Framework for Cloud Computing. Journal of Cloud Computing. 2022. Vol. 11, No. 2. P. 143-167.

16. Chandola V., Banerjee A., Kumar V. Anomaly detection: A survey. ACM Computing Surveys. 2022. Vol. 41, No. 3. P. 15:1-15:58.

17. Bhatt S., Patwa P., Battu V. Secure adaptive authentication system using behavioral biometrics. Journal of Information Security and Applications. 2023. Vol. 62. P. 102994.

18. Yang L., Guo Y., Ding X. A Comprehensive Review of Adaptive Authentication Systems for Financial Applications. IEEE Access. 2023. Vol. 11. P. 9437-9451.

19. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics. 2017. Vol. 5. P. 135-146.

20. Joulin A., Grave E., Bojanowski P., Mikolov T. Bag of Tricks for Efficient Text Classification. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. 2017. Vol. 2. P. 427-431.

21. Miramirkhani N., Starov O., Nikiforakis N. Snakeoil: Detecting Fake AI-Generated Profiles Through Language and Behavioral Analysis. Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021. P. 1234-1248.

22. Jain A., Kanhangad V. Investigating the effectiveness of different feature selection methods for keystroke dynamics-based user authentication. Journal of Intelligent Information Systems. 2020. Vol. 54. P. 275-296.

23. Arora S., Yingyu L., Ma T. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. International Conference on Learning Representations. 2017. P. 1-16.

24. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 2013. Vol. 26. P. 3111-3119.

25. Phillips P., Flynn P., Scruggs T. Overview of the face recognition grand challenge. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2021. Vol. 1. P. 947-954.

26. Фаулер М. UML. Основи: Короткий посібник по стандартній мові об'єктного моделювання / пер. з англ. А. Петренко. Київ : Діалектика, 2021. 192 с.

27. Кузнєцова Н. В. Алгоритми адаптивної автентифікації користувачів інформаційних систем на основі аналізу поведінкових паттернів. Кібербезпека: освіта, наука, техніка. 2021. № 4. С. 60-72.

28. Федорчук Є. Н. Проєктування компонентної архітектури веб-систем з використанням діаграм UML. Системи обробки інформації. 2022. № 3(170). С. 80-89.

29. Дудко М. В. UML-моделювання програмних систем : навч. посіб. / М. В. Дудко, С. В. Шаров. — Мелітополь : Люкс, 2020. — 127 с.

30. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler. — Addison-Wesley Professional, 2019. — 208 p.

31. Ambler S. The Object Primer: Agile Model-Driven Development with UML 2.0 / S. Ambler. — Cambridge University Press, 2018. — 572 p.

32. Kulba V. Sequence Diagrams: Modeling, Analysis and Applications / V. Kulba, A. Shelkov // Cybernetics and Systems Analysis. — 2020. — Vol. 56, No. 4. — P. 607–617.

33. Заворотній Д. О., Бурлаченко І. С. Класифікація кібератак з використанням моделі FastText. Збірник тез XXII Міжнародної науково-практичної конференції. Ольвійський форум – 2025 : стратегії країн Причорноморського регіону в геополітичному просторі. Технічні науки та інженерія : XXII Міжнар. наук. конф. 16-21 червня 2025 р., м. Миколаїв : тези / М-во освіти і науки України. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2025.

ДОДАТОК А

Лістинг програмного коду

iam-service-main\iam-service-main\otp_widget\src\pages\BankIdFlow.vue

```
<template>
  <div :class="['container-bank', { 'container-bank-right': !state.showHint }]">
    <div :class="['box1', { 'box1-right': !state.showHint }]">
      <div class="logo-wrap">
        <img class="logo-img" :src="state.logoUrl" alt="logo" />
        <!-- </a>
        <a :href="state.href"> -->
      </div>
      <div class="title-wrap-left">
        <p class="title-bank">Оберіть банк</p>
      </div>
      <q-select
        outlined
        v-model="model"
        :options="banks"
        :label-slot="true"
        label="Оберіть банк"
        behavior="menu"
        :option-value="
          (opt) => (Object(opt) === opt && 'id' in opt ? opt.id : null)
        "
        :option-label="
          (opt) =>
            Object(opt) === opt && 'name' in opt ? opt.name : '- Null -'
        "
      >
        <template v-slot:option="scope">
          <q-item v-bind="scope.itemProps">
            <div class="opt-wrap">
              <div class="img-opt-wrap">
                <img
                  :src="`https://id.bank.gov.ua/` + scope.opt.logoUrl"
                  alt="privat"
                  style="max-height: 100px; max-width: 140px"
                />
              </div>
              <div class="text-opt-wrap">
                <p class="opt-text">{{ scope.opt.name }}</p>
              </div>
            </div>
          </q-item>
        </template>
      </q-select>
      <div class="btn-bank-wrap">
        <q-btn color="primary" flat label="Повернутись" @click="onGoBack" />
        <q-btn
          label="Авторизуватись"
          type="submit"
          color="primary"
          class="q-ml-sm"
        />
      </div>
    </div>

    <div :class="['box2', { 'box2-right': !state.showHint }]">
      <!-- <div class="logo-wrap-left"></div> -->
```

```
      <div class="title-wrap">
        <p class="title-bank">Як це працює</p>
      </div>
      <div class="text-wrap">
        <p class="text-bank">
           Сервіс надається Національним банком України та можливий лише
          для клієнтів тих банків, які його підтримують. <br />
            Після обрання свого банку ви будете переадресовані на його сайт
          для проходження автентифікації з використанням логіна, пароля, номера
          картки. <br />
            У разі успішної автентифікації на сайті банку, система Bank ID
          передасть ваші персональні дані, що дозволить вас ідентифікувати.
        </p>
      </div>
    </div>
  </div>

  <!-- spiner -->
  <div v-if="loading" class="loader-box">
    <q-spinner-hourglass color="primary" size="4em" />
  </div>
</template>

<script>
import { useQuasar } from "quasar";
import store from "../store";
import api from "../services";
import banksName from "./banks.js";
import { ref } from "vue";

export default {
  name: "BankId",

  components: {},
  data() {
    return {
      loading: false,
    };
  },
  methods: {
    //phone btns
    async onSubmit() {
      this.loading = true;
      if (this.phone === "" || this.phone.length !== 15) {
        this.showCode = false;
        this.showNotify(false);
        this.loading = false;
      } else {
        const res = await api.postOtpSend(this.phone);
        if (res.status === 200) {
          this.showCode = true;
          this.showNotify(true);
          this.loading = false;
          return;
        }
        this.showCode = false;
        this.showNotify(false);
        this.loading = false;
      }
    },
```

```
onGoBack() {
  if (this.state.subFlow) {
    this.state.subFlow = "";
    return;
  }
  window.location.replace(this.state.returnUrl);
},

//code btns
async onSubmitCode() {
  console.log("send code");
  this.loading = true;

  const code = this.code1 + this.code2 + this.code3 + this.code4;
  if (code === "" || code.length !== 4) {
    this.showNotifyCode(false);
    this.loading = false;
  } else {
    const res = await api.postOtpCheck(code);
    if (res?.access_token) {
      this.token = res.access_token;
      this.showNotify(true);
      const redirectUrl = this.state.urlFromCiam.replace(
        ":token",
        this.token
      );
      this.loading = false;

      window.location.href = `${redirectUrl}`;
      return;
    }
  }
},
onResetCode() {
  this.code1 = "";
  this.code2 = "";
  this.code3 = "";
  this.code4 = "";
},

focusNext(e) {
  console.log(e);
  const inputs = Array.from(
    e.target.form.querySelectorAll('input[type="text"]')
  );
  const index = inputs.indexOf(e.target);

  if (index < inputs.length) {
    inputs[index + 1].focus();
  }
},
goBack() {
  this.showCode = false;
  this.phone = "";
  this.code1 = "";
  this.code2 = "";
  this.code3 = "";
  this.code4 = "";
  this.loading = false;
},
```

```
},

async mounted() {},

async beforeMount() {},
setup() {
  const $q = useQuasar();
  const { state } = store;
  const banks = banksName.banksJson;

  const url = window.location.href;
  const checkUrl = url.includes("?");
  if (checkUrl) {
    const inx = url.indexOf("?");
    const [slisedUrl, _] = url
      .slice(inx + 1)
      .replace("redirect_uri=", "")
      .split("&");
    state.urlFromCiam = slisedUrl;
  }
  return {
    model: ref(null),
    showNotify(bool) {
      if (!bool) {
        $q.notify({
          color: "red-5",
          textColor: "white",
          position: "top",
          icon: "warning",
          timeout: 100,
          message: "Введіть номер телефону",
        });
      } else {
        $q.notify({
          color: "green-4",
          timeout: 100,
          position: "top",
          textColor: "white",
          icon: "cloud_done",
          message: "Відправлено",
        });
      }
    },
    showNotifyCode(bool) {
      if (!bool) {
        $q.notify({
          color: "red-5",
          timeout: 100,
          textColor: "white",
          position: "top",
          icon: "warning",
          message: "Введіть код",
        });
      } else {
        $q.notify({
          color: "green-4",
          timeout: 100,
          position: "top",
          textColor: "white",
          icon: "cloud_done",
```

```
            message: "Відправлено",
          });
        }
      },
      state,
      banks,
    };
  },
};
</script>
```

iam-service-main\otp_widget\src\pages\DiyaPage.vue

```
<template>
  <div :class="['container-otp', { 'container-otp-right': !state.showHint }]">
    <div :class="['box1', { 'box1-right': !state.showHint }]">
      <div
        class="q-pa-md form-wrap"
        style="max-width: 800px; margin: auto; overflow: auto"
      >
        <div style="margin-bottom: 20px">
          <p style="font-size: 1.5rem; font-weight: bold; text-align: center">
            Зчитайте QR-код сканером у застосунку Дія й дотримуйтесь інструкцій
          </p>
        </div>
        <div
          v-if="this.state.desktop"
          style="margin-top: 5vh; display: flex; justify-content: center"
        >
          <qrcode-vue :value="this.state.deepLink" :size="sizeQr" level="L" />
        </div>
        <div
          v-if="!this.state.desktop"
          style="margin-top: 5vh; display: flex; justify-content: center"
        >
          <a :href="this.state.mobDeepLink" target="_blank">
            <qrcode-vue :value="this.state.deepLink" :size="sizeQr" level="L" />
          </a>
        </div>
        <div class="resend-wrap">
          <div
            v-if="timerVal"
            style="
              height: 42px;
              display: flex;
              align-items: center;
              justify-content: center;
            "
          >
            <p>Залишилося {{ timerVal }} сек</p>
          </div>
          <div
            v-if="!timerVal"
            style="
              display: flex;
              justify-content: center;
              flex-direction: column;
            "
          >
            <p style="text-align: center; margin-bottom: 10px">
              Не вдалось відсканувати код?
```

```
            </p>
          </div>
        </div>
        <div style="display: flex; justify-content: center">
          <q-btn
            text-color="green"
            no-caps
            @click="resendQr"
            unelevated
            label="Оновити QR-код"
          />
        </div>
        <div class="btn-code-wrap" style="margin-top: 1vh">
          <q-btn color="primary" flat label="Повернутись" @click="onGoBack" />
        </div>
      </div>
    </div>

    <div :class="['box2', { 'box2-right': !state.showHint }]">
      <!-- <div class="logo-wrap-left"></div> -->
      <div class="title-wrap">
        <p class="title-bank">Як це працює</p>
      </div>
      <div class="text-wrap">
        <p class="text-otp">
           QR-код, к'юар-код (скорочено від англ. quick response code, «код
          швидкої відповіді») — матричний код (двовимірний штрих-код),
          розроблений і представлений японською компанією «Denso-Wave» в 1994
          році.
          <br />
           Відкрийте вбудований додаток Камера на сумісному телефоні чи
          планшеті Android. Наведіть камеру на QR-код. Натисніть банер, який
          з'явиться. Щоб завершити вхід, виконайте вказівки на екрані.
        </p>
      </div>
    </div>
  </div>

  <!-- spiner -->
  <div v-if="loading" class="loader-box">
    <q-spinner-hourglass color="primary" size="4em" />
  </div>
</template>

<script>
import { useQuasar } from "quasar";
import QrcodeVue from "qrcode.vue";
import store from "../store";
import api from "../services";
import { startSocket } from "../socket";
import { ref, watch } from "vue";

export default {
  name: "DiyaPage",

  components: {
    QrcodeVue,
  },
  data() {
    return {
```

```
    sizeQr: 180,
    timerVal: "",
    loading: false,
    timer: null,
  };
},
methods: {
  async onGoBack() {
    if (this.state.subFlow) {
      this.state.subFlow = "";
      const value = {
        sessionId: this.state.sesionId,
        step: this.state.step,
      };
      const res = await api.postActionsCancel(value);
      if (res.statusCode === 200) {
        this.state.step = res.step;
        this.state.page = 1;
        this.state.socket.disconnect();
        this.loading = false;
      } else {
        console.log("error cancel");
      }
      return;
    }
    window.location.replace(this.state.returnUrl);
  },

  async resendQr() {
    this.loading = true;
    this.state.socket.disconnect();
    this.state.socket = null;
    clearInterval(this.timer);
    this.timer = null;
    // this.startTimer(10000);
    this.startTimer(180000);

    const value = {
      sessionId: this.state.sesionId,
      step: this.state.step,
    };
    const res = await api.postActionsResent(value);
    if (res.statusCode === 200) {
      this.state.deepLink = res.deeplink;
      const id = res.requestId;
      this.state.socket = startSocket(id, this.state.sesionId);
      this.socketFlag += 1;
      this.state.step = res.step;
      this.showNotifyResent(true);
    } else {
      this.showNotifyResent(false);
    }

    this.loading = false;
  },
  startTimer(delay = 30000) {
    let seconds = +delay / 1000;
    this.timer = setInterval(() => {
      this.timerVal = seconds--;
      if (!this.timerVal) {
```

```
        clearInterval(this.timer);
      }
    }, 1000);
  },
},

async mounted() {
  this.state.socket.on("diia-callback", async (data) => {
    if (data.status == 200) {
      //complete sessionID step
      this.loading = true;

      const value = {
        sessionId: this.state.sesionId,
        step: this.state.step,
      };
      const res = await api.postActionsComplete(value);

      if (res.statusCode == 200) {
        if (res.redirectUrl) {
          // this.loading = false;
          window.location.href = `${res.redirectUrl}`;
          return;
        } else {
          window.location.reload();
          return;
        }
      }
      if (res.statusCode == 400) {
        this.loading = false;
        this.showErrorRes(res.message || "Сталась помилка авторизації");
        return;
      }
      if (data.status == 401) {
        this.loading = false;
        showErrorRes("Помилка авторизації: не збігається поле іпн");
        return;
      }
    }
    showErrorRes(data.message);
    console.log("error auth diia", data.message);
  });
},

async beforeMount() {
  this.startTimer(180000);
  // this.startTimer(10000);
},

setup() {
  const $q = useQuasar();
  const { state } = store;
  const socketFlag = ref(1);

  watch(socketFlag, async (oldSocketFlag, newSocketFlag) => {
    console.log("resend", newSocketFlag);
    state.socket.on("diia-callback", async (data) => {
      if (data.status == 200) {
        //complete sessionID step
        this.loading = true;
```

```
      const value = {
        sessionId: state.sesionId,
        step: state.step,
      };
      const res = await api.postActionsComplete(value);

      if (res.statusCode == 200) {
        if (res.redirectUrl) {
          // this.loading = false;
          window.location.href = `${res.redirectUrl}`;
          return;
        } else {
          window.location.reload();
          return;
        }
      }
      if (res.statusCode == 400) {
        this.loading = false;
        showErrorRes(res.message || "Сталась помилка авторизації");
        return;
      }
      if (data.status == 401) {
        this.loading = false;
        showErrorRes("Помилка авторизації: не збігається поле іпн");
        return;
      }
    }
    showErrorRes(data.message);
    console.log("error auth diia", data.message);
  });
});

return {
  showNotify(bool) {
    if (!bool) {
      $q.notify({
        color: "red-5",
        textColor: "white",
        position: "top",
        icon: "warning",
        timeout: 100,
        message: "Введіть номер телефону",
      });
    } else {
      $q.notify({
        color: "green-4",
        timeout: 300,
        position: "top",
        textColor: "white",
        icon: "cloud_done",
        message: "Відправлено",
      });
    }
  },
  showNotifyCode(bool) {
    if (!bool) {
      $q.notify({
        color: "red-5",
        timeout: 100,
```

```
            textColor: "white",
            position: "top",
            icon: "warning",
            message: "Введіть код",
          });
        } else {
          $q.notify({
            color: "green-4",
            timeout: 300,
            position: "top",
            textColor: "white",
            icon: "cloud_done",
            message: "Відправлено",
          });
        }
      },
      showNotifyResent(bool) {
        if (!bool) {
          $q.notify({
            color: "red-5",
            timeout: 500,
            textColor: "white",
            position: "top",
            icon: "warning",
            message: "Сталась помилка",
          });
        } else {
          $q.notify({
            color: "green-4",
            timeout: 700,
            position: "top",
            textColor: "white",
            icon: "cloud_done",
            message: "Новий QR-код сформовано",
          });
        }
      },
      showErrorRes(str) {
        $q.notify({
          color: "red-5",
          timeout: 1700,
          textColor: "white",
          position: "top",
          icon: "warning",
          message: str,
        });
      },
      state,
      watch,
      socketFlag,
    };
  },
};
</script>
```

iam-service-main\src\idp\diia\diia.controller.ts

```
import {
    Body,
    Controller,
    Delete,
```

```
    Get,
    HttpCode,
    HttpStatus,
    Logger,
    Post,
    Put,
    Query,
    Req,
} from '@nestjs/common';
import { Request } from 'express';
import { v4 as uuid } from 'uuid';
import { FormDataRequest, MemoryStoredFile } from 'nestjs-form-data';
import { WidgetGateway } from 'src/widget/widget.gateway';
import { DiiaService } from './diia.service';
import ISession from 'src/auth/types/session.interface';
import { UserService } from 'src/user/user.service';
import IAuthToken from 'src/auth/types/auth-token.interface';
import { SessionData } from 'express-session';
import { ConfigService } from '@nestjs/config';

@Controller('/api/v1/idp/diia')
export class DiiaController {
    private readonly logger = new Logger(DiiaController.name);
    public readonly ENVIRONMENT: string;

    constructor(
        private readonly config: ConfigService,
        private readonly widgetSocket: WidgetGateway,
        private readonly diiaService: DiiaService,
        private readonly userService: UserService
    ) {
        this.ENVIRONMENT = config.get<string>('server.environment');
    }

    @HttpCode(HttpStatus.OK)
    @Post('documents-receiver')
    @FormDataRequest({ storage: MemoryStoredFile })
    async documentsReceiver(@Body() dto: IDiiaCallback, @Req() req: Request) {
        try {
            // !! ALREADY HASHED REQUESTID
            const { requestId, signature }: IDiiaEncodedData = JSON.parse(
                atob(dto.encodeData)
            );

            const res = this.widgetSocket.activeConnections.get(requestId);
            // this.logger.log('res: ' + res);
            const { clientId: clientId, sessionId } = res;

            let session = (await this.diiaService.getSession(
                sessionId,
                req
            )) as ISession;

            // ! MOCK
            let subjCN = 'Дія Надія Володимирівна';
            let subjDRFOCode = session.newUser?.attributes?.user_tax_code;
            if (this.diiaService.ENVIRONMENT === 'production') {
                const resParseSignature = await this.diiaService.getVerifyHash(
                    requestId,
                    signature
```

```
                );
                subjCN = resParseSignature.subjCN;
                subjDRFOCode = [resParseSignature.subjDRFOCode];
                this.logger.log('ПІБ: ' + subjCN);
                this.logger.log('ІНН: ' + subjDRFOCode);
            }
            const sub = (subjCN as string).split(' ');

            //-TODO: CHECK tax code from diia for find already exists users
            //-! IF user exist by this tax code -> update OLD account, using received data
            // ! UPD -> if users exists, tax code from customerId mest be EQUAL tax code
from diia


            // TODO: transfer "CREATE USER" logic to conrirm in next iterations
            const isNewUser = !!session.newUser?.inProgress;
            // todo ПЕРЕВІРКА НА CUSTOMER_ID === DIIA_TAX_CODE
            if (isNewUser && session.newUser?.attributes?.user_tax_code !== subjDRFOCode){
                await this.widgetSocket.server
                    .to(clientId)
                    .emit('diia-callback', { status: 401, message: 'The received TAX code
is not equal to the user TAX code' });

                this.widgetSocket.activeConnections.delete(requestId);
                return {
                    success: true,
                };
            }
            if (isNewUser) {
                const loginResponse = await this.userService.create(
                    {
                        emailVerified: false,
                        enabled: true,
                        firstName: sub[1],
                        lastName: sub[0],
                        username: uuid(),
                        attributes: {
                            user_email: session.newUser.attributes.user_email,
                            user_registration_date: [new Date().toISOString()],
                            user_phone: session.newUser.attributes.user_phone,
                            user_full_name: [subjCN],
                            user_first_name: [sub[1]],
                            user_middle_name: [sub[2]],
                            user_last_name: [sub[0]],
                            user_ext_code_ecom: undefined,
                            user_ext_code_incore: undefined,
                            user_last_trusted_login: [new Date().toISOString()],
                            user_tax_code: subjDRFOCode,
                        },
                        email: undefined,
                        groups: undefined,
                        requiredActions: undefined,
                    },
                    session.authParams.realm
                );

                // IF session.userinfo.authorization.exists !== undefined -> make old
accaunt "enabled: false"
                if (!session.userinfo.authorization.exists)
                    await this.userService.update(
```

```
                        {
                            enabled: false
                        },
                        session.authParams.realm,
                        session.userinfo.authorization.exists
                    );

                const ott = uuid();
                const auth: IAuthToken = {
                    authSession: session.authParams.authSession,
                    access_token: loginResponse.access_token,
                    refresh_token: loginResponse.refresh_token,
                    ott: ott,
                };
                session.auth = auth;
                (await this.diiaService.updateSession(
                    sessionId,
                    session as SessionData,
                    req
                )) as ISession;
            } else {
                if (!session.userinfo?.user?.sub)
                    this.logger.log(
                        `Empty sub: ${session.userinfo?.user?.sub}`
                    );

                const attributes = await this.userService.getUserAttributes(
                    session.userinfo?.user?.sub
                );

                // if TAX code DEFINED && code not equal TAX code from DIIA -> return 401
exception
                if(this.ENVIRONMENT  ===  "production"  &&  attributes.user_tax_code  !==
undefined && attributes.user_tax_code !== subjDRFOCode){
                    await this.widgetSocket.server
                        .to(clientId)
                        .emit('diia-callback', { status: 401, message: 'The received TAX
code is not equal to the user TAX code' });

                    this.widgetSocket.activeConnections.delete(requestId);
                    return {
                        success: true,
                    };
                }

                //    this.logger.log(`Update    'user_last_trusted_login'    ->    userId
(${session.userinfo?.user?.sub})`)
                await this.userService.update(
                    {
                        attributes: {
                            ...attributes,
                            user_last_trusted_login: [new Date().toISOString()],
                        },
                    },
                    session.authParams.realm,
                    session.userinfo?.user?.sub
                );
            }

            await this.widgetSocket.server
```

```
                .to(clientId)
                .emit('diia-callback', { status: 200, message: 'OK' });

            this.widgetSocket.activeConnections.delete(requestId);
            return {
                success: true,
            };
        } catch (e) {
            this.logger.error(e);
            return {
                success: false,
            };
        }
    }
}
```