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, д.т.н., проф.,
_____Ю.П.Кондратенко
«____»_____2022 року

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

WEB-ЗАСТОСУНОК ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЙНОЇ ДІЯЛЬНОСТІ В ІТ-КОМПАНІЇ

Спеціальність 124 «Системний аналіз»

124 – МКР – 607.21610301

Студент _____О. С. Бабін
«____» лютого 2022 р.
Консультант _____ Н. М. Болюбаш
канд. пед. наук, доцент
«____» лютого 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **124 «Системний аналіз»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____ 20__ р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу

Бабіну Олександру Сергійовичу

1. Тема магістерської кваліфікаційної роботи «Web-застосунок забезпечення комунікаційної діяльності в IT-компанії».

Керівник роботи Болюбаш Надія Миколаївна, к. пед. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «__» ____ 20__ р. № _____

2. Строк подання студентом роботи ___ _____ 20__ р.

3. Вхідні (початкові) дані до роботи: загальні відомості про спілкування всередині IT-компанії та забезпечення безпеки під час обміну інформацією, дані про працівників, їх цифрові сліди та повідомлення.

Очікуваний результат роботи: веб-застосунок забезпечення комунікаційної діяльності в IT-компанії з надійною системою шифрування внутрішньо корпоративної інформації, автоматизація процесів HR.

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

- розкриття теоретичних засад комунікаційної діяльності ІТ-компаній та сучасного стану програмного забезпечення, яке використовується для її підтримки;
- дослідження підходів до забезпечення надійності захисту корпоративної інформації;
- обґрунтуваня вибору технологій і засобів розробки Web-застосунку для забезпечення комунікаційної діяльності;
- розробка та здійснення програмної реалізації Web-застосунку для забезпечення комунікаційної діяльності ІТ-компанії з надійною системою шифрування інформації.

5. Перелік графічного матеріалу: презентація, рисунки, таблиці.

6. Завдання до спеціальної частини: Аналіз умов праці та навчання при надзвичайних ситуаціях.

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	д.б.н.,професор Л. І. Григор'єва	
Методична частина	к.пед.н., доцент Н.М. Болюбаш	

Керівник роботи __к. пед. наук, доц. Болюбаш Н. М._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Бабін О. С._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «_____» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН

виконання магістерської кваліфікаційної роботи

Тема: «Web-застосунок забезпечення комунікаційної діяльності в IT-компанії»

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Отримання завдання на магістерську наукову роботу	18.10.2021	01.11.2021	Виконано
2.	Огляд літератури за темою роботи	01.11.2021	22.11.2021	Виконано
3.	Аналіз існуючих ПЗ для забезпечення комунікацій всередині компаній	22.11.2021	26.11.2021	Виконано
4.	Аналіз предметної області та методів забезпечення надійного захисту інформації	01.12.2021	13.12.2021	Виконано
5.	Робота над першим та другим розділом пояснювальної записки	14.12.2021	27.12.2021	Виконано
6.	Створення дизайну, проектування та програмна реалізація Web-додатку	28.12.2021	04.01.2022	Виконано
7.	Робота над третім розділом пояснювальної записки	04.012022	14.01.2022	Виконано
8.	Тестування системи	18.01.2022	24.01.2022	Виконано
9.	Розробка методичної частини	24.01.2022	25.01.2022	Виконано
10.	Розробка спеціальної частини з безпеки в надзвичайних ситуаціях	25.01.2022	01.02.2022	Виконано
11.	Створення слайдів для захисту та написання доповіді	01.02.2022	15.02.2022	Виконано
12.	Обговорення отриманих результатів з керівником та попередній захист магістерської роботи	28.01.2022	31.01.2022	Виконано
13.	Корегування роботи за результатами попереднього захисту	01.02.2022	15.02.2022	Виконано
14.	Остаточне оформлення пояснювальної записки та слайдів	16.02.2022		Виконано
15.	Захист магістерської роботи	23.02.2022		Виконано

Розробив студент _____Бабін О.С._____ _____

(прізвище та ініціали) (*підпис*)

Керівник роботи __к.пед.н., доц. Болюбаш Н.М. _____ _____

(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 202__р.

АНОТАЦІЯ
до магістерської кваліфікаційної роботи роботи

Тема: «Web-застосунок забезпечення комунікаційної діяльності в ІТ-компанії»

Студент: Бабін Олександр Сергійович

Керівник: к.пед.н доцент Болюбаш Надія Миколаївна

Магістерська кваліфікаційна робота присвячена розробці та здійсненню програмної реалізації Web-застосунку для забезпечення комунікаційної діяльності ІТ-компанії з надійним захистом корпоративної інформації.

Об'єкт дослідження – комунікаційна діяльність в ІТ-компаніях.

Предмет дослідження – програмні засоби для забезпечення безпечної комунікаційної діяльності ІТ-компанії.

Мета дослідження – підвищення ефективності та безпеки комунікації в ІТ-компанії шляхом розробки Web-застосунку з надійною системою шифрування корпоративної інформації й автоматизацією процесів управління персоналом.

Дипломна робота складається з фахового розділу, методичної і спеціальної частини з охорони праці. Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, висновків та додатку.

У першому розділі розкрито теоретичні засади комунікаційної діяльності ІТ-компаній, проаналізувано сучасний стан програмного забезпечення, яке використовується для її підтримки та забезпечення, досліджено підходи до забезпечення надійності захисту інформації в комунікаційних системах.

У другому розділі обґрунтовано вибір технологій і засобів розробки Web-застосунку для забезпечення комунікаційної діяльності. У третьому розділі описано розробку та програмну реалізацію Web-застосунку для забезпечення комунікаційної діяльності ІТ-компанії з надійною системою шифрування інформації.

У спеціальній частині з охорони праці розглядаються питання аналізу умов праці та навчання поведінки при надзвичайних ситуаціях.

Дипломна робота містить ___ сторінку (без додатків), ___ рисунків, ___ таблиці, ___ джерел, ___ додаток.

ABSTRACT

for master's scientific work

Subject: “Web-application for communication activities in IT companies”

Student Babin Oleksandr Serhiiovych
Leader: Ph.D., associate professor Bolyubash Nadiya Mikolaivna

Master's thesis is devoted to the development and implementation of software implementation of Web-application to ensure the communication activities of IT companies with reliable protection of corporate information.

Object of research – communication activities in IT companies.

Subject of research – software to ensure the secure communication activities of the IT company.

The purpose of the study is to increase the efficiency and security of communication in an IT company by developing a Web application with a reliable system of encryption of corporate information and automation of personnel management processes.

Thesis consists of a professional section, methodical and special part on labor protection. The explanatory note of the thesis consists of an introduction, three sections, conclusions and an appendix.

The first section reveals the theoretical foundations of communication activities of IT companies, analyzes the current state of software used to support and provide it, explores approaches to ensuring the reliability of information security in communication systems. The second section substantiates the choice of technologies and tools for developing Web-applications to ensure communication activities. The third section describes the development and software implementation of a Web application to ensure the communication activities of an IT company with a reliable information encryption system.

The special part of labor protection considers the issues of working conditions and training in emergency situations.

Thesis contains ___ page (without appendices), ___ figures, ___ tables, ___ sources, ___

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«WEB-ЗАСТОСУНОК ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЙНОЇ ДІЯЛЬНОСТІ В ІТ-КОМПАНІЇ»

Спеціальність 124 «Системний аналіз»

124 – МКР – 607. 21610301

Студент _____ Бабін О.С.
«__» _____ 20___ р.

Консультант _____ Болюбаш Н.М.
к.пед.наук, доцент
«__» _____ 20___ р.

м. Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UI – інтерфейс користувача (англ. User Interface)

UX – взаємодія користувача з інтерфейсом

MVC – схема розділення даних додатку та управляючої логіки на три компоненти: модель, представлення, контролер (англ. Model-View-Controller)

BLL – рівень бізнес-логіки (англ. Business logic layer)

DAL – шар доступу до даних (англ. Data access layer)

SPA – односторінковий застосунок (англ. Single-page application)

DOM – об’єктна модель документу (англ. Document Object Model)

HR – людські ресурси (співробітники компанії, англ. Human resources)

PP – співробітники, які беруть участь в оцінці персоналу компанії (англ. People partner)

ВСТУП

Актуальність Розбудова інформаційного суспільства в Україні супроводжується інтенсивним впровадженням інформаційних технологій у всі сфери життєдіяльності. Карантин, введений у зв'язку з пандемією COVID-19, прискорив процеси впровадження віддалених форм діяльності працівників різних компаній із використанням сучасних Інтернет-технологій та хмарних сервісів. IT-компанії усе частіше переходять до розробки програмного забезпечення в територіально розподілених командах, що обумовлює необхідність забезпечення ефективних та безпечних віддалених комунікацій для фахівців. Одним із шляхів вдосконалення та підвищення ефективності роботи IT-компаній в умовах територіально розподіленої роботи учасників команди є розробка веб-застосунку забезпечення їх комунікаційної діяльності з надійною системою шифрування внутрішньо корпоративної інформації.

Процес комунікації в IT-компаніях є обміном інформацією між двома або більше учасниками проєктних команд з метою забезпечення розуміння інформації, яка передається у процесі виконання ними функціональних обов'язків та особистих зв'язків усередині команди. Використання Інтернет-технологій для розподіленого у просторі спілкування обумовило різке зростання кількості програмних засобів, які спрямовані на ефективне забезпечення комунікаційної діяльності організацій. До найкращих програм для ефективного командного ділового спілкування сьогодні можна віднести корпоративні месенджери Телеграм, Slack, Flock, Skype, Google Meet, Samepage, хмарний сервіс Highfive, цифрове робоче місце, яке об'єднує розрізнені кадри Beekeeper. Їх використання спрощує спілкування між колегами, дозволяє відправляти голосові та текстові повідомлення, забезпечує відеозв'язок, формує канали, які є робочим простором, дозволяє підтримувати та координувати різні графіки діяльності, що сприяє співробітництву та швидкому прийняттю рішень. Додаток MS Teams збирає в єдиний інтерфейс чати, збори, файли, завдання й календар, що дозволяє легко

спілкуватися, вирішувати загальні завдання, обговорювати проєкт з колегами, планувати й втілювати плани в життя.

Проте питанням забезпечення безпеки корпоративні месенджери приділяють недостаню увагу. Одним із найбезпечніших месенджерів є Телеграм, у якому за безпеку відповідає власний протокол шифрування MTProto, що використовує відразу кілька технологій. Телеграм дозволяє організувати закритий корпоративний зв'язок, однак він недостатньо орієнтований на систему комунікацій ІТ-компаній. Тому доцільним є розробка Web-застосунку забезпечення комунікаційної діяльності в ІТ-компаніях із надійною системою шифрування внутрішньо корпоративної ділової інформації, який підтримує закрите спілкування та обмін діловою інформацією, допомагає усунути вузькі місця у проєкті, забезпечує безперебійну комунікацію з членами команди розробників.

Мета дослідження – підвищення ефективності та безпеки комунікації в ІТ-компанії шляхом розробки Web-застосунку з надійною системою шифрування корпоративної інформації й автоматизацією процесів управління персоналом.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

- розкрити теоретичні засади комунікаційної діяльності ІТ-компаній та сучасний стан програмного забезпечення, яке використовується для її підтримки;
- дослідити підходи до забезпечення надійності захисту корпоративної інформації;
- обґрунтувати вибір технологій і засобів розробки Web-застосунку для забезпечення комунікаційної діяльності;
- розробити та здійснити програмну реалізацію Web-застосунку для забезпечення комунікаційної діяльності в ІТ-компанії з надійною системою шифрування інформації.

Об'єктом дослідження є комунікаційна діяльність в ІТ-компаніях.

Предметом дослідження є програмні засоби для забезпечення безпечної комунікаційної діяльності ІТ-компанії.

Методологічною основою дослідження є загальнонаукові аналітичні методи та методи забезпечення надійності захисту інформації, які дозволили вивчити предмет та об’єкт дослідження, дослідити розвиток науково-методичних засад, напрямів та шляхів підвищення ефективності комунікації співробітників ІТ-компанії за рахунок підвищення безпеки захисту корпоративної інформації та автоматизації процесів управління персоналом.

Наукова новизна одержаних результатів дослідження полягає у тому, що автором: запропоновано та обґрунтовано напрями вдосконалення автоматизації процесів управління персоналом в ІТ-компаніях, спрямовані на забезпечення їх ефективної взаємодії; одержали подальший розвиток підходи до забезпечення безпеки комунікацій; узагальнено теоретичні засади комунікаційної діяльності ІТ-компаній.

Результати дослідження обговорювалися на Всеукраїнській науково-практичній конференції молодих вчених, аспірантів і студентів «Інтелектуальні інформаційні системи» та отримали схвалення.

Практичне значення отриманих результатів полягає в тому, що сформульовані теоретичні положення та практичні рекомендації щодо підвищення ефективності та безпеки комунікації в ІТ-компанії можна застосувати у діяльності ІТ-компаній шляхом впровадження розробленого Web-застосунку.

Структура магістерської роботи. Відповідно до мети, завдань і предмета дослідження, магістерська робота містить основну, методичну та спеціальну частини. Основна частина магістерської роботи складається із вступу, трьох розділів, висновку, списку використаних джерел та __ додатків. Загальний обсяг магістерської роботи – __ сторінок, із них основного тексту основної частини – __ сторінок, методичної частини – __ сторінок, спеціальної – __ сторінок. Кількість використаних джерел – __.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ КОМУНІКАЦІЙНОЇ ДІЯЛЬНОСТІ ІТ-КОМПАНІЙ

1.1. Особиливості комунікацій в ІТ-компаніях

Робоче місце у ІТ-компанії завжди було місцем змін і трансформацій, які було прискорено у зв’язку з пандемією Covid-19. Розробка програмного забезпечення все більше грунтується на віддаленій роботі фахівців територіально розподіленої команди. Пандемія створила багато проблем і труднощів у забезпеченні ефективних розподілених внутрішньокорпоративних комунікацій.

Сучасні організації стикаються з робочим середовищем все більшої складності, база співробітників змінюється, дистанційна робота стає нормою, потреба в інформації стає як ніколи важливою, а цифрове робоче місце стає дедалі складнішим. Щоб процвітати в такій складній бізнес-ситуації, стратегія внутрішніх комунікацій має бути готовою до адаптації. Зробимо огляд еволюції спілкування співробітників у світі, який швидко змінюється, особливо у сфері ІТ. Розглянемо тенденції внутрішньої комунікації, які залишаться в 2021-2022 роках, і як вони впливають на бізнес-процеси.

Звичний нам світ перейшов у режим home-office, ІТ-компанії перевели своїх співробітників на віддалену роботу, зіткнувшись не лише з технічними проблемами. Віддалений режим роботи – нова реальність. За даними ESET, одним із трендів стають нові виклики безпеки для компаній, спровоковані віддаленим режимом роботи у зв'язку з пандемією коронавірусу. Дослідження ринку та компаній показує, що все більше програмістів вибирають віддалений режим роботи. Така тенденція увійде у звичний режим після хвилі Covid-19.

Тому постають питання, які потребують першочергового вирішення. Як у такому режимі підтримати працездатність працівників? Як забезпечити не просто внутрішню комунікацію у рамках проекту, але й у різних кластерах по департаментах, офісах або глобально? Як об'єднати автоматизацію департаменту

HR та BP аналіз працівників. Як створити рішення щодо підбору проектів для нових задач, моніторингу, створення івентів для розвитку Hard й Soft скіллів працівників та заохотити їх?

У світі, який динамічно змінюється, на новий рівень виходять багато процесів, які забезпечують управління персоналом, його оцінку та забезпечення внутрішньокорпоративної комунікації. Сучасні кадровики (PP – співробітники) є HR-фахівцями (HR – людські ресурси, англ. Human resources) й займаються розвитком корпоративної культури, оцінкою ефефктивності співробітників, визначенням вимог до майбутніх співробітників, веденням бази даних співробітників та кандидатів, організацією опитувань, проведенням зустрічей 1:1 і визначенням продуктивності роботи, оцінки співробітників (англ. Performance review).

Одним з видів сервісів колективної роботи користувачів є соціальні мережі. Соціальна мережа – це сукупність соціальних елементів (людей, груп, організацій, спільнот), які обмінюються між собою різними даними. Також, необхідно надати функціонал, що підходить під комунікацію у IT-компаніях, щоб HR та PP спеціалісти змогли легко створювати зустрічі, отримувати актуальну інформацію щодо фахівців, полегшити процес підбору проекту та комунікації всередині окремих кластерів, отримувати фідбек, тощо.

У той час як колись було цілком нормально запускати електронний лист і чекати, поки співробітники всієї компанії прочитають його протягом тижня, це більше не вихід. Зараз це одна з найважливіших передових практик внутрішньої комунікації, яка забезпечує миттєву доставку повідомлень. Директиви, політика та новини, якими потрібно поділитися з персоналом, тепер постійно змінюються. Правило, яке застосовувалося в понеділок, можливо, доведеться змінити у вівторок вранці. Коли справа доходить до повідомлення про зміни, затримка не просто небажана, вона також може бути небезпечною. Якщо у передачі важливої інформації спостерігається затримка, люди продовжуватимуть діяти таким чином, що може загрожувати їхньому здоров’ю та безпеці.

Однією з найочевидніших тенденцій внутрішніх комунікацій, що випливають на поверхню, є раптова обізнаність передового працівника. Ці ключові працівники, яких раніше не помічали, тепер є першими у списку пріоритетів. Тепер, коли керівники перевіряють спілкування зі співробітниками, їх турбує наявність таких працівників на передовій. Хто ці працівники? Це люди, які змушують речі відбуватися. Стає все більш очевидним, що працівники на передовій є одними з найважливіших співробітників в організації.

Зробимо менеджментський огляд з боку HR та PP. Для огляду необхідно спочатку зрозуміти основні функції, які HR та PP департаменти виконують в IT-компаніях.

Перша функція - це рекрутмент (recruitment), тобто пошук та інтерв'ювання кандидатів, наймання та адаптація персоналу. Рекрутер повинен оцінювати як професійний досвід, а й менеджерські навички, і особистісні характеристики, повинен розбиратися у бізнесі.

Друга функція – оцінка персоналу (performance). Важливо враховувати Soft та Hard скілли фахівців та стежити за тим, щоб компанія проводила активні тренінги для розвитку та вивчення нових технологій для співробітників. Також важливо розуміти, що конкретний проект може мати досить не тривіальні вимоги до кандидата, тому можна заохочувати менторів для розвитку та оцінки потенціального кадру.

Третя функція – навчання та розвиток персоналу (training & development). Існує багато різних систем: корпоративний університет, зовнішні тренінги, внутрішня система навчання. Навчання часто розглядається як розкіш або якийсь бонус за добре виконану роботу. Тому навчання постійно є першим пунктом, який скорочується, коли бюджети компаній знаходяться під тиском. Наразі, однак, має бути зрозуміло, що гроші втрачаються, якщо працівники не мають належних навичок чи знань, щоб брати участь у певних проектах чи закінчити їх – не кажучи вже про те, як це впливає на задоволеність роботою та моральний дух. Навчання всередині компаній слід розглядати як інвестицію, а не як розкіш.

Незважаючи на це, ступінь зміни продуктивності співробітника в результаті навчання не завжди відповідає очікуванням керівництва, незважаючи на значні кошти на навчання. Це питання має на меті визначити важливість навчання: наскільки важлива IT-освіта для співробітників, чи була можливість пройти навчання фактором, коли вони вирішили претендувати на свою поточну роботу, і як IT-спеціалісти розглядають навчання на робочому місці – що впливає це впливає на компанію в довгостроковій перспективі та загальний статус співробітників на ринку праці.

Четверта функція – система зарплат, бонусів та компенсацій (compensation&benefits). Компенсація – це винагорода, яка виплачується працівнику за роботу, яку вони виконують для організації. Коли працівник працює в організації, йому повертають гроші, пільги, такі як безкоштовне харчування, необмежений час відпустки, чудове медичне обслуговування, бонуси тощо за їхній час і талант. Все це є частиною винагороди працівникам, яка може мати фінансовий або нефінансовий характер.

З іншого боку, переваги (benefits) відносяться до нефінансової винагороди, яка надається працівнику на знак вдячності за його роботу. Серед переваг, які користуються популярністю серед співробітників, є гнучкий графік роботи, можливість працювати віддалено, доступ до абонементу в спортзал і харчування.

П'ята функція – корпоративна культура (corporate culture & internal communication). До корпоративної культури відносяться такі речі, як місія та візія компанії, соціальна відповідальність. Грубо кажучи, компанія має відповідати через усі ці тези на ключове питання: чим вона відрізняється від інших на ринку, яку цінність несе суспільству та країні. Цю інформацію треба доносити і до працівників — вони є носіями корпоративної культури. Кожна дія в рамках корпоративної культури та соціальної відповідальності має співпадати з маркетинговою стратегією. Інакше це марна трата грошей, яка не підтримує імідж компанії. Якщо корпоративна культура компанії нічим не відрізняється на фоні

конкурентів, її співробітники не розцінюватимуть роботу як щось важливе. Вони підуть до тих, хто більше заплатить чи видасть найкращі умови.

Шоста функція – взаємодія із гендиректором або "коучинг керівника" або стратегічний HR. Гендиректор теж людина, він може банально втомитися і йому потрібен партнер для управління командою. Сигналізувати про проблеми та запропонувати способи їх вирішення – це пряме завдання HR високого порядку.

Зроблений огляд дозволив визначити ключові функції, які потрібно автоматизувати у Web-застосунку для підтримки комунікації у IT-компанії. Це, перш за все, кадровий облік: автоматизація всіх процесів роботи з персоналом – від кадрового адміністрування до завдань планування і управління. Рішення можна використовувати для врахування кадрової діяльності кількох організацій будь-якої структури.

Ефективність платформи, яка буде обрана для підтримки комунікацій, безпосередньо залежить від її використання командою IT-фахівців. Якщо працювати з нею складно, швидше за все, переваг, на які розраховували, отримано не буде. Правильно підібрана платформа буде захоплюючою, простою у використанні, і при цьому зможе органічно забезпечити потреби команди у внутрішньому спілкуванні, спільному використанні та зберіганні документів, управлінні завданнями та організації спільної роботи. Чим рідше співробітникам доводиться перемикати контекст, то краще. Команді буде простіше зосередитися на деталях і весь процес буде націлений на підвищення продуктивності.

Звичайні мережі не можуть покрити сторони автоматизації HR та PP функцій, та не мають усіх функцій, необхідних бізнесу у IT-сфері. Такі напрацювання зазвичай існують як внутрішні системи, до яких можна отримати доступ лише працівнику, та зазвичай досить примітивні у функціях та виборі технологій для реалізації. Тому компанії часто використовують декілька рішень, купуючи окремі пакети для комунікації, менеджменту, роботи HR департаменту, тощо. Єдиного рішення, яке змогло покрити всі потреби аутсортингової розробки програмного забезпечення компаній та мало необхідний функціонал для

автоматизації управління й оцінки персмоналу з надійним захистом корпоративної інформації, немає. Тому є необхідність у використання такого застосунку для комерційних цілей, організувавши деплоймент на боці серверів IT-компанії, яка буде використовувати мережу.

1.2. Засоби забезпечення комунікаційної діяльності компаній

Web-застосунок для забезпечення комунікації з точки зору інтернету - це web-ресурс, який базується на об'єднанні людей зі спільними інтересами або діяльністю для обміну інформацією один з другом. З розвитком концепції Web 2.0, що передбачає, що більшу частину змісту web-сайтів створюють користувачі цих сайтів, а не їх web-майстер, розвиток мереж комунікації отримало величезний поштовх.

Сьогодні користувачами таких мереж є мільйонів людей по всьому світу, витрачаючи терабайти інтернет-трафіку щодня для відвідування сотень відповідних web-ресурсів. Багато комунікаційних мереж являють собою багатомільйонні бізнеси, які заробляють в основному на рекламі. А обсяг контенту у вигляді музичних і відео-файлів, завантажених користувачами в архів комунікаційних мереж, становить конкуренцію навіть спеціалізованим ресурсів і є причиною приєднання ще більшої кількості учасників.

Більшість комунікаційних мереж на сьогодні є набором особистих сторінок їх учасників, що містять інформацію про користувача і його інтересах, один або кілька альбомів з фотографіями, розміщених учасником, особистий блог, званий в російськомовних мережах «стіною», посилання на відео-і музичні файли, завантажені користувачем і т. п. Користувачі мають можливість додавати один одного в контакти і обмінюватися один з одним особистими повідомленнями і записами на «стінах».

Ідея використання комп'ютерних мереж для організації соціальної взаємодії не нова. Далекими предками комунікаційних мереж можна вважати сервіси

Usenet і ARPANET, LISTSERV, ну і звичайно ж, BBS і мережу Fido. Ера Web дала новий поштовх розвитку комунікаційних мереж.

Ідея гіперпосилань, а також мультимедійні можливості виявилися справжньою знахідкою для розвитку подібних ресурсів. Першими соціальними проектами у всесвітній павутині вважаються theGlobe.com (Заснований в 1994 р), Geocities (1994 г.) і Tripod (1995 г.). Ці ресурси були онлайн-спільноти, що надають людям можливість спілкування один з одним в чаткомнатах, а також дозволяють користувачам розміщувати записи на особистій сторінці і обговорювати їх.

Остання можливість була прабатьком більш сучасного феномена під назвою «блогінг». TheGlobe.com виявився найбільш комерційно успішним серед них проектом, проіснувавши до 2007 р .; в кращі свої роки він набирав 800 мільйонів доларів в ринковій капіталізації.

Інші соціальні мережі об'єднували людей в мережу за допомогою email-адрес. серед таких сайтів можна виділити Classmates.com (1995 г.), спрямований на підтримку зв'язків з колишніми однокласниками і SixDegrees.com (1997 г.). Останній сайт дозволяв Створення списку особистих профілів, посилати повідомлення користувачам зі списку «друзів», була присутня можливість подивитися список людей, які мають спільні з учасником інтереси, - в загальному був родоначальником мереж комунікації в сучасному її розумінні. На жаль, цей ресурс виявився невигідним і в 2001-му був змушений закритись. В даний час, мабуть, кожна сучасна людина, що користується Інтернетом, має свою сторінку в комунікаційних мережах, таких як «Facebook», «Telegram». «LinkedIn», «Мійсвіт», «Behance»,«Instagram» і т.п..

Справжній ривок у розвитку web-сервісів комунікаційних мереж стався між 2002 та 2005 роками. Саме в цей час з'явилася перша сучасна соціальна мережа Friendster (2002 г.), а потім і найбільші на сьогоднішній момент соціальні мережі - MySpace (2003 рік) і Facebook (2004) . У 2005 р MySpace обігнав за кількістю переглянутих сторінок компанію Google, а Facebook зараз є найбільшим web-

сайтом комунікаційних мереж в світі, зібравши 132.1 мільйона унікальних відвідувань за один лише червень 2008 р. Дана сфера є дуже перспективною в цілому, оскільки проблеми з обмеженням свобод все більше зростають с часом. Завдяки стабільно високим темпам розвитку сфери дана індустрія має високу інвестиційну привабливість, обсяг сфери з кожним роком збільшується.

Можливо, ви чули про популярних ESN, таких як Yammer, Teams, Jive, Chatter та Jabber. Але як корпоративні соціальні мережі зайняли таке важливе становище і чому?

Ще на початку 1990-х років, коли Інтернет тільки-но з'явився, люди почали використовувати глобальну мережу для об'єднання та формування спільнот за інтересами. Такі сайти, як theGlobe.com та Geocities (закрився в березні 2019 року після майже 25 років існування), стали універсальними майданчиками для обміну контентом у мережі. Ці мережі виросли настільки, що всі прагнули туди потрапити, включаючи рекламодавців.

На прикладі таких ранніх сайтів виросли сучасні платформи комунікаційних мереж, які використовують багато хто з нас: Facebook, Twitter та LinkedIn. Взявши ряд ідей у цих проектів з величезною популярністю на споживчому ринку, почали з'являтися та набирати обертів соціальні мережі для компаній, такі як Yammer та Jabber.

Yammer був створений як соціальна мережа за аналогією з Facebook, але для внутрішнього корпоративного використання. Ви можете спілкуватися з будь-яким з інших співробітників через інтуїтивно зрозумілий інтерфейс платформи, а стрічка активності показує останні дії та контент ваших колег. Компанії оцінили можливість використання єдиної платформи для спілкування та спільної роботи, оскільки це допомагає кожному співробітнику залишатися в курсі того, що відбувається. Microsoft придбала Yammer у 2012 році та використала цю модель для створення продукту Microsoft Teams.

Комунікаційний огляд. Більшість із продуктів web-сервісів та комунікаційних мереж зараз використовується для комунікації у IT компаніях

різного масштабу по кількості працівників та різних по сферах розробки. Маючи досвід роботи в великих компаніях, таких як Globelink, GlobalLogic та Ciklum, я можу зазначити, що, наприклад, Microsoft Teams та Google сервіси є досить популярним рішенням для комунікації: проведенню планінгів, мітингів, власним плануванням часу у календарі та використанням групп для обміну останніх апдейтів на PR’s.

Для документування функціоналу програмного забезпечення, основних архітектурних рішень, роботи мікро-сервісів та мікро-фронтенду часто використовую Confluence - що дозволяє організовувати робочий простір.

Платформа багатьох ESN на додаток до функції чату включає інструменти спільного доступу до документів та управління завданнями. Але деякі компанії можуть усвідомити, що переваги корпоративної соціальної мережі доступні їм у ПЗ для внутрішньої організації робочих просторів, яке вони вже використовують і в якому є можливості управління проектами, наприклад Confluence.

У початковому сенсі Confluence — це робоче місце, у якому організації можуть створювати і впорядковувати базу знань, і навіть обмінюватися командах знаннями і проектної документацією. Однак, крім цього, Confluence забезпечує можливість двостороннього спілкування та спільної роботи над контентом. А за допомогою стрічки популярних матеріалів (інструменту для перегляду цікавого контенту, опублікованого по всій компанії), команди можуть залишатися в курсі останніх оголошень, зроблених колегами.

Наприклад, міжнародна некомерційна організація Mercy Ships використовує Confluence як корпоративну соціальну мережу, закликаючи учасників команд ділитися особистими та робочими успіхами з усією компанією. Коли ці історії публікуються, як правило, у формі записів у внутрішньому блозі, інші учасники корпоративних команд можуть підтримати автора коментарем або поставити свої запитання. Все це дійсно сприяє зближенню колективу та формуванню корпоративної спільноти.

Перед впровадженням ESN подумайте над рядом питань: наприклад, хто у вашій організації відповідатиме за обслуговування цієї мережі. Деякі компанії вирішують централізовано віддати володіння мережею та завдання модерації в руки однієї команди, наприклад, IT або HR; інші ж досягають успіху, розподіляючи ці права та завдання по всіх командах. У ситуації з Mercy Ships все, що потрібно, – це доручити IT-відділу налаштувати внутрішній екземпляр Confluence і встановити деякі робочі принципи щодо використання мережі, етикету спілкування та відповідних рекомендацій, яким міг би наслідувати кожен співробітник компанії.

Зрештою, ефективність платформи, яку ви оберете, безпосередньо залежить від її використання вашою командою. Якщо працювати з нею складно, швидше за все, ви не отримаєте переваг, на які розраховували. Правильно підібрана платформа буде захоплюючою, простою у використанні, і при цьому зможе органічно забезпечити потреби команди у внутрішньому спілкуванні, спільному використанні та зберіганні документів, управлінні завданнями та організації спільної роботи. Що рідше вашим співробітникам доводиться перемикати контекст, то краще. Команді буде простіше зосередитися на подробицях і весь процес не буде підривати продуктивність.

Звичайні мережі не можуть покрити сторони автоматизації HR та PP функцій, та не мають всіх функцій необхідних бізнесу. Такі напрацювання зазвичай існують як внутрішні системи, до яких можна отримати доступ лише працівнику, та зазвичай досить примітивні у функціях та вибору технологій для реалізації. Тому компанії часто використовують декілька рішень, купуючи окремі пакети для комунікації, менеджменту, роботи HR департаменту, тощо. Єдиного рішення, яке змогло покрити всі потреби більшості IT аутсорс компаній та підходила та мала необхідний функціонал, немає. Тому, використовувати додаток можно для комерційних цілей, організувавши деплоймент на боці серверів IT-компанії, яка буде використовувати мережу.

Зробимо аналіз аналогів та виявимо недоліки існуючих рішень.

Microsoft Teams (також Slack та Flock входять до месенджерів із схожим функціоналом та загальними недоліками) – корпоративна платформа (рис. 1.1), що об'єднує в робочому просторі чат, зустрічі, нотатки та вкладення. Розроблена компанією Microsoft як конкурент найпопулярнішого корпоративного рішення Slack. Сервіс представлений у листопаді 2016 року, одночасно стала доступною попередня версія.

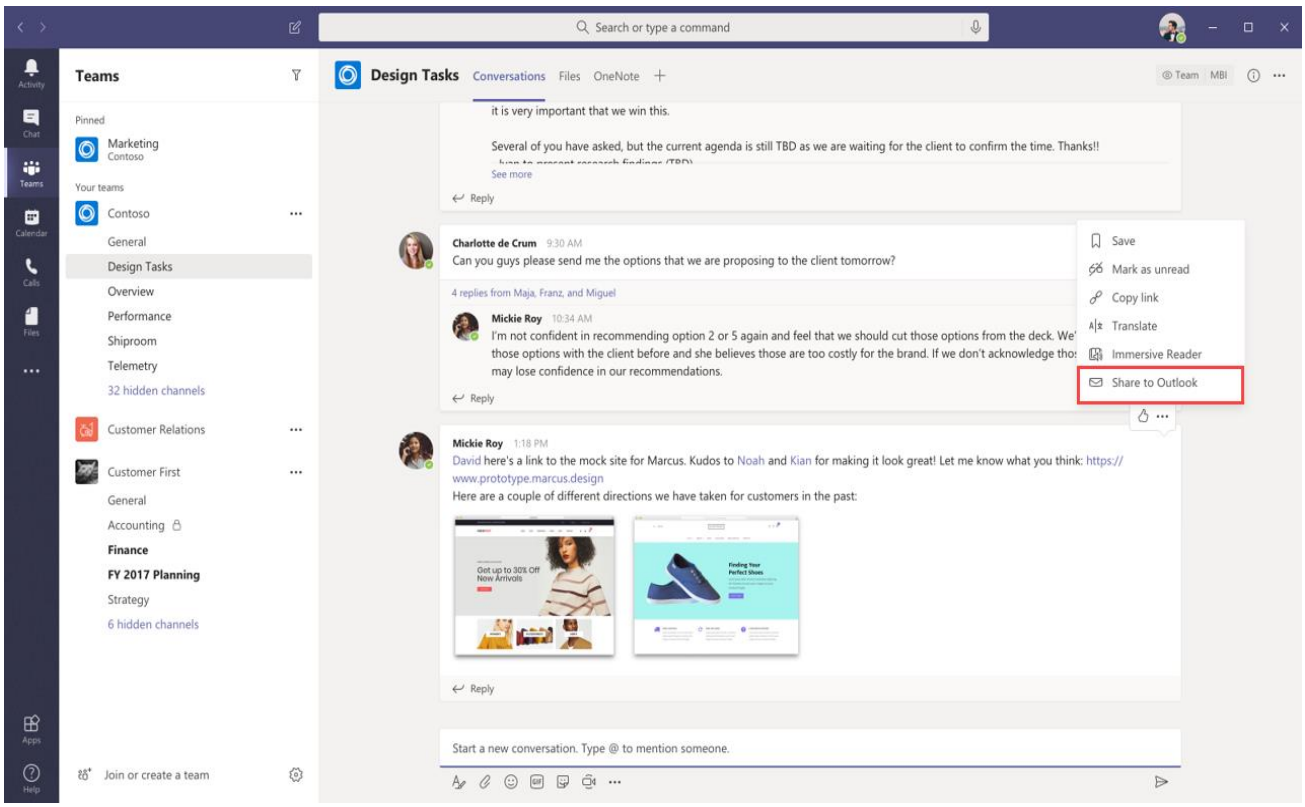

Рис. 1.1. Інтерфейс платформи Microsoft Teams

Можливості Microsoft Teams:

1. Chat — дозволяє надіслати особисті або групі повідомлення по роботі, проекту.

2. Teams — надає можливість для створення команд та каналів, щоб зібрати людей разом та працювати у просторах із бесідами та файлами.

3. Календар — забезпечує з’єднання з іншими людьми до, під час і після зборів. Цей Teams синхронізується з Outlook календарем.

4. Програми: пошук знайомих програм та знайомство з новими програмами для спрощення роботи, налаштування та керування ними.

Недоліки: немає інструментів менеджменту працівників, не підходить для збору необхідних даних для HR та PP спеціалістів, не підходить для управління та комунікацій у великих компаніях, адже є рішенням більш локальним під окремий проект; платформа працює не завжди коректно, існують баги та проблеми при завантаженні сторонніх файлів та документів; немає новостної стрічки, щоб бути у курсі останніх подій в компанії, тренінгах та вебінарах (функціонал “канали” складно використовувати для цих цілей, адже при великій кількості постів знайти щось конкретно складно у Teams, та не покриває весь описаний функціонал), немає можливості реагувати та запрошувати працівників на конкретний івент.

Skype (та інші аналоги, такі як Google Meets, WhatsApp, Telegram) - месенджери (рис. 1.2) з підтримкою відеодзвінкців, не виконують ніяких функцій, окрім коммунікаційних, між двома або більше людьми. Таке рішення не підходить під описану задачу, адже не автоматизує роботу HR та PP, та не є внутрішнім рішенням під соціальну мережу всередині IT-компанії для заохочення працівників до розвитку та комунікацій. Месенджери не є комплексним рішенням для бізнесу, адже виконують більше функцію особистого спілкування. Не дозволяють розділяти робочі та власні контакти.

Samepage (також Beekeeper) - сервіс, що полегшує спілкування, керування проектами, проведення зустрічей, онлайн-спільну роботу та багато іншого, поєднуючи командний чат, відеоконференції, спільний доступ до екрана, керування завданнями, спільний доступ до файлів та спільну роботу з командними документами в режимі реального часу в єдиному хмарному робочому просторі для спільної роботи. Основні недоліки:

1. Важко отримати справжню картину всіх проектів і сторінок, які відбуваються одночасно.

2. Налаштувати членів команди на роботу може бути досить складним, є певна крива навчання, і це не дуже інтуїтивно зрозуміло.

3. Немає інформації щодо захисту інформації та алгоритмів, які для цього використовуються.

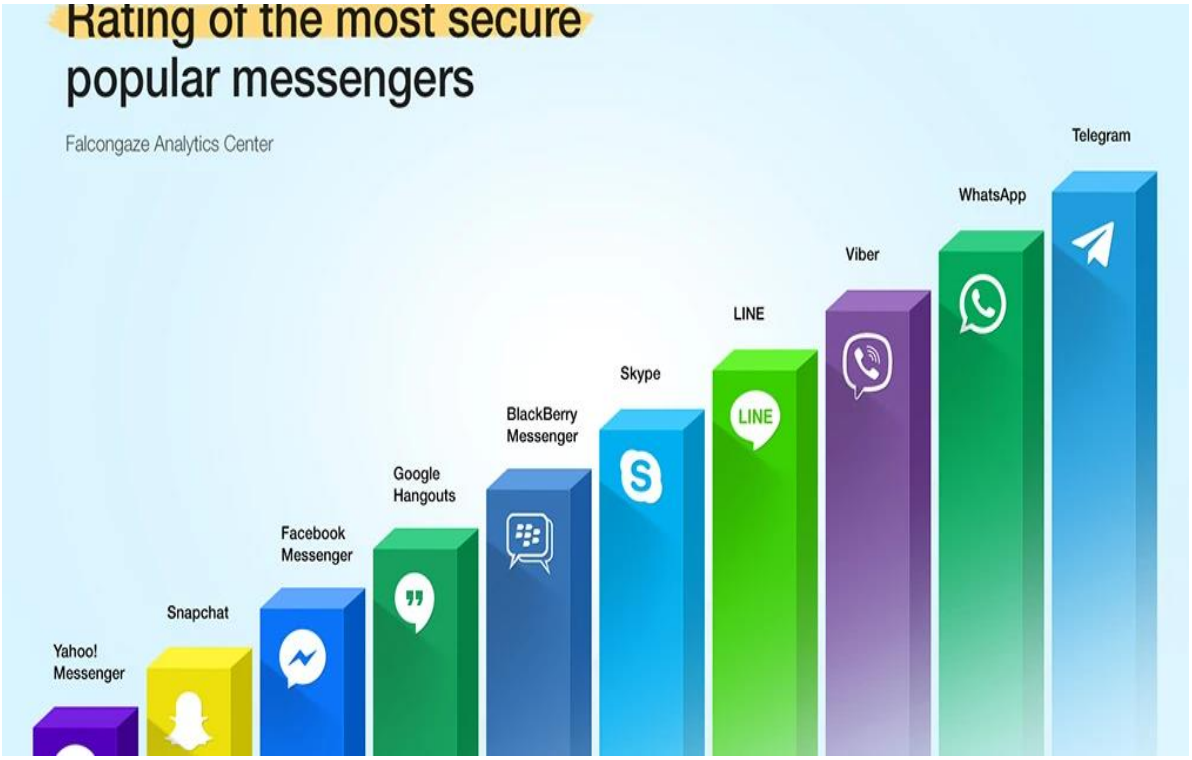

Рис. 1.2. Актуальні мессенджери по рівню безпеки

Працювати HR у рамках всієї компанії для забезпечення автоматизації управління персоналом не вийде - для аутсорс компаній використовують клауд платформу за реквестом кастомера, та використовується під конкретний проект. Для менеджмента повного списку працівників та аналізу команд не підходить.

1.3. Методи забезпечення надійності захисту даних

Для забезпечення надійності захисту використовують часткове або повне шифрування повідомлень та файлів, самознищення повідомлень і цілих чатів,

блокування можливостей для скріншотів. Різко знижує витік корпоративної інформації застосування багатофакторної автентифікації на шифрування.

У світі IT, де кількість випадків кіберзлочинів зростає, Важливо визначитися з методами, які найкраще підходить для захиститу мережевої безпеки організації.

Шифрування — це процес перетворення відкритого тексту (незашифрованих даних) у зашифрований текст (зашифровані дані), щоб лише люди, які мають секретний ключ (формально відомий як ключ дешифрування), мали доступ до цих зашифрованих даних і могли декодувати інформацію. Існує кілька різних методів шифрування.

Шифрування є процесом перетворення даних із формату для читання в зашифрований формат, який можуть прочитати лише авторизовані користувачі, які можуть перетворити закодовані дані назад у вихідні дані та отримати доступ до вихідних даних. Процес перетворення вихідного повідомлення в закодований формат відправником відомий як шифрування, а процес перетворення закодованого повідомлення назад у вихідний формат відомий як дешифрування. Оскільки ризик загроз кібербезпеці зростає з кожним днем, важливо знати основну техніку шифрування користувачів, які використовують Інтернет.

Як працює шифрування?

Наскрізне шифрування (англ. end-to-end encryption), при якому доступ до інформації мають тільки ті користувачі, які приймають участь у спілкуванні, не дозволяє отримати доступ до криптографічних ключів третім особам, є найбільш надійним способом захисту даних. Для обміну ключами при цьому можуть бути застосовані симетричний та асиметричний алгоритми.

Симетричне шифрування використовує єдиний ключ, який використовується обома сторонами, що спілкуються, для шифрування та дешифрування. Асиметричне шифрування використовує два ключі, один з яких є приватним, а другий – відкритим. Відкритий ключ є загальним ключем і використовується як частина протоколу шифрування, тоді як закритий ключ є

лише на пристрої і ніколи не надається іншим людям. Комбінація відкритого та закритого ключів використовується для створення тимчасового спільного ключа під час надсилання захищеного тексту.

Обмін інформацією з використанням шифрування відбувається наступним чином. Користувач А надсилає повідомлення користувачу В, яке в той же час викликає обмін відкритими ключами. І користувач А, і користувач Б використовують відкритий ключ іншої особи разом зі своїм закритим ключем для створення тимчасового спільного ключа. Потім вони використовують спільний ключ для кодування своїх повідомлень, а їхні відкриті ключі використовуються для перевірки автентичності цих спільних ключів.

Спільні ключі безперервно видаляються та генеруються, що гарантує, що чат пана Б неможливо розшифрувати в майбутньому. Зловмисник міг побачити, що користувач А і користувач Б обмінялися повідомленнями, але оскільки зловмисник не має спільного ключа пана А і містера Б, ці тексти не можуть бути розшифровані зловмисником.

Основні задачі, які дозволяж вирішувати шифрування даних:

1. Аутентифікація. У сучасному світі, де в Інтернеті працює так багато шахрайських веб-сайтів, автентифікація є важливою функцією для захисту конфіденційних даних. Ця функція гарантує, що тільки призначений користувач отримає доступ до спільної інформації.

2. Безпека. Шифрування – це метод, за допомогою якого ви можете встановити цифровий замок на електронний запис таким чином, щоб до встановленого електронного запису не міг отримати доступ ніхто, крім призначеного користувача. Він підвищує безпеку повідомлень або файлів, скремблюючи вміст, що робить його нечитаним ніким, крім тих, у кого є ключі для його декодування.

3. Конфіденційність – шифрування забезпечує конфіденційність персональних даних. Це гарантує, що ніхто не зможе отримати доступ до наших

даних, крім призначеного одержувача. Це запобігає доступу хакерів, зловмисників, зловмисників до наших конфіденційних даних.

4. Законодавство – у багатьох країнах існують певні законодавчі та нормативні вимоги до організації, яка займається особистою інформацією користувачів, щодо збереження даних користувача в зашифрованому вигляді. Основною метою шифрування даних, що зберігаються на нашому комп'ютері або пристроях, є забезпечення конфіденційності та захист наших даних та інтелектуальної власності.

Основні методи шифрування даних можна розділити на три типи:

1. Методи симетричного шифрування – це також називається криптографією з приватним ключем. У цьому методі і відправник, і одержувач мають доступ до одного ключа, а одержувач використовує той самий ключ для розшифровки повідомлення, надісланого відправником. Отже, одержувач повинен мати ключ до того, як повідомлення буде розшифровано.
2. Методи асиметричного шифрування – також відомі як криптографія з відкритим ключем. У цьому методі використовуються два ключі, тобто відкритий ключ і закритий ключ. Відкритий ключ, як випливає з назви, є загальнодоступним. Найцікавішою частиною цієї нової системи є те, що текст може бути закодований відкритим ключем, але той самий ключ не може використовуватися для декодування тексту. Для його декодування необхідно використовувати відповідний закритий ключ. Жодні закриті чи відкриті ключі інших користувачів не зможуть декодувати текст. Завдяки цій системі передача даних дуже безпечна.
3. Хешування – це процес перетворення введених даних будь-якої довжини в рядок тексту фіксованого розміру за допомогою математичної функції. Хешування — це метод зберігання та отримання даних із хеш-таблиць. Хешування можна використовувати для перевірки документа.

Виявлено, що найнебезпечнішими алгоритмами серед найбільш часто використовуваних алгоритмів шифрування є AES, Triple DES, Blowfish, Twofish, RSA.

1. AES – розшифровується як «Advanced Encryption System», є найбільш популярними та надійними алгоритмами симетричного шифрування, які були розроблені для заміни алгоритму DES. Це швидше в порівнянні з алгоритмом DES. Він використовується в різних програмах, таких як бездротовий захист, безпека Wi-Fi, VPN, протокол SSL тощо. Багато державних організацій довіряють алгоритму шифрування AES для захисту своїх конфіденційних даних.

2. TDES – означає потрійний алгоритм шифрування даних і є одним із типів симетричного алгоритму шифрування. Алгоритм потрійного шифрування даних піддається силі часу, як і будь-який інший алгоритм шифрування. Це оновлена версія алгоритму шифрування DES. TDES є частиною таких криптографічних протоколів, як SSH, OpenVPN, TLS. Він вміщує алгоритм DES тричі до кожного блоку даних і є широко використовуваним алгоритмом шифрування в платіжних системах, технологіях у фінансовому секторі, яка використовується для шифрування PIN-коду банкоматів, паролів UNIX.

3. RSA – це асиметричний алгоритм шифрування для шифрування даних, що передаються через Інтернет. Потенційному хакеру потрібно багато часу та енергії, щоб зламати системи, тому що це створює величезну купу балаканини, яка може їх засмутити. Таким чином, це сильний і надійний алгоритм шифрування, який можна використовувати для захисту мережевої безпеки організації.

4. Blowfish – це ще один алгоритм симетричного шифрування, розроблений Брюсом Шнайєром як альтернатива DES. Він добре відомий своєю швидкістю та гнучкістю. Його може використовувати будь-хто, оскільки цей алгоритм є загальнодоступним. Це шифр з розміром ключа 32-448 біт і розміром блоку 64 біт. Blowfish широко використовується на платформах електронної комерції та інструментах керування паролями.

5. Twofish — ще один симетричний блочний шифр, який був розроблений як наступник Blowfish. Ключовою особливістю Twofish є те, що він використовує попередньо порівняні ключі S-блоки та складний розклад клавіш. Дві риби також не запатентовані і розміщені у суспільному надбанні для використання будь-ким. Він вважається найшвидшим і використовується як в апаратному, так і в програмному середовищі. Цей метод зазвичай використовується в багатьох програмних рішеннях для шифрування файлів і папок.

6. RSA (Рівест Шамір Адельман) – це асиметричний криптографічний алгоритм. Він спирається на розкладання двох великих простих чисел на множники. Ще одне велике число створюється за допомогою цих двох випадкових простих чисел. Повідомлення може бути розшифровано лише одержувачем, який знає ці два прості числа. Хакеру було б важко визначити початкове просте число з цього великого помноженого числа. Але під час шифрування великого обсягу даних алгоритм може уповільнити. RSA зазвичай використовується в різних програмах, таких як сертифікати SSL/TLS, криптовалюти, цифрові підписи та шифрування електронної пошти.

Алгоритм шифрування ECC – розшифровується як криптографія з еліптичною кривою. Це досить складний метод, оскільки він використовує криву, яка є схематичним зображенням точок, які задовольняють математичне рівняння. Він використовує коротші клавіші, які вимагають меншого навантаження на мережу. Він забезпечує такий же рівень захисту, як і RSA, але забезпечує більшу безпеку та швидшу продуктивність.

Неналежно захищені API є основною мішенню для зловмисників, тому організації звертають увагу на безпеку програм. У недавньому опитуванні, проведеному Forrester, 28 відсотків тих, хто приймає рішення щодо безпеки, повідомили, що їхнім головним тактичним пріоритетом є покращення можливостей безпеки програм.

Не випадково безпека є головною проблемою для організацій, які прискорили темпи розробки додатків. Оскільки вони намагаються винайти нові способи взаємодії зі своїми клієнтами через свої програми, недоліки безпеки неминучі. І якщо їм не вистачає надійних методів безпеки для пом'якшення помилок розробників, вони не можуть захиститися від все більш витончених зловмисників і атак.

1.4. Оптимізований алгоритм шифрування RSA

RSA (Rivest, Shamir та Adleman) — криптографічний алгоритм з відкритим ключем, придатний і для шифрування, і для цифрового підпису. Математичний апарат, що лежить в основі роботи алгоритму дуже складний і базується на обчислювальній складності задачі факторизації великих цілих чисел. Існує багато дослідження з приводу модифікацій алгоритму та його удосконалення. Розглянемо китайську теорему про залишку та її вплив при використанні у даній криптосистемі.

Сутність китайської теореми про залишки є наступною. Для будь-якої пари простих натуральних чисел n_1 і n_2 можна знайти таке ціле число x, що $x = x_1 \bmod n_1$ і $x = x_2 \bmod n_2$.

Криптографічні системи з відкритим ключем використовують односторонні функції, які дозволяють відносно легко розрахувати $f(x)$, якщо відомо x та гарантують практичну неможливість з використанням сучасних обчислювальних засобів за прийнятний час розрахувати x, якщо відомо $y = f(x)$. Криптосистема базується на задачі факторизації добутку двох великих чисел, коли для шифрування необхідно реалізувати піднесення у степінь числа, великого по модулю. В основі дешифрування лежить обчислення функції Ейлера від великого числа, що вимагає розкладу числа на прості множники.

Перевагою алгоритму симетричної криптографії RSA є час процедури шифрування та дешифрування, який є помірно швидким через ефект генератора ключів. Оскільки процедура, як правило, швидка, цей розрахунок є розумним для розширеної системи кореспонденції, яка прогресує як GSM. Тим не менш, надсилання повідомлень різним клієнтам має несподіваний ключ у порівнянні з минулим. Таким чином, кількість уловів буде законно відповідати кількості клієнтів. Використання спільного ключа працює для шифрування, але проблема в тому, що обидві сторони повинні домовитися про ключ. Це проблематично в реальному світі, оскільки непрактично чи безпечно ділитися через мережу. Рішення полягає у використанні такого алгоритму, як RSA, який генерує пару ключів, що містить відкритий та закритий ключ. Як вказують на їх назви, приватний ключ повинен зберігатися в секреті, тоді як відкритим ключем можна вільно ділитися (рис. 1.3).

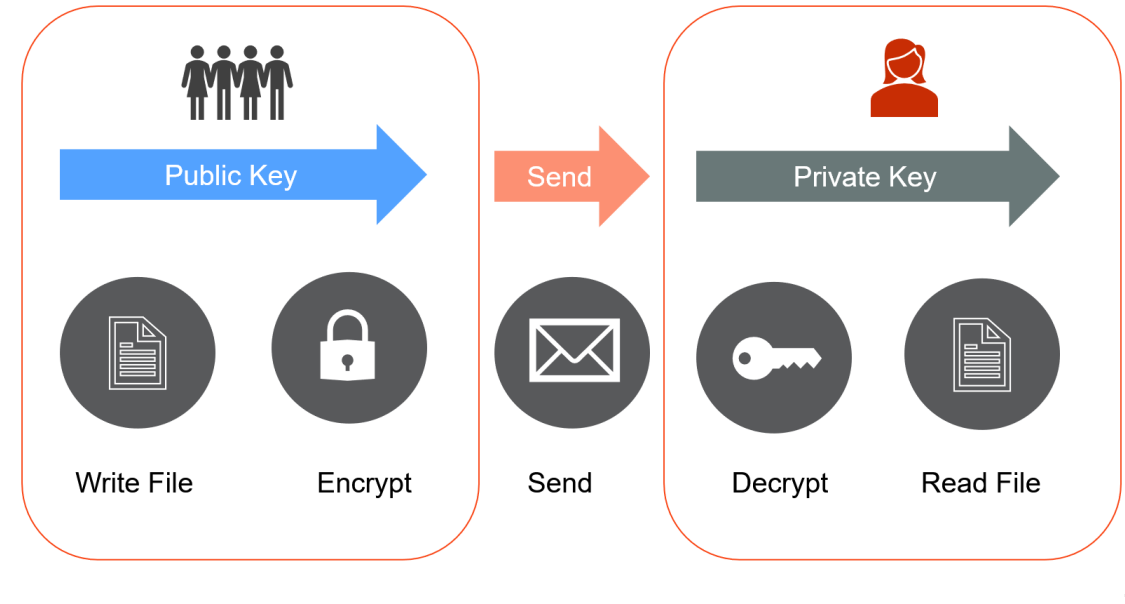

Рис. 1.3. Шифрування за допомогою відкритого та закритого ключів

Користувачі мають відкритий та приватний ключ, який тримається у секреті. Кожен із ключів складається з пари цілих чисел. Генерація ключів здійснюється наступним чином:

1) обирають два великих простих числа *p* та *q* заданого розміру (512, 1024 чи 2048 біт) та розраховують модуль – добуток цих чисел: *n = p*q*;

2) обчислюють значення функції Ейлера від модуля *n*: *f(n) = (p – 1)*(q – 1)* та обирають ціле взаємно просте число *e* – відкриту експоненту, яке буде знаходитися на проміжку від 1 до *f(n)*: $1 < e < f(n)$;

3) обчислюють з використанням алгоритму Евкліда секретну експоненту – число *d*, мультипликативно зворотне відкритій експоненті *e* по модулю *f(n)*: $d \cdot e = 1 (\mathrm{mod}\ f(n))$ (залишок від ділення по модулю *f(n)* добутку $d \cdot e$ дорівнює 1);

4) у результаті отримують пару чисел відкритого ключа $\{e, n\}$ та пару чисел приватного ключа $\{d, n\}$.

Передача повідомлення від одного користувача до іншого здійснюється у наступним чином (рис. 1.4):

1) беруть відкритий ключ $\{e, n\}$ і текст повідомлення *m* та зашифровують повідомлення з використанням відкритого ключа користувача, який передає повідомлення: $c = E(m) = m^e \bmod n$;
2) отримане зашифроване повідомлення розшифровується з використанням приватного ключа $\{d, n\}$ по формулі: $m = D(c) = c^d \bmod n$.

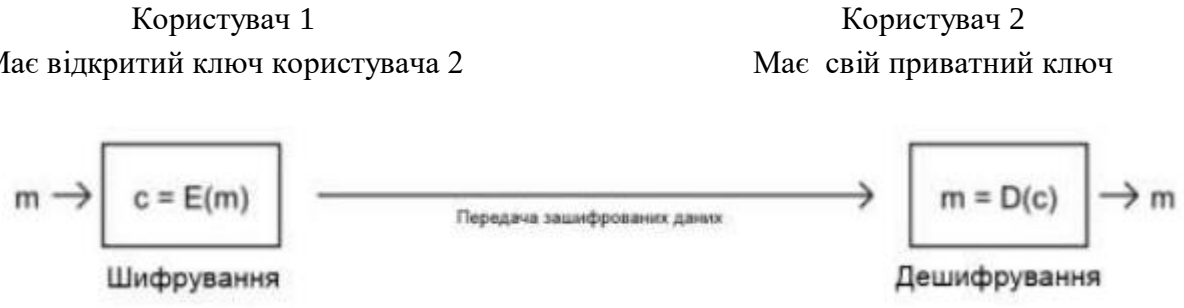

Рис. 1.4. Схема обміну заними відповідно до алгоритму шифрування RSA

Шифрування варто використовувати для всіх даних та полів, які існують у системі для безпечної передачі комерційної та приватної інформації.

Розглянемо оптимізований алгоритм RSA. Він полягає у тому, що дані пакуються за допомогою алгоритмів стиснення, а потім шифруються за допомогою обчислення RSA. Закодоване повідомлення надсилається за допомогою Інтернету бенефіціару. Після прийняття повідомлення спочатку його потрібно розшифрувати. У цей момент розшифроване повідомлення має бути розпаковано, щоб отримати небхідні дані.

RSA працює повільно, коли використовують 1024 або більше біт для представлення простих чисел. У описаному вище алгоритмі час, затрачений на шифрування, пропорційний числу одиничних біт у двійковому представленні відкритої експоненти *e*, а дешифрування є повільнішим за шифрування з достатньо великим показником обчислювальної складності. Для прискорення передачі шифрованої інформації необхідно скоротити кількість операцій. Це можна зробити, знаючи *p* і *q* у розкладі *n* = *p***q* та обчисливши:

$$m_p = c^d \bmod p = c^{d \bmod p-1} \bmod p \ ,$$

$$m_q = c^d \bmod q = c^{d \bmod q-1} \bmod q .$$

Так як *p* і *q* є числами порядку 2^{512}, потрібно буде зробити два піднесення до степеня з показником у 512 біт по модулю 512-бітного числа, що виконується швидше, ніж одне піднесення у степінь з 1024-бітним показником по модулю 1024 бітного числа. Потім необхідно відновити повідомлення *m*, використовуючи m_p та m_q. Це дозволяє зробити китайська теорема про залишки.

Таким чином, основним обчислювально складним етапом алгоритму RSA є обчислення за формулами: $c = m^e \bmod n$ та $m = c^d \bmod n$. Для підвищення швидкості на стороні дешифрування використовують CRT – китайську теорему залишку.

Основна ідея запропонованого методу оптимізує роботу алгоритму за допомогою використання китайської теореми залишку та розширює безпеку за допомогою двох відкритих ключів у розшифровці. Пропонуючи алгоритм CRT з алгоритмом RSA, комбінація допоможе зробити швидкий зв'язок і стиснути великі дані. Запропонована модель для оптимізованого алгоритму RSA наведена на рисунку 1.5.

Запропонована структура намагається перепроектувати виконання криптосистеми RSA за допомогою системи, яка має покращити швидкість на стороні шифрування RSA за допомогою теореми китайського залишку CRT і підвищити безпеку даних за допомогою двох ключових факторів, а не єдиний ключовий фактор. Використання спорадичного цілого числа в будь-якій точці кодування порівнянного повідомлення кілька разів, воно виглядатиме інакше.

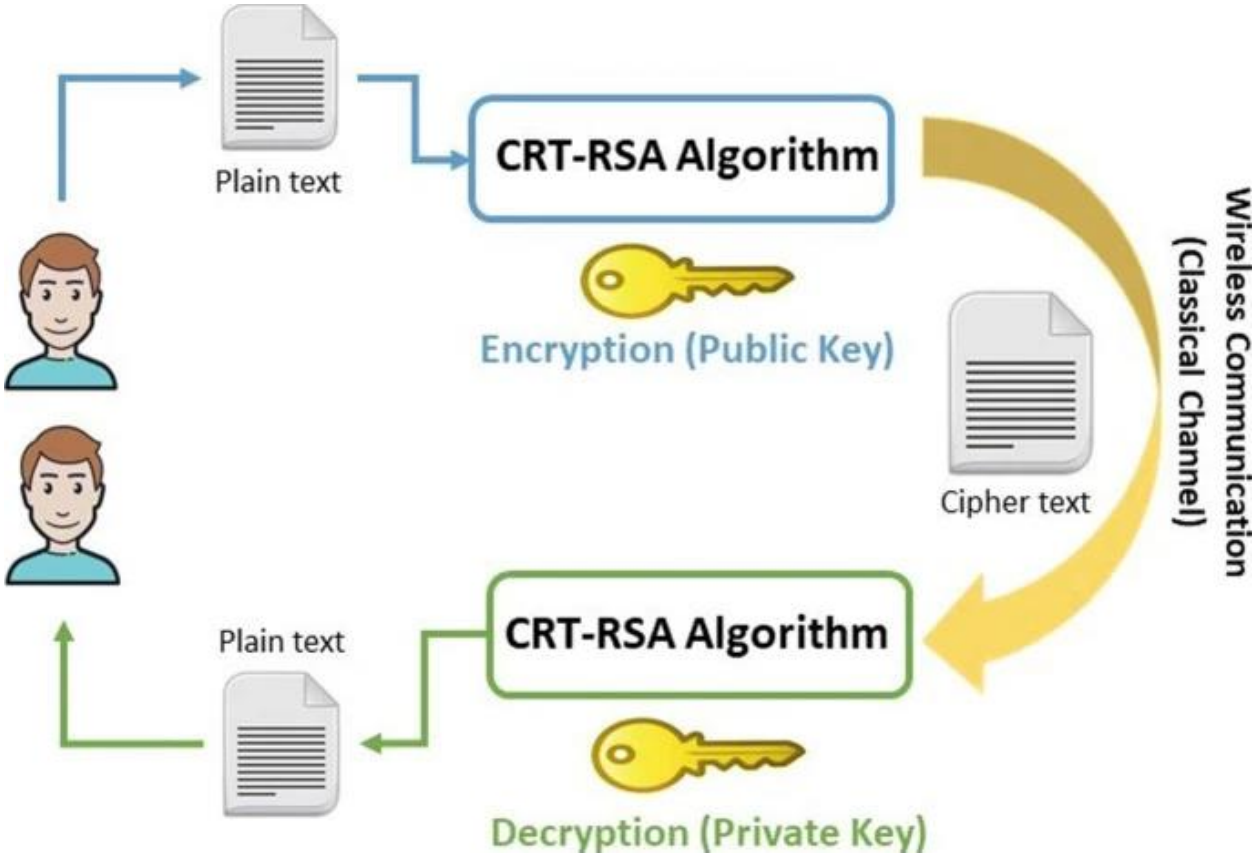

Рис. 1.5. Запропонована оптимізована модель алгоритму RSA

Загальна ідея цієї стратегії полягає в тому, щоб прискорити виконання обміну повідомленнями і зробити його поступово безпечним і одночасно зменшити час дешифрування. Використовуючи чотири простих числа та двозначну композицію вмісту для кожного повідомлення, перевірка оцінки стає все більш непередбачуваною. Нижче наведено деякі кроки для процесу шифрування/дешифрування (рис. 1.6).

На етапах шифрування та дешифрування a, b, c і d є простими числами. Для факторізації n і z цілі значення, тоді як x позначають шифрований текст. На всіх етапах генеруються прості числа p і q, а потім обчислюється модуль n. аналогічно, e для шифрування відкритого тексту і d для розшифровки зашифрованого тексту.

Algorithm 2 Steps for encryption/decryption are as follows.

prime numbers: a, b, c, and d

1: **def** n = ab, and z = cd {Calculate Estimation (n, z)}
2: $\phi(n)$ = (a-1) (b-1); $\phi(z)$ = (c-1) (d-1) {Calculate}
3: **while** 1 < p < n and 1 < g < z **do**
4: gcd (p, $\phi(n)$) = 1 {Encryption}
5: def gcd(n, z) {Factor of n and z}
6: gcd (p, ϕ (z)) = 1
7: **end while**
8: **for** d: (xd = 1 mod ($\phi(n)$)) **do**
9: **for** T: (tg = 1 mod ($\phi(z)$)) **do**
10: xa = x mod (a-1); xb = x mod (b-1) {Calculate private key}
11: xc = x mod (c-1), xd = x mod (s-1)
12: **Return**
13: **end for**
14: **end for**
15: Open key KU = (p,n),(g,z) {Cipher-text to Plain-text}
16: private key VK = t,z, xa, xb, xc, xd
17: **Return**

Рис. 1.6. Кроки для процесу шифрування/дешифрування

У RSA за допомогою методу кількох ключів використовували прості числа для генерування більш ніж одного відкритого та особистого ключа, що забезпечує кращий захист порівняно з набором правил RSA та RSA-CRT. У RSA, використовуючи метод кількох ключів, ми використовували публічні та

персональні ключі. Це робить особу більш захищеною, оскільки вона не зазнає постійних атак чи пограбування з використанням неавторизованих сторін і покращує захист та ефективність обміну інформацією через мережу. Однак набагато меншим темпом оцінюється набір правил RSA і CRT-RSA. Оскільки звичайний RSA використовував великі числа для створення одного відкритого ключа і одного непублічного ключа для шифрування та дешифрування, це робить його набагато менш стійким, і він без проблем розкладається. RSA займає більше часу за допомогою використання кількох ключів для шифрування та дешифрування, ніж CRT-RSA, використовуючи використання CRT у розшифровці набору правил RSA, він вимагає набагато менше часу обробки та меншу кількість спогадів для самого останнього декодованого результату порівняно з RSA за допомогою використання кількох клавіш.

Порівняння між існуючим RSA та запропонованим CRT-RSA. Виконання декодування RSA-CRT виконується з коду на основі JavaScript. Деякі експерименти залежать від різних ситуацій, включаючи різну довжину повідомлень і відмінні розміри ключів для декодування повідомлень. Ці експерименти використовуються для перевірки елементів презентації та безпеки CRT-RSA.

Час шифрування — це період, який потрібен для розрахунку для кодування простого вмісту. У міру збільшення довжини ключа час шифрування береться на обидва кроки обчислення безперервно. Дані свідчать про те, що час шифрування RSA-CRT майже в кілька разів швидше, ніж час шифрування RSA (рис. 1.7).

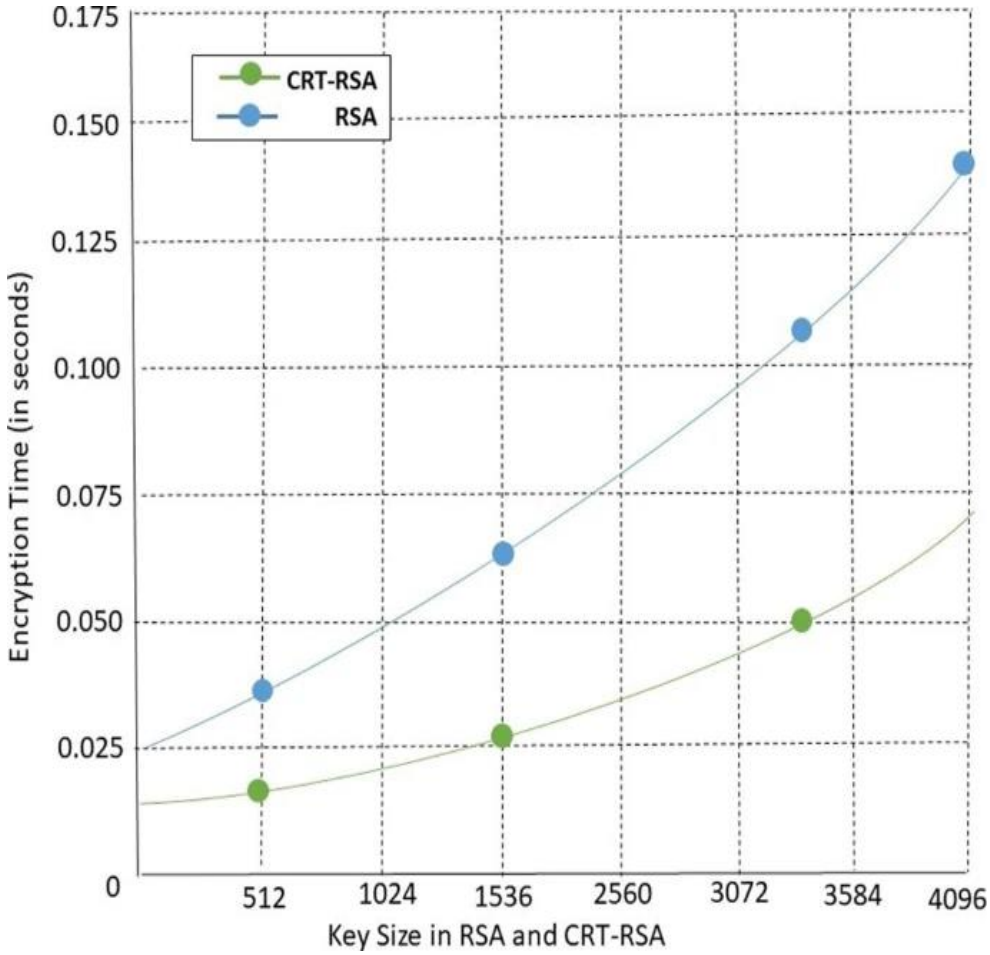

Рис. 1.7. Кроки для процесу шифрування/дешифрування

1.5. Постановка задачі

Головною тенденцією безпеки сьогодні є запобігання зовнішнім атакам застсунків. Безпека додатків стала пріоритетом для компаній, оскільки команди розробників випускають все частіше, що призводить до помилок, які впливають на безпеку програми. Експлойти веб-додатків є найпоширенішим типом зовнішньої атаки. Атаки ботів поширилися на програми, які впливають на всі функціональні області в організації.

Метою є підвищення ефективності та безпеки комунікації в ІТ-компанії шляхом розробки Web-застосунку з надійною системою шифрування корпоративної інформації й автоматизацією процесів управління персоналом.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- розкрити теоретичні засади комунікаційної діяльності ІТ-компаній та сучасний стан програмного забезпечення, яке використовується для її підтримки;
- дослідити підходи до забезпечення надійності захисту корпоративної інформації;
- обґрунтувати вибір технологій і засобів розробки Web-застосунку для забезпечення комунікаційної діяльності;
- розробити та здійснити програмну реалізацію Web-застосунку для забезпечення комунікаційної діяльності ІТ-компанії з надійною системою шифрування інформації.

Об’єктом дослідження є комунікаційна діяльність в ІТ-компаніях.

Предметом дослідження є програмні засоби для забезпечення безпечної комунікаційної діяльності ІТ-компанії.

Базисні характеристики мережі для внутрішньої комунікації на стадії проектування повинні бути наступними: правильний інтерфейс по чітким визначенням UI/UX, безпечна авторизація, та реалізовані основні функції на етапі проектування.

Також, основним критерієм є врахування UI/UX дизайну продукту. Хороший дизайн інтерфейсу користувача покращує зовнішній вигляд продукту і виглядає привабливо. Менше – краще, тому интерфейс повинен виглядати чистим і без зайвих елементів. Користувачам слід представляти лише той контекст, який дійсно необхідний. Інформацію або функціонал, яка не є життєво важливою, слід видалити, оскільки вона може заплутати користувачів. Загалом, він повинен виглядати простим у використанні та змушувати користувача відчувати, що цей продукт може виконувати роботу.

Висновок до розділу 1

Установлено, що комунікаційна діяльність ІТ-компаній зазнає трансформацій, прискорених у зв'язку з пандемією Covid-19, які полягають у все більшому переході до розробки програмного забезпечення територіально розподіленою командою фахівців. Віддалений режим роботи обумовлює необхідність у використанні програмного забезпечення, яке здатне забезпечити ефективні розподілені внутрішньокорпоративні комунікації.

У результаті проведеного дослідження виявлено, що до ключових функцій, які потрібно автоматизувати для підтримки комунікації в ІТ-компанії, необхідно віднести кадровий облік: автоматизацію всіх процесів роботи з персоналом – від кадрового адміністрування до завдань планування і управління. Сучасні кадровики (РР – співробітники) є HR-фахівцями (HR – людські ресурси, англ. Human resources) й займаються розвитком корпоративної культури, оцінкою ефефктивності співробітників, визначенням вимог до майбутніх співробітників, веденням бази даних співробітників та кандидатів, організацією опитувань, проведенням зустрічей 1:1 і визначенням продуктивності роботи, оцінки співробітників (англ. Performance review).

Використання Інтернет-технологій для розподіленого у просторі спілкування обумовило різке зростання кількості програмних засобів, які спрямовані на ефективне забезпечення комунікаційної діяльності організацій. До найкращих програм для ефективного командного ділового спілкування сьогодні можна віднести корпоративні месенджери Телеграм, Slack, Flock, Skype, Google Meet, Samepage, хмарний сервіс Highfive, цифрове робоче місце, яке об'єднує розрізнені кадри Beekeeper. Їх використання спрощує спілкування між колегами, дозволяє відправляти голосові та текстові повідомлення, забезпечує відеозв'язок, формує канали, які є робочим простором, дозволяє підтримувати та координувати різні графіки діяльності, що сприяє співробітництву та швидкому прийняттю

рішень. Однак корпоративні месенджери не можуть повністю покрити сторони автоматизації HR та PP функцій, та приділяють недостаню увагу питанням забезпечення безпеки. Єдиного рішення, яке змогло покрити всі потреби аутсортингового розробки програмного забезпечення компаній та мало необхідний функціонал для автоматизації управління й оцінки персмоналу з надійним захистом корпоративної інформації, немає.

Установлено, що для забезпечення надійності захисту використовують часткове або повне шифрування повідомлень та файлів, самознищення повідомлень і цілих чатів, блокування можливостей для скріншотів. Різко знижує витік корпоративної інформації застосування багатофакторної автентифікації. Наскрізне шифрування (англ. end-to-end encryption), при якому доступ до інформації мають тільки ті користувачі, які приймають участь у спілкуванні, не дозволяє отримати доступ до криптографічних ключів третім особам, є найбільш надійним способом захисту даних. Для обміну ключами при цьому можуть бути застосовані симетричний та асиметричний алгоритми. Симетричне шифрування використовує єдиний ключ, який використовується обома сторонами, що спілкуються, для шифрування та дешифрування. Асиметричне шифрування використовує два ключі, один з яких є приватним, а другий – відкритим. Виявлено найнебезпечніші алгоритми серед найбільш часто використовуваних алгоритмів шифрування: AES, Triple DES, Blowfish, Twofish, RSA.

Застосовування комбінації алгоритму RSA з китайською теоремою про залишки CRT дозволяє суттєво прискорити швидкість шифрування/дешифрування повідомлень і передачу інформації та забезпечує стиснення великих даних. Китайська теорема про залишки розвантажує RSA від піднесення у степінь з дуже великим показником, зменшуючи його розрядність вдвічі. Таким чином застосування криптографічного алгоритму RSA-CRT дозволяє збільшити швидкість алгоритму на практиці втричі та покращити безпеку даних за допомогою двох, а не одного ключового фактора.

РОЗДІЛ 2. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ WEB-ЗАСТОСУНКУ

2.1. Підходи до проектування і створення Web-застосунку

Web-застосунок є сайтом з елементами інтерактиву. Мови програмування відповідають за зручний інтерфейс редагування даних, зв’язок бази даних з кінцевим сайтом, складні форми, безліч різних фрагментів. Масштабні проекти web-застсунків зазвичай мають такі особливості:

– наявність набору сукупностей (підсистем), які знаходяться в тісній взаємодії, виконують певні локальні цілі;

– складність опису (велика кількість функцій, процесів, елементів даних і складних взаємозв'язків між ними), що вимагає ретельного моделювання і аналізу даних і процесів.

Web- застосунок є ефективним засобом для забезпечення комунікації у наш час. Як і будь-який інший інструмент, заснований на принципі безпосереднього відгуку, перш за все він повинен зацікавити відвідувача, а потім спонукати його на певні дії. Проте, багато хто ігнорує цю особливість головної сторінки, що часто призводить до того, що відвідувачі не затримуються на сайті надовго і залишають його, ледь знайшовши.

Кожен вебсайт має містити на головній сторінці тільки важливу для користувача інформацію, та давати можливість легко орієнтуватися по потрібним користувачу статтям і розділам. Інакше сайт не буде ефективним і не матиме сутності існування. Адже на даний час специфіка сайтів постійно вдосконалюється для максимально зручного і простого користування.

Зробивши лише кілька змін, простий web-сайт може перетворитися на більш надійний і ефективний інструмент. Важливо пам'ятати, що з дня на день на потенційних клієнтів обрушується потік інформації та різних рекламних повідомлень, і що в плані завоювання їхньої уваги існує гранично жорстка

конкуренція. web-сайт, здатний привернути увагу і викликати цікавість, спонукає клієнтів не лише переглянути залишилися сторінки і залишити відгуки, але і знову відвідати його через деякий час, а також рекомендувати своїм друзям і знайомим.

Також, беручи до уваги специфіку даного сайту, варто зазначити, що сайт такого типу має бути не тільки привабливим за візуальними якостями, але й максимально корисним. Адже мова йде про сферу інформаційних послуг, суспільних комунікацій і співпраці, то ж подібні сайти мають містити якомога більше актуальної і бажаної для використання інформації.

Отже, є декілька основних компонентів, які може містити подібний сайт. По-перше, опис сайту – для чого він і на кого орієнтований. Головна сторінка може містити розділи, де є якомога більше інформації про діяльність сайту чи компанії/фірми/угрупованню, розділ з контактами, де можна знайти потрібні контакти для зв`язку. Кожен з розділів має бути максимально зрозумілим до використання. Також повинна бути сторінка користувача, де має міститись особистий кабінет, з якого користувач/клієнт може безпосередньо через сайт зв'язатися зі стороною яка надає інформацію.

Для успішної реалізації проекту та подальшого його просування, об’єкт проектування повинен бути насамперед детально описаний, побудовані функціональні та інформаційні моделі. Досвід, набутий при виконанні дипломної роботи показує, що розробка такого проекту трудомістка і довготривала робота, яка вимагає високої кваліфікації від розробників, які займаються розробкою проекту.

2.2. Вибір Frontend фреймворків та бібліотек

Застосунок може бути розроблено як на одному тільки фреймворкі, так і на одних тільки бібліотеках. Однак частіше усього розробники використовують фреймворк та сторонні бібліотеки одночасно.

Серед безлічі фреймворків, бібліотек та можливостей розробки Web-застосунку варто виділити кілька основних. До них входять такі: ReactJS, AngularJS, VueJS (рис. 2.1). Вони вважаються самими популярними та ефективними у використанні та є дуже гнучкими у можливостях, що до створення сайту. У кожного є свої переваги та зручності.

Рис. 2.1. Найпопулярніщі фреймворки для Frontend розробки

AngularJS – це веб-фреймворк із відкритим кодом на основі JavaScript, який в основному підтримується Google та спільнотою користувачів та корпорацій для вирішення багатьох проблем, що виникають при розробці односторінкових додатків. Він спрямований на спрощення як розробки, так і тестування таких додатків, забезпечуючи структуру для архітектури модель-вигляд-контролер (MVC) на стороні клієнта та модель-вигляд-модель-перегляд (MVVM), а також компоненти, які зазвичай використовуються в інтернет-додатках з багатим функціоналом. AngularJS використовується як інтерфейс MEAN, що складається з бази даних MongoDB, фреймворку сервера веб-додатків Express.js, самого Angular.js та середовища виконання сервера Node.js.

Фреймворк Angular відомий своєю функціональністю. Його переваги:

1. Архітектура Model-View-Controller не тільки надає цінність фреймворку під час створення клієнтської програми, але також створює основу для інших функцій, таких як прив’язка даних і області видимості. Завдяки архітектурі MVC можна ізолювати логіку програми від рівня інтерфейсу користувача та підтримувати розділення проблем. Контролер отримує всі запити до програми та працює з моделлю, щоб підготувати будь-які дані, необхідні для представлення. Подання використовує дані, підготовлені контролером, і відображає остаточну презентабельну відповідь.

2. Розширена архітектура дизайну. Деякі з великих веб-додатків містять багато компонентів. Angular спрощує спосіб керування цими компонентами, навіть якщо новий програміст приєднується до проекту після того, як процес розробки вже почався. Архітектура побудована таким чином, що допомагає програмісту легко знаходити та розробляти код.

3. Модулі. Модуль – це механізм, який групує пов’язані директиви, компоненти, канали та служби таким чином, що їх можна об’єднати з іншими модулями для створення програми. Програму на основі Angular можна розглядати як головоломку, де кожен модуль потрібно, щоб мати можливість побачити повну картину. Існує кілька способів додати різні елементи до модуля. Angular вирішує проблему використання глобальної функції, обмежуючи область дії всіх функцій модулем, у якому вона була визначена та використана.

4. Служби та ін'єкція залежностей (DI). Для сервісу або компоненту іноді можуть знадобитися інші залежні сервіси для виконання завдання. Для виконання цих залежностей використовується шаблон дизайну Dependency Injection. Він розподіляє завдання між різними службами. Служба клієнта не створюватиме залежний об’єкт, скоріше він буде створений і введений за допомогою Angular-інжектора. Інжектор Angular відповідає за створення екземплярів служби та введення їх у такі класи, як компоненти та служби.

5. Спеціальні директиви покращують функціональність HTML і підходять для динамічних програм на стороні клієнта. Усі вони починаються з префікса ng, щоб HTML міг їх ідентифікувати.

6. TypeScript: кращі інструменти, чистіший код і вища масштабованість. Angular написаний за допомогою TypeScript, який є над набором для JavaScript. Він повністю відповідає JavaScript, а також допомагає виявляти й усувати типові помилки під час кодування. У той час як невеликі проекти JavaScript не потребують такого покращення, додатки корпоративного масштабу потребують, щоб розробники зробили свій код чистішим і частіше перевіряли його якість.

Серед переваг Angular також ж є й недоліки, які є досить популярними та вагомими для деяких проектів:

1. Обмежені можливості SEO. Основним недоліком використання Angular є обмежені параметри SEO та погана доступність для сканерів пошукових систем.

2. Angular є багатослівним і складним. Часта скарга, яку ви почуєте від розробників Angular, - це багатослівність інструменту. І ця проблема не сильно змінилася з часів AngularJS.

3. Крута крива навчання. Якщо ви залучаєте нових розробників, які знайомі з JavaScript, для використання нового Angular, їм буде важко в порівнянні з адаптацією React або Vue. Це пояснюється тим, що набір тем і аспектів, які потрібно висвітлити, досить великий.

4. У документації CLI бракує деталей. Деякі розробники висловлюють занепокоєння щодо поточного стану документації CLI. Хоча командний рядок дуже корисний для розробників Angular, ви не знайдете достатньо інформації в їхній офіційній документації на GitHub, і вам доведеться витрачати більше часу на вивчення потоків на GitHub, щоб отримати відповіді.

Незважаючи на те, що платформа має певні недоліки, Angular є повно функціональною та динамічною структурою. А його зручність, гнучкість і ремонтопридатність робить його унікальним і дає шанси створювати чудові та

успішні веб-додатки. Такі компанії як Google, Microsoft, Upwork, Udemy та багато інших, мають багато компонентів та сервісів, розроблених на Angular.

Далі розглянемо React, який не є фреймворком у чистому вигляді, а є модифікованою бібліотекою, яка включає у себе Virtual DOM, JSX, изоморфність і компонентний підхід. Бібліотека React застосовується для створення інтерфейсу та є ефективним рішенням для динамічних web-додатків. Розглянемо основні переваги використовування React:

1. Надійна підтримка великої організації (Facebook), яка прагне підтримувати розвиток проекту. На мою думку, це пріоритет номер один для будь-якої бібліотеки, тому що без цього: час витрачається даремно на оновлення бібліотеки, яку ви ніколи не будете використовувати, мати кодову базу з бібліотекою, яку важко підтримувати, тому що через кілька років lane, найняти розробників для підтримки непідтримуваної бібліотеки дуже важко.

2. Як і для будь-яких бібліотек, підтримка спільноти з відкритим кодом має вирішальне значення для успіху будь-якої фреймворку, оскільки це дає змогу використовувати більше готових компонентів, які можна було б використовувати прямо з коробки => робить розробку за допомогою React легкою.

3. Компоненти React з Stateful та Stateless роблять організацію вашого коду легкою. Ці компоненти також дозволять писати чисті модульні тести для логіки.

4. Життєвий цикл компонента React. Він пропонує різноманітні методи життєвого циклу, що дозволяє обробляти різні сценарії завантаження та маніпулювання даними в інтерфейсі користувача.

5. Документація React дуже добре підтримується з великою кількістю прикладів, що пояснюють кожну функцію.

6. Чуйність (Responsiveness) є дуже важливим критерієм при виборі інтерфейсу користувача, а React дуже чуйний. Він робить деякі акуратні оптимізації при повторному візуалізації з використанням віртуальної DOM і

повторно відтворює лише ті частини DOM, які змінилися. Завдяки цій оптимізації програми React працюють дуже швидко.

7. React Native дозволяє створювати програми, які охоплюють веб-і мобільні інтерфейси (iOS і Android). Це робить вивчення React ще більш привабливим, оскільки, використовуючи одну бібліотеку, ви можете створювати програми, які охоплюють Інтернет, iOS та Android.

8. create-react-app — це зусилля Facebook (творців React), завдяки яким розпочати роботу з React дуже легко. Він виконує всю важку роботу з конфігураціями за вас і дозволяє зосередитися лише на розробці.

9. Невеликий розмір, мінімізований React + React DOM становить менше 150 Кб, що робить завантаження інтерфейсу користувача за допомогою react дуже швидко.

10. React + Enzyme (за підтримки Airbnb) + Sinon + Mocha + Chai робить модульне тестування компонентів інтерфейсу користувача веселим і покращує загальну ремонтопридатність проекту.

Також, визначимо і мінуси:

1. React може бути дуже розчарованим, якщо ви не почнете думати в React. React забезпечує ієрархію потоку даних зверху вниз і не пропонує можливості обміну даними назад. Це великий зрушення в мисленні, що виникло з Angular 1.x. Це обмеження дійсно є важливим фактором, який визначає, як організувати ваш код і як ви можете написати власні компоненти.

2. Через самонав’язану ієрархію зверху вниз, у вас виникнуть важкі батьки з дурними дітьми. Оскільки батькам доведеться виконувати основну частину роботи, вони зазвичай керують станом, передаючи зворотні виклики дочірнім компонентам. Якщо не спроектовано належним чином, це може призвести до пекла зворотного виклику, де ви можете мати зворотні виклики, які проходять як кілька дітей.

3. Як і в будь-якій бібліотеці, вам доведеться перебирати кілька проектів, щоб знайти способи уникнути «важких батьківських компонентів» у

React. Одним з очевидних рішень є використання деяких інструментів з відкритим кодом, як-от Redux, але ми вибрали його варіант, щоб задовольнити наші потреби.

У кожного фреймворка є свої плюси і мінуси, а це означає, що при розробці продукту потрібно зробити правильний вибір для кожного окремого випадку. Для конкретно цього кейсу було обрано React.

2.3. Бібліотека управління станом Redux

Протягом багатьох років масове зростання React.JS породило різні бібліотеки управління станом. Бібліотеки управління станом, доступні в React, які є в нашому розпорядженні, величезні. Тому знання того, яку бібліотеку управління вибрати для конкретного проекту, щоб не захоплюватися шумом і новинами від спільноти React, є важливим фактором для полегшення розробки програми. Деякі розробники вирішують цю проблему, використовуючи React Hooks; інші поєднують їх із бібліотеками керування станом програми, такими як Redux чи MobX.

Першим у нашому списку є Redux. Це вже деякий час, майже перша бібліотека управління станом на основі реакції. Бібліотека управління Redux була створена для вирішення проблеми в додатку для електронної комерції. Він надає об’єкт JavaScript під назвою store, який після налаштування включає всі стани у програмі та оновлює їх, коли це необхідно. Ось спрощена візуалізація того, як працює Redux.

Redux часто використовується з React. Причина полягає в тому, що Redux обробляє оновлення стану у відповідь на дії користувача, особливо в інтерфейсі користувача. Крім того, Redux можна використовувати як самостійне управління станом з будь-якого фреймворка. Redux є однією з найпопулярніших бібліотек управління станом React.

По-перше, Redux дозволяє керувати станом програми в одному місці та зберігати зміни у програмі більш передбачуваними та відстежуваними. Це

полегшує визначення змін, що відбуваються у додатку. На жаль, усі ці переваги мають певні обмеження та компроміси. Часто розробники вважають, що використання Redux додає деякий шаблонний код, що робить дрібниці, здавалося б, приголомшливими; однак це залежить виключно від архітектурного рішення програми.

Один із найпростіших способів дізнатися, коли дійсно потрібно використовувати Redux, це коли локальне керування станом починає виглядати безладно. У міру розвитку програми розподіл стану між компонентами стає втомливим. У цей момент виникає необхідність пошуку способу зробити процес оптимальним.

Використання Redux з React позбавляє від клопоту підняття на північ, полегшуючи відстеження, яка дія спричиняє будь-які зміни, отже, спрощуючи програму та полегшуючи її обслуговування. Давайте подивимося на деякі компроміси, які виникають при використанні Redux для управління станом:

1. Підтримка громади. Як офіційна бібліотека прив'язки для React та Redux, React-Redux охоплює велику спільноту користувачів. це полегшує запити про допомогу, дізнатися про найкращі методи роботи, використовувати бібліотеки, побудовані на React-Redux, і повторно використовувати свої знання в різних програмах. Це найпопулярніша бібліотека управління станом React на Github.

2. Підвищує продуктивність. React Redux забезпечує оптимізацію продуктивності, так що лише підключений компонент повторно відтворюється лише тоді, коли це потрібно; отже, збереження глобального стану програми не призведе до жодних проблем.

3. Redux робить стан передбачуваним. У Redux стан завжди передбачуваний. Якщо той самий стан і дія переходять до редуктора, він отримає той самий результат, оскільки редуктори є чистими функціями. Стан також незмінний і ніколи не змінюється. Це дає можливість виконувати складні завдання, такі як нескінченні скасування та повторення. Також можна реалізувати подорож у часі

— тобто можливість переміщатися назад і вперед між попередніми станами та переглядати результати в режимі реального часу.

4. Зберігання стану на локальному сховищі. Можливе збереження частини стану програми в локальному сховищі та відновлення після оновлення. Це робить зберігання таких речей, як дані кошика в локальному сховищі, дійсно чудовим.

5. Відтворення на стороні сервера. Ми також можемо використовувати redux для відтворення на стороні сервера. З його допомогою ви можете обробляти початкову візуалізацію програми, надсилаючи стан програми на сервер разом із відповіддю на запит сервера.

6. Redux обслуговується. Redux суворо ставиться до того, як повинен бути розроблений код, що полегшує розуміння будь-якої структури програми Redux для тих, хто знає Redux. Як правило, це полегшує догляд. Це також допоможе вам відокремити вашу бізнес-логіку від дерева компонентів. Для великомасштабних додатків дуже важливо, щоб ваша програма була більш передбачуваною та придатною для обслуговування.

7. Налагодження робиться легко. Redux полегшує налагодження програми. Записуючи дії та стан, легко зрозуміти помилки кодування, мережеві помилки та інші форми помилок, які можуть виникнути під час виробництва. Окрім ведення журналів, він має чудові інструменти розробника, які дозволяють переміщувати дії в часі, зберігати дії під час оновлення сторінки тощо. Для середніх та великих програм налагодження займає більше часу, ніж фактична розробка функцій.

У більшості випадків об'єкти стану і мутації стану не обов'язково пов'язані з одним компонентом (мал. 2.2). Вони пов'язані через дерево компонентів, і потрібно піднімати і опускати стан (мал.2.3). Рішення, таким чином, полягає в тому, щоб ввести бібліотеку управління станом, таку як MobX або Redux. Вона дає інструменти для збереження стану, зміни стану і отримання оновлень стану. Існує одне місце для пошуку, одне місце для зміни і одне місце для отримання оновлень стану. Він дотримується принципу єдиного джерела правди. Це

полегшує міркування про зміни станів, тому що вони відокремлюються від компонентів.

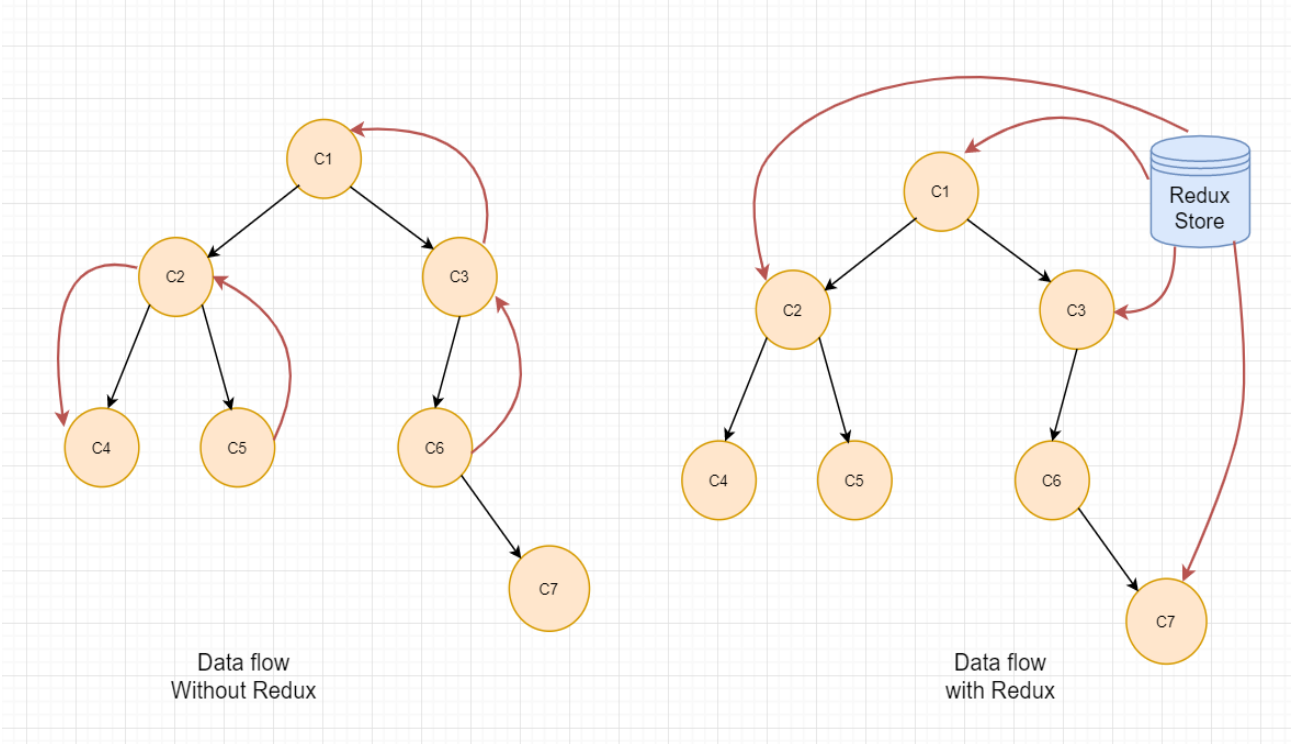

Рис. 2.2. Порівняння зміни стану у компонентах з Redux

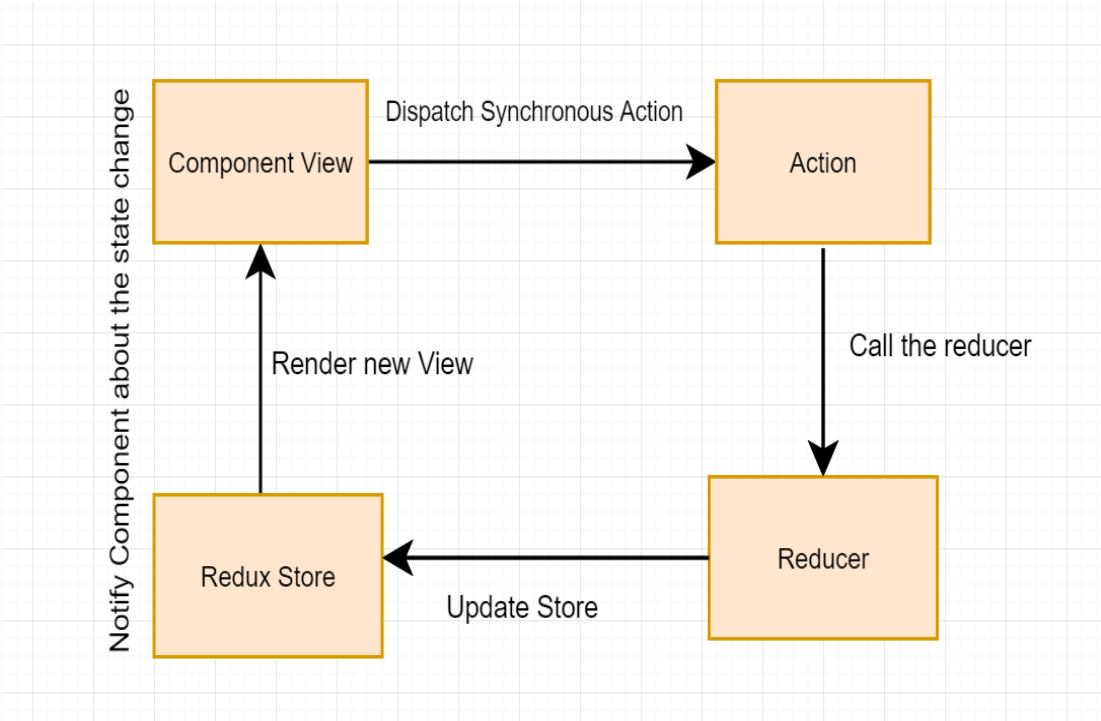

Рис. 2.3. Потік даних у Redux

Бібліотеки управління станом, такі як Redux і MobX, часто мають доповнення до утиліт, наприклад, для Angular вони мають angular-redux і mobx-angular, щоб дати компонентам доступ до стану. Часто ці компоненти називають контейнерними компонентами або, якщо бути більш точним, пов'язаними компонентами. З будь-якої точки вашої ієрархії компонентів ви можете отримати доступ до стану і змінити його, оновивши свій компонент до пов'язаного.

Усі стани web-застосунку збережено в об'єкті всередині одного стору (store). Єдиний спосіб змінити дерево стану - це викликати екшену (action), об'єкт описує те, що сталося. Щоб вказати, яким чином екшени перетворюють дерево стану, існують чисті "редюсери". Купа односторінкових фреймворків і бібліотек додатків мала ті ж проблеми управління станом, які в кінцевому підсумку були вирішені за допомогою всеосяжної моделі потоків. Redux є наступником такого підходу.

У MobX він знову рухається в зворотному напрямку. Ми знову починаємо мутувати стан безпосередньо, не використовуючи переваги функціонального програмування. У той час як в Redux є встановлена процедура налаштування станів, MobX поступається в цьому. Але було б розумно прийняти кращий досвід в MobX.

Проміжне програмне забезпечення Redux (Redux middleware) — це функція або фрагмент коду, який знаходиться між дією і редуктором і може взаємодіяти з відправленою дією, перш ніж досягти функції редуктора (рис. 2.4). Проміжне програмне забезпечення Redux можна використовувати для багатьох цілей, таких як ведення журналів (наприклад, redux-logger), асинхронні виклики API тощо.

Redux Thunk — це проміжне програмне забезпечення, яке дозволяє викликати розробників дій, які повертають функцію (thunk), яка приймає метод диспетчеризації магазину як аргумент і яка згодом використовується для відправлення синхронної дії після завершення API або побічних ефектів.

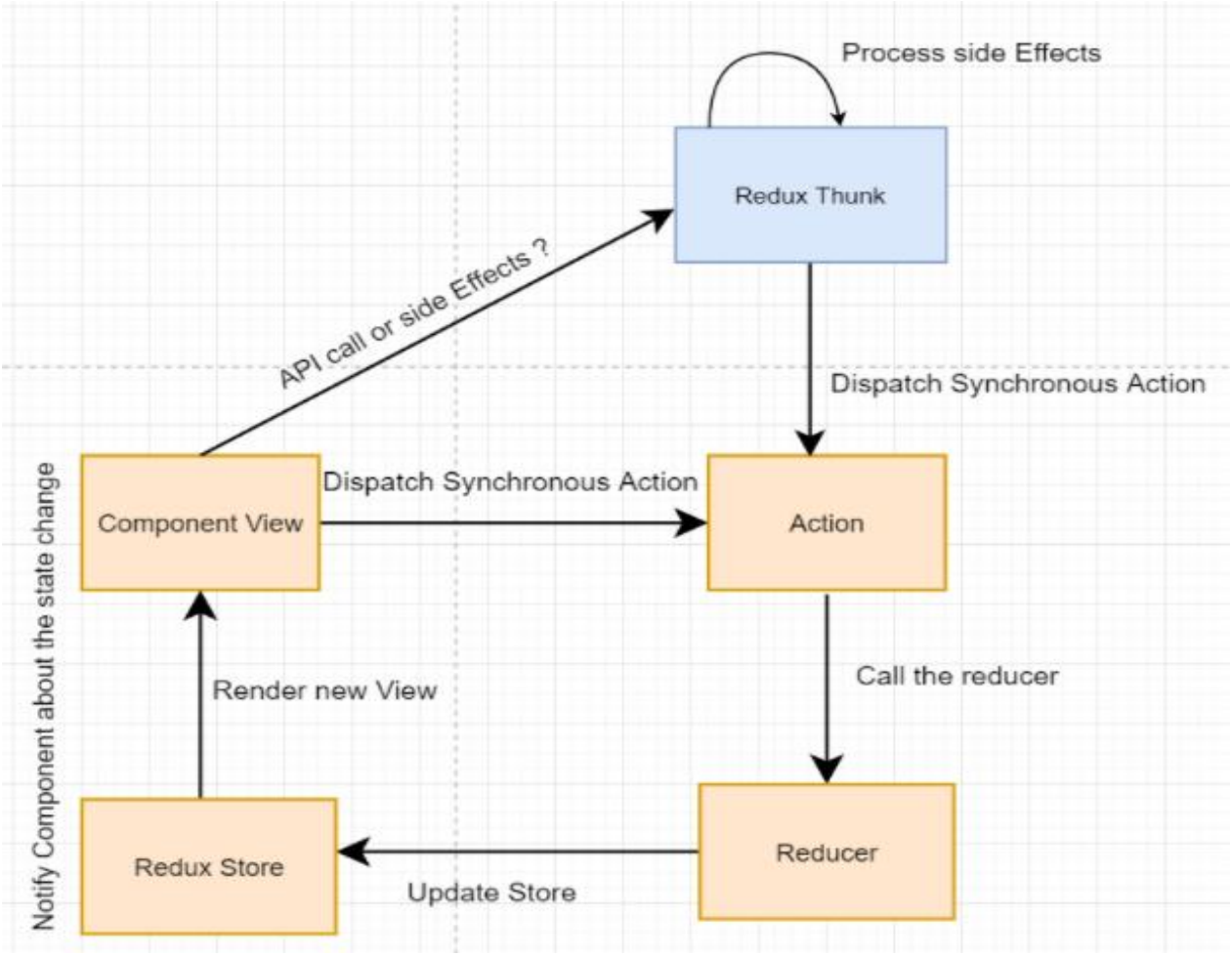

Рис. 2.4. Як працює Thunk

2.4. Бібліотека для створення інтерфейсу Bootstrap

Для створення UI варто розглянути Bootstrap, який має деякі дуже особливі речі, які потрібні під час створення перших ітерацій веб-інтерфейсу, для яких Bootstrap особливо добре підходить.

Неймовірно низька енергія активації, неймовірно висока швидкість ітерації Bootstrap доступний на вимогу, його можна використовувати за лічені секунди, і немає більш швидкого способу запустити інтерфейс веб-програми за допомогою повної документованої системи проектування. «Швидкість» вимірюється тим, наскільки швидко та рентабельно ви досягаєте цілей. Коли справа доходить до побудови веб-інтерфейсу, це означає, що інтерфейс буде опублікований і повторюватися макетом, вибором керування та потоком користувачів, доки ви не

знайдете те, що люди хочуть використовувати. Якщо ви успішно здійснюєте доставку продукту, надаєте цінність та тримаєте користувачів, може виникнути певний момент, коли час завантаження буде скорочуватися на мілісекунди. Це не буде в перший день, і якщо ви робите це правильно, те, що напружує продуктивність вашого додатка, безпосередньо пов’язане з вашими основними інноваціями, а не з обслуговуванням товарного інтерфейсу.

Повна та доступна документація, зрозуміла кожному, тому що елементи керування Bootstrap добре задокументовані як в їхній офіційній документації, так і в різних спільнотах підтримки. Це перевага, яка поширюється на кожного члена вашої розширеної команди розробників і дозволяє масштабувати реалізацію інтерфейсу користувача.

Створення та підтримка еталонних реалізацій для елементів керування інтерфейсу користувача є надзвичайно дорогими з точки зору часу розробника і легко не відповідає напряму продукту. Попередня підготовка цих ресурсів звільняє розробників і дизайнерів, щоб вони могли зосередитися на проблемах вищого порядку і підтримувати процес розробки інтерфейсу користувача в швидкості. Висновок: якщо його немає в документації Bootstrap, перегляньте свій вибір.

Надмірність в іменуванні класів є якістю CSS, а не Bootstrap. Хороші угоди про найменування класів CSS є логічними, інтуїтивно зрозумілими, лаконічними, обслуговуваними, точними та мають розшифровувані зв’язки з іменами інших класів. Але їх найважливіша якість полягає в тому, щоб об’єкт, який вони описують, виглядав і діяв так, як задумано. Бути DRY не стоїть на першому місці в списку проблем, особливо під час швидкого створення прототипів і ранніх циклів розробки продукту. Це може змусити розробників, які не живуть на передньому плані, щетинитися, коли вони бачать такі речі, як class=”btn btn-primary btn-lg”.

У глибині свого розуму вони знають, що існують способи ідентифікувати та обмежити елементи, які не потребують цього наївного стилю тегування та успадкування.

Потворна правда полягає в тому, що приблизно через одну хвилину після того, як виникає потреба писати CSS, виникає потреба в конвенції про іменування класів. Приблизно через п’ять хвилин після цього приймається рішення, що додати кілька додаткових персонажів — це невелика ціна, яку потрібно заплатити, щоб об’єкт виглядав і поводився так, як ви хочете. Звідти дуже коротко їздити до місця, де btn btn-primary btn-lg виглядає зовсім крихітним.

Bootstrap — це зріла платформа з широким поширенням. Він є частиною багатьох програм, які працюють у виробництві. Як ви, напевно, зрозуміли, я вважаю, що він ідеально підходить для розробки інтерфейсу 0-1 для веб-додатків.

2.5. Вибір Backend технологій для створення web-застосунку

2.5.1. Платформа Node.js

Node.js — це середовище виконання JavaScript з відкритим вихідним кодом і міжплатформним. Це популярний інструмент практично для будь-яких проектів! Node.js запускає JavaScript V8, ядро Google Chrome, поза браузером. Це дозволяє Node.js бути дуже продуктивним. Програма Node.js працює в одному процесі, не створюючи новий потік для кожного запиту. Node.js містить у своїй стандартній бібліотеці набір асинхронних примітивів введення-виводу, які запобігають блокування коду JavaScript, і загалом бібліотеки в Node.js написані з використанням неблокуючих парадигм, що робить поведінку блокування скоріше винятком, ніж нормою (рис. 2.5).

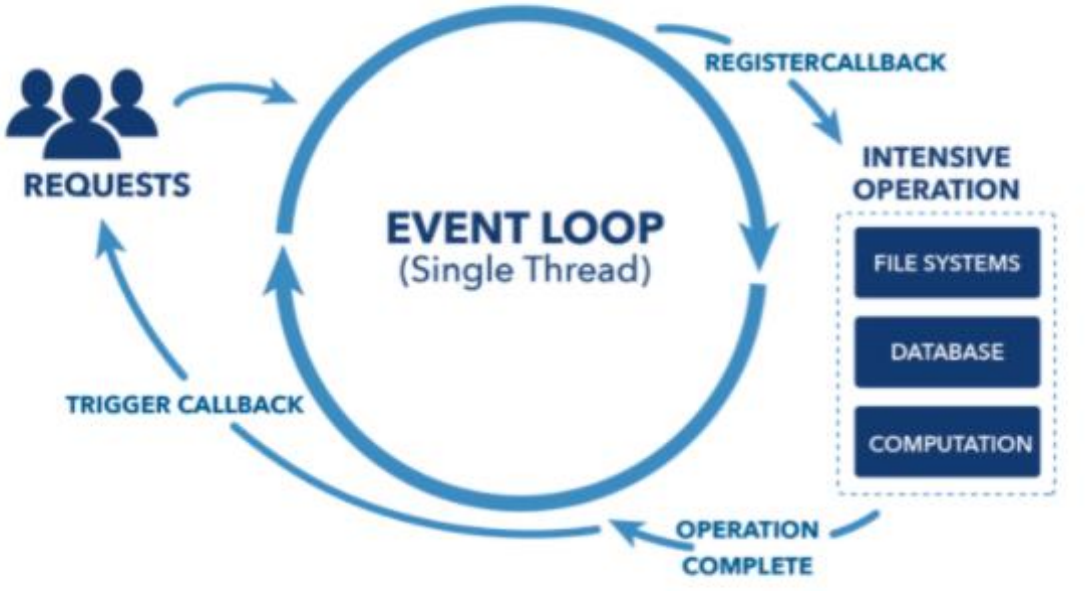

Рис. 2.5. Схема виконання Node.js

За останні роки популярність Node.js значно зросла через його надзвичайно легкість і високу гнучкість. Node.js постачається з великою бібліотекою JavaScript модулі, які спрощують процес розробки. Завдяки своїй природі з відкритим вихідним кодом, Node.js став неймовірно популярним як для розробки веб- та мобільних додатків. Останні статистичні дані показують, що:

1. Станом на початок 2020 року понад 50% розробників використовують у своїх проектах Node.js.
2. У США понад 28 000 веб-сайтів створено за технологією Node.js.
3. Великі імена, такі як eBay, AliExpress та інші, покладаються на Node.js.
4. Node.JS використовується веб-сайтами з інтенсивним трафіком, такими як Netflix, PayPal і Groupon.
5. З 2009 року, коли Node.js був представлений у світі розробників, його популярність різко зросла. У Github Node.js має 75,9 тис. зірочок, 19 тис. форків і 3 тис. спостерігачів. У Stack у нього 71,8 тисяч підписників і 8,3 тисячі голосів. Ці цифри можуть показати, наскільки

популярний Node.js. Популярні технічні гіганти, такі як Microsoft і Netflix, використовують Node.js.

Node.js посів перше місце в опитуванні розробників StackOverflow 2020 року. Більше половини респондентів в опитуванні повідомили, що використовували його у своїх проектах.

2.5.2. Фреймворк ExpressJS

Express.js — одна з найкращих фреймворків веб-додатків Node.js. Він дуже легкий і гнучкий, який має надійні функції для розробки веб-сайтів і мобільних додатків. Фреймворк Express.js забезпечує рівень фундаментальних функцій веб-додатків, не приховуючи функції Node.js. Існує безліч популярних фреймворків, заснованих на Express.js. Express — це бекенд-частина компонента, відомого як стек MEAN.

MEAN — це безкоштовний набір програмного забезпечення JavaScript з відкритим вихідним кодом для розробки інтерактивних веб-сайтів і веб-програм, який має наведені нижче аспекти:

1. MongoDB – стандартна база даних NoSQL
2. Express.js – фреймворк веб-додатків за замовчуванням

ExpressJS — це попередньо вбудований фреймворк Node.js, який може дозволити вам розробляти швидші та інтелектуальні веб-додатки на стороні сервера. Деякі з його особливостей — доступність, мінімалізм, універсальність, зручність використання, а оскільки він створений у самому NodeJS, його ефективність також була отримана від NodeJS.

Підводячи підсумок, все, що Bootstrap робить для HTML/CSS і адаптивного веб-дизайну, ExpressJS робить для NodeJS.

Це зробило програмування на Node JS простою і дало розробникам деякі додаткові можливості для розширення їх програмування на стороні сервера. ExpressJS є найпопулярнішим фреймворком Node JS на даний момент.

Давайте спочатку розглянемо переваги, які він нам пропонує:

1. Дозволяє просто та швидко розробляти веб-програму Node.js.
2. Простий і легкий у налаштуванні та налаштуванні.
3. Дозволяє визначити маршрути ваших програм на основі методів HTTP та URL-адрес.
4. Містить кілька компонентів проміжного програмного забезпечення, які можна використовувати на запит і у відповідь для виконання додаткових обов'язків.
5. Легко інтегрується з різними механізмами шаблонів, такими як Jade, Vash, EJS тощо.
6. Це допоможе визначити помилку обробки проміжного програмного забезпечення.
7. Легко обробляти статичні файли та служби програми.
8. Це дозволяє створити сервер із REST API.
9. Швидке підключення до таких баз даних, як MongoDB, Redis, MySQL.

Основні переваги вибору Express.JS:

1. Покращує масштабованість програми. Перша перевага використання Express.js для розробки бекенда полягає в тому, що ви можете легко масштабувати свою програму. Завдяки наявності Node.JS ви можете легко масштабувати свою програму в будь-якому випадку, додаючи вузли та додаючи до неї додаткові ресурси.

2. Одна й та сама мова може використовуватися як для бекенда, так і для інтерфейсу. Ще одна перевага використання Express.js полягає в тому, що за допомогою JavaScript ви можете виконувати як фронтенд, так і бекенд кодування. Незважаючи на те, що існує багато подібних фреймворків, вони забезпечують різну підтримку мови інтерфейсу та бекенда, що дуже ускладнює роботу з ними. Але з експресом. JS, ви рідко зіткнетеся з такими проблемами.

3. Express.JS підтримує функцію кешування. Функція кешування забезпечується Express.js, і перевага цього полягає в тому, що вам не доведеться

повторно виконувати коди знову і знову. Крім того, це дозволить веб-сайту завантажуватися швидше, ніж будь-коли раніше.

4. Він вимагає менших витрат на розробку та обслуговування Express.JS — це повноцінний JavaScript, який не вимагає від вас залучення численних розробників для керування інтерфейсом і бекендом веб-додатків. Таким чином, це дозволяє значно скоротити витрати на розробку та підтримку програми.

5. Підвищує ефективність і продуктивність. Раніше було зазначено, що код JavaScript інтерпретується Node.js через двигун JavaScript V8 від Google. Код JavaScript відповідає цим механізмом прямо в машинний код. Це робить ефективне застосування коду простішим і швидшим. Оскільки це полегшує неблокуючі операції введення-виводу, швидкість виконання коду також покращується середовищем виконання.

6. Має підтримку великої та активної громади. Node.js має привілей мати широку та віддану групу розробників, які продовжують вносити свій внесок у його безперервне зростання та прогрес на регулярній основі. Насправді групам розробників добре допомагають розробники JavaScript, які надають GitHub готові та прості рішення та коди. Очікується, що в майбутньому розробники ініціюють ще кілька розробників.

7. Одночасно керує запитами. Оскільки Node.js надає можливість вибору неблокуючих систем вводу-виводу, він дозволяє обробляти кілька запитів одночасно. Фреймворк може керувати ефективною обробкою одночасних запитів краще, ніж інші, наприклад Ruby або Python. Вхідні запити позиціонуються та запускаються швидко та систематично.

8. Node.js дуже розширюваний. Node.js вважається високорозширюваним, що означає, що Node.js можна змінювати та розширювати відповідно до його специфікацій. Також можна використовувати JSON, щоб запропонувати можливості для обміну даними між клієнтом і веб-сервером. Також підтримуються інтегровані API для створення серверів HTTP, TCP, DNS тощо.

2.5.3. Бібліотека Speakeasy

Двофакторна аутентифікація — це захід безпеки, реалізований у програмі для підвищення безпеки шляхом надання двох окремих доказів для авторизації в системі. Двофакторна аутентифікація (2FA) працює крім аутентифікації імені користувача/електронної пошти та пароля.

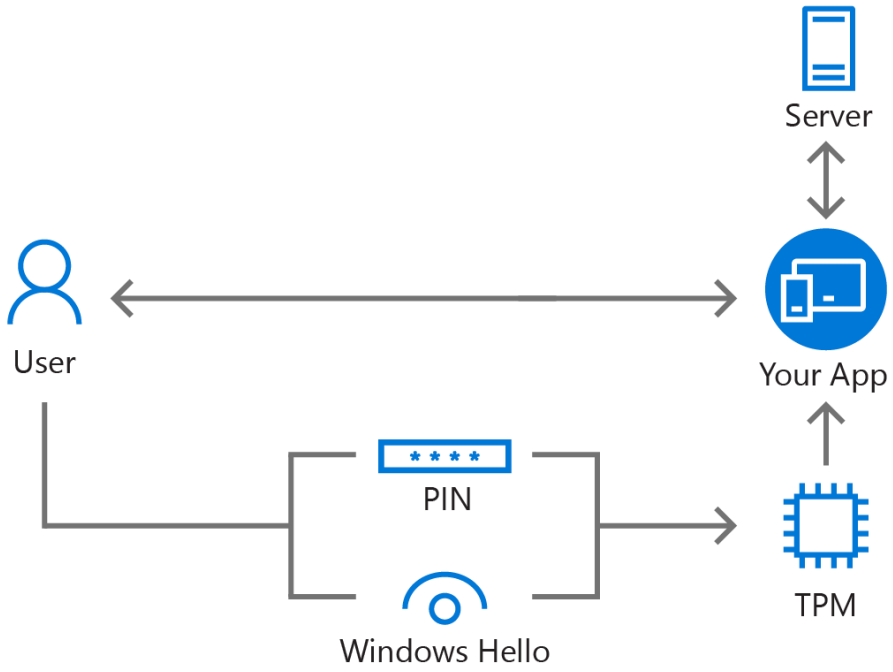

Рис. 2.6. Схема роботи двофакторної аутентифікації

Оскільки технології стали більш розвиненими та складними, традиційні методи захисту вашої особистої інформації більше неприйнятні. Це призвело до розробки альтернатив, які допоможуть забезпечити безпеку ваших даних. Однією з таких розробок є двофакторна аутентифікація, також відома як 2FA. Двофакторна аутентифікація забезпечує додатковий рівень безпеки на додаток до стандартного процесу аутентифікації. Рівень двофакторної автентифікації вимагає введення додаткових даних для доступу до облікового запису. Одним із способів реалізації 2FA є використання бібліотеки speakeasy. Бібліотека Speakeasy забезпечує двофакторну аутентифікацію за допомогою одноразового пароля (OTP). Бібліотека забезпечує додатковий рівень безпеки стандартного процесу

аутентифікації програми. Таким же чином двофакторний метод аутентифікації вимагає від вас ввести додаткові дані як докази доступу до свого облікового запису. Найпоширеніші форми двофакторної аутентифікації включають введення коду, який використовується Google і Facebook для додаткової безпеки. Використовуючи OTP, Speakeasy надає додаткові дані, необхідні для доступу до облікового запису.

Переваги двофакторної аутентифікації:

1. Додано рівень безпеки та доданий захист від кібератак.
2. Не потребує додаткових витрат під час налаштування.

Легке та зручне налаштування. Для більшості реалізацій користувачеві достатньо ввімкнути двофакторну аутентифікацію та відсканувати QR-код або ввести номер свого мобільного телефону, щоб він міг переглядати або отримувати коди аутентифікації відповідно.

2.5.4. Сервіси Google API

Google API сервіси дотримуються протоколів REST, а документація та посилання насправді досить зрозумілі. Їх дуже багато, це пов’язано з тим, що Google орієнтується на розробника під час розробки своїх продуктів.

Вам знадобиться лише обліковий запис Google і обліковий запис Google Cloud Platform: протоколи аутентифікації Google вимагають створення хмарного проекту Google. Завдяки цьому ви можете авторизувати запити API різними способами: з ключем облікового запису служби, для дій, які виконуються нелюдськими користувачами, створеними на сторінці Google Cloud IAM створивши потік Oauth2 (див. рис.2.7), який створюватиме облікові дані кінцевого користувача та дозволить нам виконувати операції так, ніби це виконує сам користувач з ключем API, простим зашифрованим рядком, який ідентифікує проект Google Cloud для цілей квот, виставлення рахунків та моніторингу.

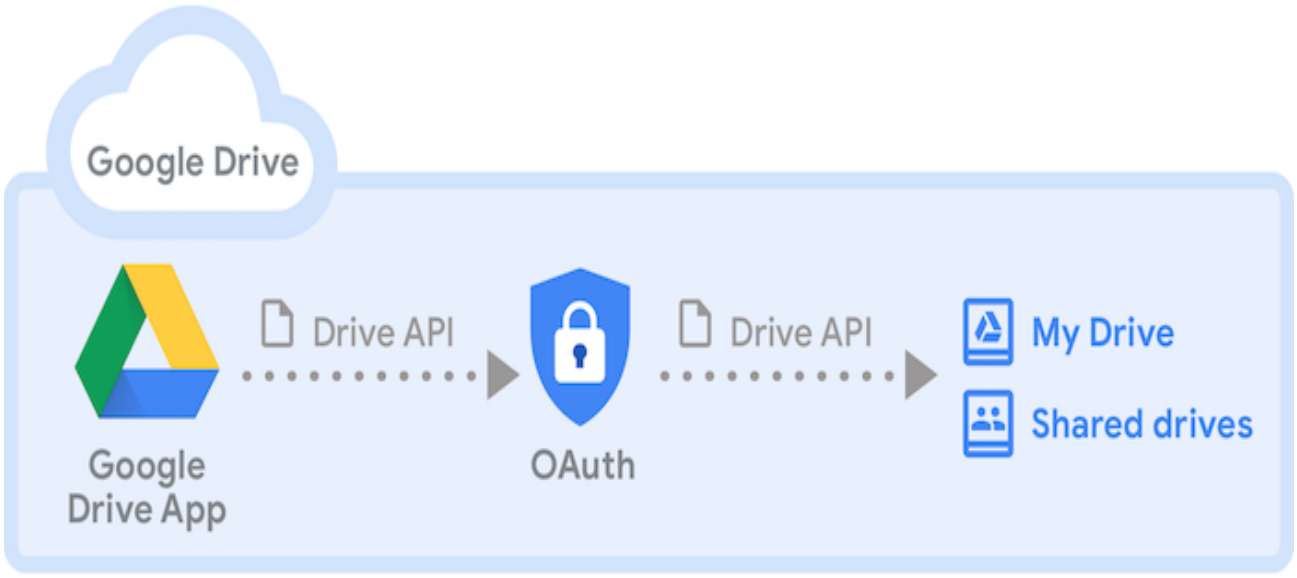

Рис.2.7. Потік Oauth2

2.5.5. Загальні відомості про Mongoose.JS

Що таке база даних NoSQL? Ви напевно чули про бази даних SQL. Ці бази даних зберігають дані, з якими можна взаємодіяти за допомогою мови запитів під назвою SQL. SQL зберігає дані в кількох таблицях, які потім можна об’єднати, щоб порівняти та об’єднати великі обсяги даних. NoSQL, з іншого боку, зберігає дані в JSON-подібних об’єктах, які називаються документами. Кожен документ схожий на рядок у базі даних SQL і складається з пар ключ-значення, як і об’єкти JavaScript.

Mongoose.js — це ODM MongoDB для Node.js. Маппер об’єктних документів (ODM) — це те, що відображає об’єкти в базі даних документів, наприклад MongoDB. Під капотом він використовує вбудований драйвер MongoDB для Node.js, але надає деякі додаткові переваги, включаючи можливість визначати схеми, застосовувати систему типів, перевіряти дані та використовувати асинхронні функції JavaScript.

Висновок до розділу 2

Для розробки односторінкового Web-застосунку було використано JavaScript бібліотекку React, стейт менеджери Redux та Thunk, бібліотека для створення інтерфейсу Bootstrap. Для створення серверної частини застосунку було використано програмну платформу Node.js, фреймворк для створення API ExpressJS, бібліотеку Speakeasy для двофакторної авторизації та бібліотеку Moments.js для роботи з датами. Для отримання й зберігання даних було обрано базу даних MongoDB та бібліотеку Mongoose. Для інтеграції з сервісами Google використано Google API.

РОЗДІЛ 3. РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ ДЛЯ ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЙНОЇ ДІЯЛЬНОСТІ ІТ-КОМПАНІЇ

3.1. Функціональна модель системи

Коли ми прагнемо побудувати модель, яку можна використовувати з програмним забезпеченням для моделювання, ми можемо почати з функціонального моделювання. Функціональна модель (або, як альтернатива, функціональна модель) — це графічне представлення системи, причому кожен будівельний блок представляє дискретну функцію разом із входами та виходами, які входять і виходять із системи та між функціями. Різні функціональні моделі можуть бути створені різними командами, які об’єднуються разом, щоб отримати загальне уявлення про систему.

Ми розглянемо, як виконувати функціональне моделювання за допомогою специфікації IDF0 інтегрованих методів визначення (IDEF), яка забезпечує простий, визначений підхід, який можна застосувати до будь-якої системи. Функціональне моделювання є центральним у більшості комплексних платформ моделювання для систем, програмних продуктів і парадигм. Сам по собі або як частина багатогранного підходу він може полегшити аналіз, відкриття та проектування. Як і всі інші роботи з моделювання та моделювання, основною передумовою функціонального моделювання є розуміння мети, точки зору та контексту:

1. Яка причина створення моделі (мета)?

2. Яку перспективу слід прийняти для призначення моделі, створення та використання (точка зору)?

3. Що таке предмет і межі/обсяг моделі (контекст)?

Ці фундаментальні питання будуть інформувати та формувати повний процес функціонального моделювання.

Специфікації моделювання, такі як класичні специфікації моделювання інтегрованих методів визначення (IDEF), забезпечують узгодженість, доступність та агностичність цим зусиллям моделювання.

Специфікація IDEF0 (рис. 3.1), зокрема, є старою, але класичною специфікацією функціонального моделювання, яка корисна для інженерів, які починають працювати з функціональним моделюванням. IDEF0 вписується в більш широкий набір специфікацій моделювання IDEF, який також охоплює інформаційне моделювання, моделювання даних і моделювання процесів. Веб-сайт IDEF пропонує їх «широкий діапазон використання, від функціонального моделювання до даних, моделювання, об'єктно-орієнтованого аналізу/проектування та отримання знань». Розробка цих специфікацій моделювання була зумовлена головним чином проектами Міністерства оборони США.

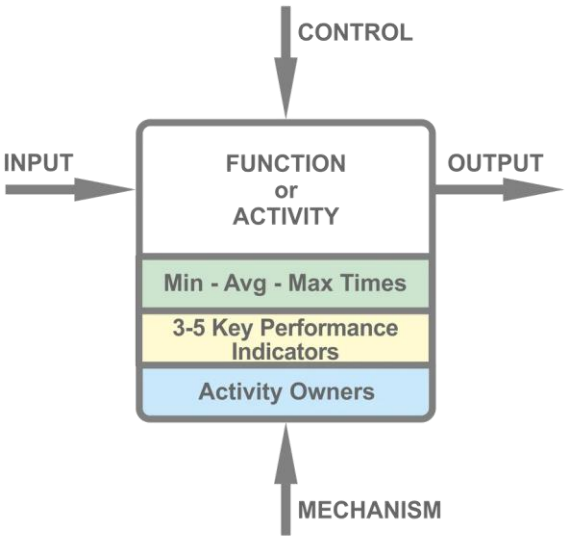

Рис. 3.1. Модель функції інтегрованого визначення методів (IDEF0)

Будівельні блоки IDEF0 — Фундаментальним будівельним блоком функціонального моделювання IDEF0 є функція з сингулярними компонентами. Функція перетворює входи у вихідні дані за допомогою елементів керування та механізмів:

1. Входи трансформуються функцією.

2. Вихідні дані створює функція.

3. Елементи керування керують функцією.

4. Механізми (засоби, за допомогою яких виконується функція).

Кожна функція IDEF0 повинна додатково включати (рис. 3.2):

1. Ім'я функції. Назви функцій мають описувати діяльність у формі дієслова та іменника, наприклад «побудувати артефакт», «проект специфікації» або «інспектувати трафік».

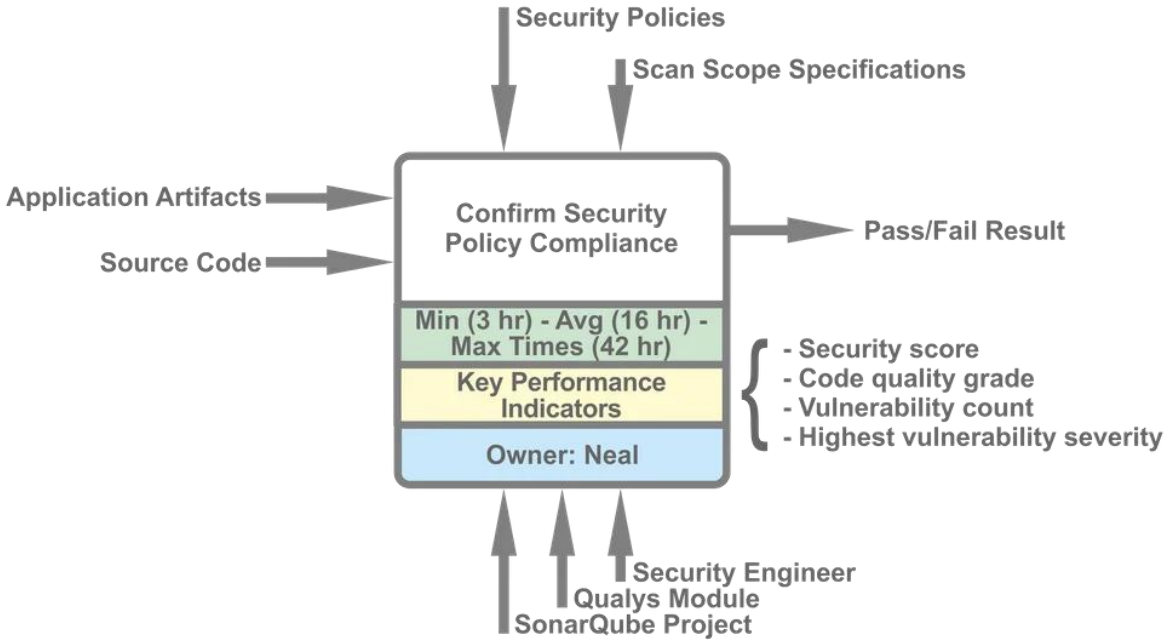

Рис. 3.2. IDEF0 Приклад однокомпонентної функції із заповненими виходами, входами, елементами керування та механізмами

2. Інформація щодо вивчення часу. До функції, якщо це можливо, слід додати інформацію з досліджень часу. Ця інформація є цінною для розуміння самостійно, але також сприяє подальшому моделюванню всієї функціональної моделі. Дані дослідження часу можуть бути простими мінімальними, максимальними та середніми або більш точними статистичними даними.

3. Ключові показники ефективності. Ключові показники ефективності стосуються інших важливих показників, крім тривалості діяльності – напр. оцінка безпеки, кількість повторних спроб або вартість діяльності.

4. Власник діяльності. Нарешті, власнику діяльності слід призначити всі функції – володіння сприяє участі, мотивації та точності.

Декомпозиція функцій IDEF0 (рис. 3.3):

1. Кожну функцію можна розкласти на 3-6 дітей. Тобто кожна функція може бути розбита на менші моделі функцій.

2. «Приємне місце» в декомпозиції функціонального моделювання – це те, де подальша декомпозиція не дасть жодної цінності і де немає надто багато компонентних моделей, щоб мати некеровану складність.

3. Мета, точка зору та контекст, визначені перед моделюванням, визначать необхідний рівень декомпозиції.

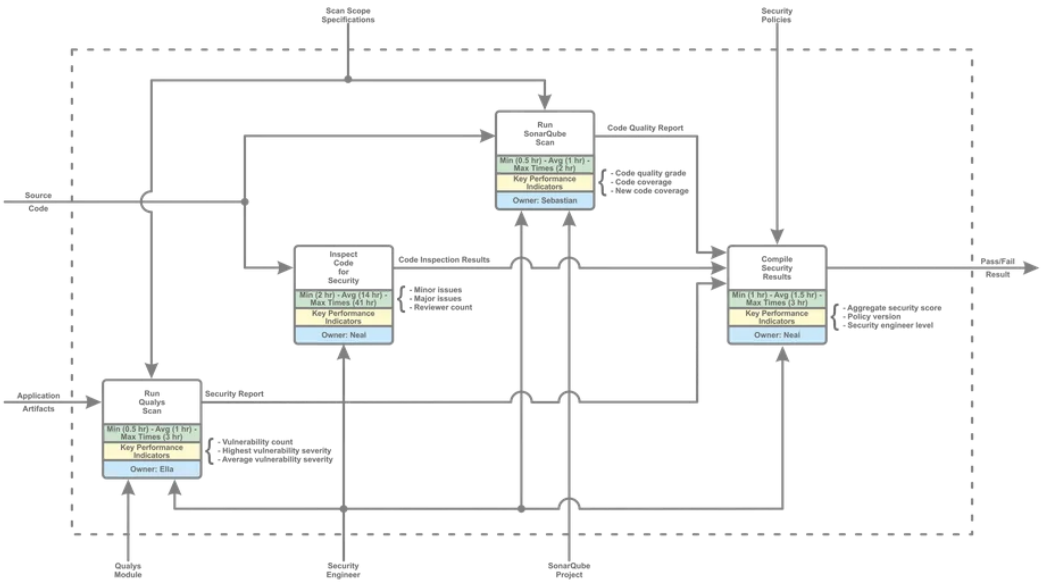

Рис. 3.3. Функція IDEF0 розбита на менші функції з різними іменами та власниками

Створюючи функціональні моделі, складні соціально-технічні системи можна аналізувати, розуміти та вдосконалювати. Ми отримуємо краще розуміння робочої системи в цілому, і модель може бути використана як вхідні дані для моделювання (якщо інструмент дозволяє), налаштовуючи змінні для вимірювання ефектів.

Процес проектування web-застосунку забезпечення комунікаційної діяльності в IT-компанії можна навести у діаграмі (рис. 3.4)

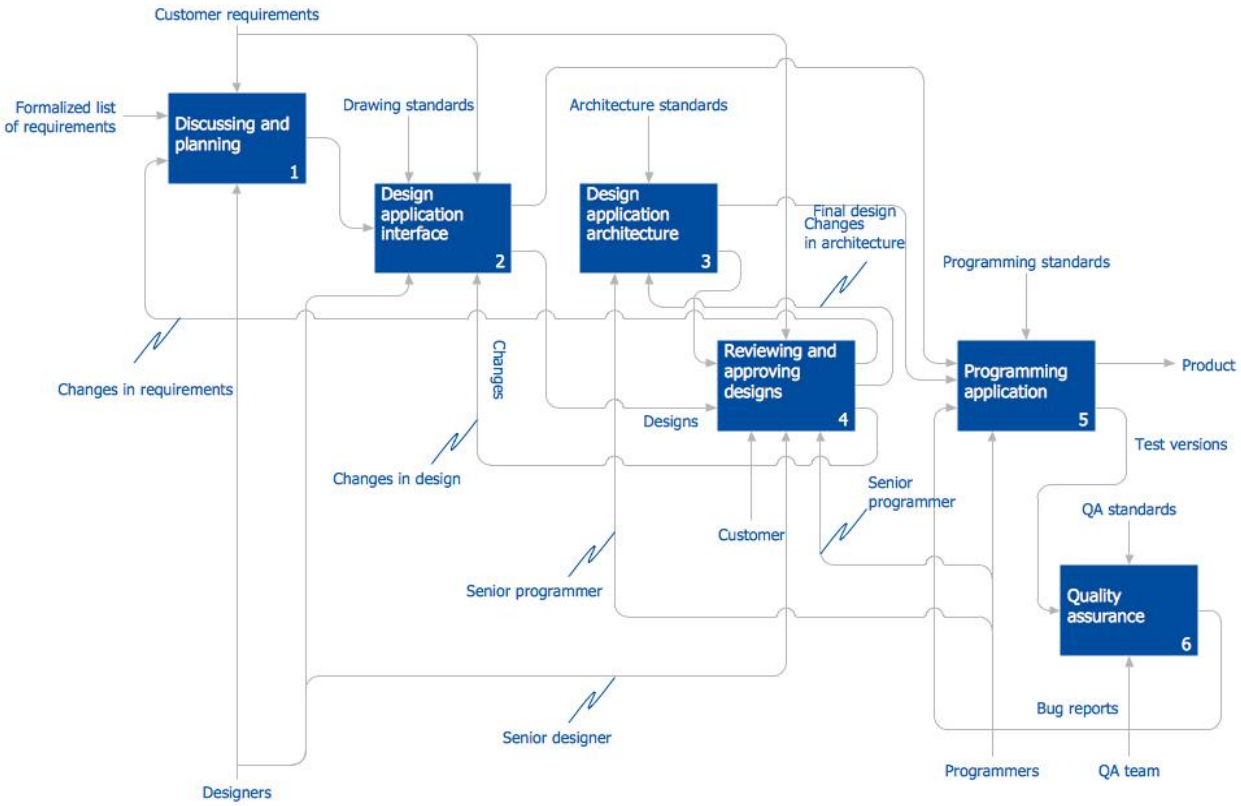

Рис. 3.4. Діаграма процесу проектування web-застосунку забезпечення комунікаційної діяльності в IT-компанії

3.2. Проектування та розробка Web-застосунку

API (рис. 3.5) – це набір програмного коду, який забезпечує передачу даних між одним програмним продуктом та іншим. Він також містить умови обміну даними.

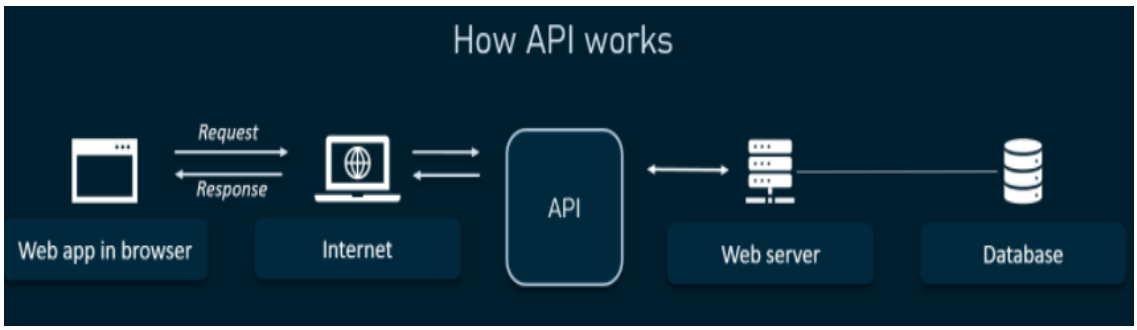

Рис. 3.5. Схема роботи API

Інтерфейси прикладного програмування складаються з двох компонентів:

1. Технічна специфікація, що описує варіанти обміну даними між рішеннями зі специфікацією, виконаною у вигляді запиту на обробку та протоколів доставки даних.

2. Інтерфейс програмного забезпечення, написаний відповідно до специфікації, яка його представляє.

Програмне забезпечення, яке має отримати доступ до інформації (тобто, Х тарифів на номери в готелі на певні дати) або функціональності (тобто маршруту від точки А до точки В на карті на основі місцезнаходження користувача) з іншого програмного забезпечення, викликає свій API, вказуючи вимоги щодо того, як дані/функціональні можливості мають надаватися. Інше програмне забезпечення повертає дані/функціональні можливості, які запитувала попередня програма.

Інтерфейс, за допомогою якого ці дві програми спілкуються, — це те, що визначає API.

Документація API — це посібник для розробників, який містить всю необхідну інформацію про те, як працювати з API та користуватися послугами, які він надає. Детальніше про документацію ми поговоримо в одному з наступних розділів. Кожен API містить і реалізується за допомогою викликів функцій – мовних операторів, які запитують програмне забезпечення для виконання певних дій і послуг. API служать багатьом цілям. Як правило, вони можуть спростити та прискорити розробку програмного забезпечення. Розробники можуть додавати функціональні можливості (наприклад, систему рекомендацій, бронювання житла, розпізнавання зображень, обробку платежів) від інших постачальників до існуючих рішень або створювати нові програми за допомогою послуг сторонніх постачальників. У всіх цих випадках фахівцям не доводиться мати справу з вихідним кодом, намагаючись зрозуміти, як працює інше рішення. Вони просто підключають своє програмне забезпечення до іншого. Іншими словами, API служать шаром абстракції між двома системами, приховуючи складність і робочі деталі останньої.

У кожного програмного сервісу є «endpoint» (), кінцеві точки які являють собою URL на які ми можемо робити запити з браузеру (див. рис. 3.6). Це і є основною характеристикою endpoint`у. Наступною характеристикою є тип запита який будемо робити, до недавніх пір було лише два запита get, коли ми бажаємо отримати інформацію, та post коли ми бажаємо її відправити (див. рис. 3.6). Request payload – це те що саме з запиту ми маємо надсилати на сервер. Також http коди, які вказують на помилки при запиті.

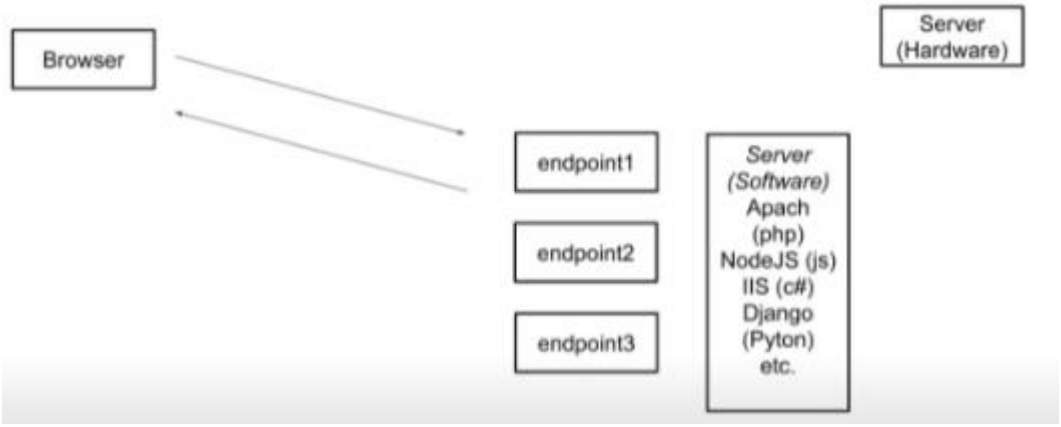

Рис. 3.6. Схема роботи Server API

Почнемо роботу зі створення моделі, для прикладу виберемо таблицю користувача. Однією з відмінностей між рідним драйвером MongoDB Node.js і Mongoose є те, що Mongoose.js вимагає створення схем/моделей перед зміною бази даних.

Робота з Mongoose. Кожна колекція повинна мати відповідну модель, пов’язану з нею. Наприклад, якщо ми зберігаємо користувачів у нашій базі даних, ми повинні створити модель користувача, щоб Mongoose знав, які поля має кожен користувач.

Створимо нову папку під назвою models. У цьому файлі створимофайл під назвою User.js. Ця модель міститиме всі поля, необхідні для створення користувача (див. рис. 3.7).

```
const { model, Schema } = require("mongoose");

const User = new Schema({
  _id: {
    type: Number,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  surname: {
    type: String,
    required: true,
  },
  lastname: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    min: 3,
    max: 40,
  },
  phone: {
    type: Number,
    required: true,
    min: 7,
  },
  working_place: {
    type: Number,
    required: true,
  },
  is_active: {
    type: Boolean,
    required: true,
  },
});

module.exports = model("user", User);
```

Рис. 3.7. Схема моделі користувача (User)

Робота з Node.js. Створимо новий файл у головному каталозі та назвемо його createUser.js. Потім імпортуємо модель, яку щойно створили (див. рис. 3.8).

Щоб створити новий документ, нам потрібно створити новий екземпляр моделі та передати дані, які ми хочемо зберегти в документі. Необхідно викликати метод .save() у новому документі, щоб переконатися, що він записаний у базу даних. Метод .save() повертає Promise, тому необхідно переконатися, що використовується async/await або .then() для запуску логіки після нього.

```
const User = require("./models/User");

const newUser = new User({
  _id: 1,
  name: "Oleksandr",
  surname: "Babin",
  lastName: "Serhiyovych",
  email: "babin.oleksandr@localdomain.com",
  phone: "+380664006080",
  working_place: 001,
  is_active: true,
});

newUser.save().then((res, err) => {
  // callback to save user to database
  // handle res and err
});
```

Рис. 3.8. Створення нового користувача (createUser.js)

Запити до бази даних. Далі ми поговоримо про різні способи доступу до даних з бази даних. Вимога для createUser.js в index.js, тому новий користувач не створюється щоразу, коли ми запускаємо код. Далі створимо новий файл query.js і імпортувати його в index.js так само, як це робили з createUser.js.

Розглянемо, як можна зробити запит до бази даних MongoDB. Метод .find() можна викликати з експортованої моделі, наприклад User. Фрагмент нижче (див. рис. 3.9) демонструє один із способів використання методу .find().

```
const User = require("./models/User");

const findAllUsers = async () => {
  const allUsers = await User.find();
  console.log(allUsers);
};

findAllUsers();
```

Рис. 3.9. Пошук по всіх користувачах (query.js)

Далі ми розглянемо редагування існуючих документів у MongoDB. Створимо новий файл під назвою update.js і імпортуємо його замість query.js.

Припустимо, що користувач у нашій базі даних змінив телефон, тому нам потрібно буде змінити поле телефону у базі даних. Наведений нижче фрагмент демонструє цей сценарій (див. рис. 3.10).

```
const User = require("./models/User");

const changePhone = async (_id, phone) => {
  const user = await User.findOne({ _id });
  if (!user) {
    throw new Error("User not found");
  }
  user.phone = phone;
  const res = await user.save();
  return res;
};

changePhone(1, "+380664008090");
```

Рис. 3.10. Редагування існуючих даних

У наступному прикладі коду показано, як можна видалити документ з MongoDB (див. рис 3.11).

```
const User = require("./models/User");

const deleteUser = async (_id) => {
  await User.deleteOne({ _id });
  const res = await User.find();

  return res;
};

deleteUser(2);
```

Рис. 3.11. Видалення користувача

Цей фрагмент показує два способи видалення документа в MongoDB. Перша функція набагато коротша і не вимагає обробки помилок, тоді як друга вимагає. Залежно від вашої програми, одна функція може підійти вам краще, ніж

інша. .deleteOne() приймає такий параметр, як .findOne() або .find(). Ви все ще можете використовувати оператори порівняння та інші оператори пошуку.Важливо зазначити, що .deleteOne() — це метод, який можна викликати лише на такій моделі, як User (зверніть увагу на використання великих літер). З іншого боку, .delete() можна викликати лише в таких документах, як змінна користувача.

Використання Express.JS для роутингу API. Необхідно створити новий JavaScript файл у папці routes. Цей файл повинен містити усі необхідні маршрути для додатку. Розглянемо базовий роут для управління об’єктом користувача (див. рис. 3.12).

```
1  import express from "express";
2  import User from "./models/User";
3  const router = express.Router();
4
5  // basic URL for this endpoint http://localhost:8080/api/users/:_id
6  router.route('/users/:_id')
7
8      // get the user
9      .get(function(req, res) {
10         // handle GET query
11     })
12
13     // update user
14     .put(function(req, res) {
15       // handle PUT query
16     })
17
18     // delete user
19     .delete(function(req, res) {
20         User.remove(
21           {
22             _id: req.params._id,
23           },
24           function (err, user) {
25             if (err) res.send(err);
26
27             res.json({ message: "Successfully deleted" });
28           }
29         );
30     });
31
```

Рис. 3.12. Маршрут API для роботи з об’єктом користувача

Шифрування RSA. Реалізуємо генерацію двох ключів, використовуючи алгоритм шифрування RSA, для того щоб зашифрувати створенні дані (та всі наступні дані, які необхідно захистити), для того, щоб лише користувач з приватним ключем зміг розшифрувати дані (див. рис. 3.13).

```
const { generateKeyPair } = require('crypto');
generateKeyPair('rsa', {
  modulusLength: 4096,
  publicKeyEncoding: {
    type: 'spki',
    format: 'pem'
  },
  privateKeyEncoding: {
    type: 'pkcs8',
    format: 'pem',
    cipher: 'aes-256-cbc',
    passphrase: 'top secret'
  }
}, (err, publicKey, privateKey) => {
  // Handle errors and use the generated key pair.
});
```

Рис. 3.13. Генерація publicKey та privateKey для шифрування

Інтеграція з Google. Розглянемо схему інтеграції користувача з сервісами Google (див. рис.3.14)

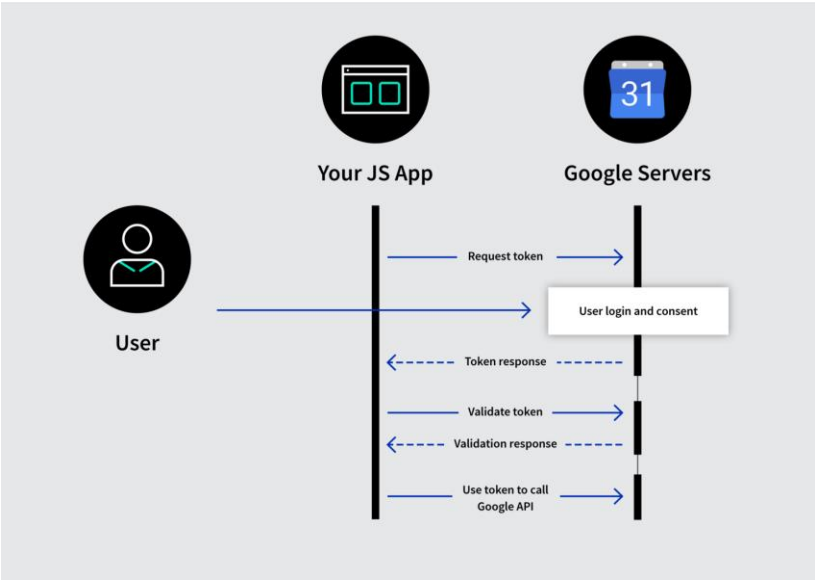

Рис. 3.14. Google API інтеграція

Реалізація Frontend. Побудуємо файлову структуру застосунку. Неправильна структура програми React вплине на масштабованість та придатність до обслуговування.

У міру зростання програми ми можемо додавати нові та видаляти деякі старі функції, тому кожен компонент має бути слабо пов’язаний один з одним. Потрібно згрупувати файли на основі функції. Тобто всі файли функції знаходяться в одній папці.Загалом, структури проекту React часто ітеративно розвиваються разом із масштабом та складністю проекту. Коли додаються нові бібліотеки, такі як Redux і React Router, початкову структуру потрібно переробити, щоб вмістити додаткову складність. Через тиск на терміни завершення проекту рефакторинг застрягає у відставанні, поки проект не стане повністю непридатним для підтримки.

Розглянемо структуру каталога, який використовується в додатках виробничого рівня, і проаналізуємо переваги та недоліки. Залежно від розміру проекту, масштабів, складності та майбутніх аспектів, найбільш підходяща структура варіюється. Вибрана структура якості відправної точки у виборі правильної початкової структури, щоб майбутній рефакторинг був мінімальним.

Розподіл станів-поглядів (view-state split) відокремлює компоненти представлення та логіки від компонента стану, але вводить додаткову структурування всередині стану (див. рис. 3.15).

Зміни у вищезгаданій структурі прості. Стан тепер вкладений ще одним рівнем, де групуються дії та редуктори певної функції програми. Завдяки цьому відразу видно, де потрібно внести зміни для певної функції. Наприклад, якщо ваш API вирішив надіслати нові поля і тепер хочемо показати їх у своїх компонентах Profile, спочатку редагуєте API, потім стан і, нарешті, оновлюєте ProfileComponent. Переваги такого підходу наступні:

1. Додавати нові функції програми та підтримувати поточні функції легко.
2. Стан добре організований, ніякої плутанини з розміщенням.

3. Весь Redux зосереджений в одному місці, тому рефакторинг є простим. Наприклад, якщо захочемо використовувати Redux Toolkit через деякий час, то потрібно внести зміни лише до файлів у каталозі стану.

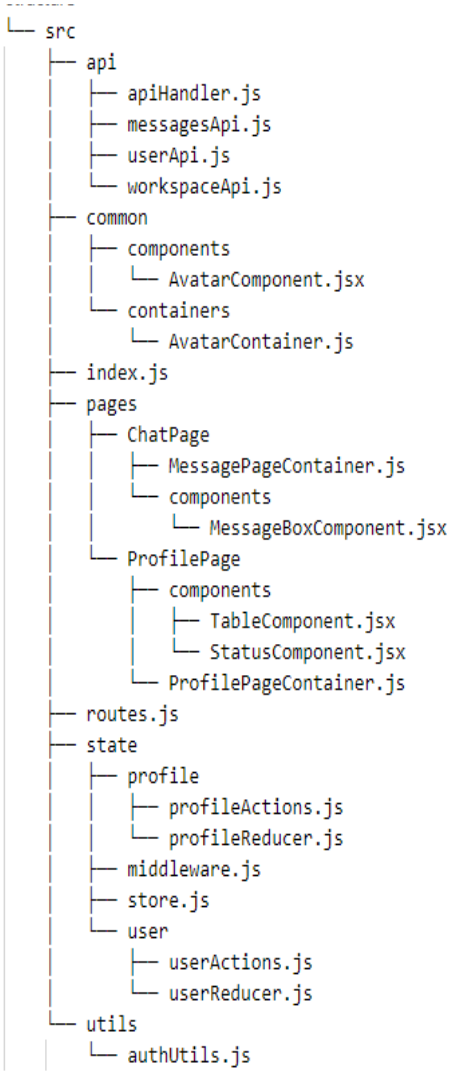

Рис. 3.15. Структура view-state split

Створимо логіку роботи на стороні Frontend’у, та почнемо із опису логіки отримання даних з API. Для прикладу я буду розглядати отримання списку користувачів. Для початку необхідно створити ActionType та UserType для нашого редюсера (див. рис. 3.16). ActionType являє собою enum та перераховує можливі сценарії отримання даних. UserType є типом, що ініціалізує типи полей.

src > types > user.ts > ...

```ts
export enum UserActionTypes {
  FETCH_USERS = "FETCH_USERS",
  FETCH_USERS_SUCCESS = "FETCH_USERS_SUCCESS",
  FETCH_USERS__ERROR = "FETCH_USERS__ERROR",
}

export type UserType = {
  users: any[];
  loading: boolean;
  error: null | string;
};

type FetchUsersAction = {
  type: UserActionTypes.FETCH_USERS;
};

type FetchUsersSuccessAction = {
  type: UserActionTypes.FETCH_USERS_SUCCESS;
  payload: any[];
};

type FetchUsersErrorAction = {
  type: UserActionTypes.FETCH_USERS__ERROR;
  payload: string;
};

export type UserAction =
  | FetchUsersAction
  | FetchUsersSuccessAction
  | FetchUsersErrorAction;
```

Рис. 3.16. ActionType та UserType списку користувачів

Далі опишемо редюсер, який приймає по замовчуванням initialState, який залежить від типу Action'а (див. рис. 3.17).

src > store > reducers > userReducer.ts > ...

```ts
import { UserActionTypes, UserType, UserAction } from "../../types/user";

const initialState: UserType = {
  users: [],
  loading: false,
  error: null,
};

export const userReducer = (
  state = initialState,
  action: UserAction
): UserType => {
  switch (action.type) {
    case UserActionTypes.FETCH_USERS:
      return { users: [], loading: true, error: null };

    case UserActionTypes.FETCH_USERS_SUCCESS:
      return { users: action.payload, loading: false, error: null };

    case UserActionTypes.FETCH_USERS__ERROR:
      return { users: [], loading: false, error: action.payload };

    default:
      return state;
  }
};
```

Рис. 3.17. UserReduces

Далі необхідно описати Action. У функцію dispatch предаємо необхідний ActionType, та, якщо необхідно, відповідь на запит до сервера або повідомлення про помилку (див. рис.3.18).

src > store > action-creators > user.ts > ...

```ts
import axios from "axios";
import { Dispatch } from "redux";
import { APIGateway } from "../../config";
import { UserAction, UserActionTypes } from "../../types/user";


export const fetchUsers = () => {
  return async (dispatch: Dispatch<UserAction>) => {
    try {
      dispatch({ type: UserActionTypes.FETCH_USERS });
      const res = await axios.get(APIGateway);
      dispatch({
        type: UserActionTypes.FETCH_USERS_SUCCESS,
        payload: res.data,
      });
    } catch (error) {
      dispatch({
        type: UserActionTypes.FETCH_USERS__ERROR,
        payload: "Something went wrong",
      });
    }
  };
};

```

Рис. 3.18. Action-creator для користувача

Засобами React та використовуючи Bootstrap для UI елементів отримуємо список користувачів, та рендеримо (див. рис. 3.19).

Весь функціонал web-додатку є досить шаблонним, та прямо реалізує архітектуру та приклади, які приведені вище.

3.3. Інтерфейс застосунку

Розглянемо реалізований web-додаток. Почнемо зі сторінки авторизації (рис. 3.20). Основна форма дозволяє використати корпоративну пошту та пароль, також є поле для двофакторної авторизації. Також, існує можливість логіну напряму через корпоративну пошту, що використовує Google сервіси.

```tsx
import React, { useEffect } from "react";
import { useActions } from "../hooks/useActions";
import { useTypesSelector } from "../hooks/useTypesSelector";
import UserItem from "./UserItem";

export const UserList: React.FC = () => {
  const { users, loading, error } = useTypesSelector((state) => state.user);
  const { fetchUsers } = useActions();

  useEffect(() => {
    fetchUsers();
  }, []);

  if (loading) {
    return <h1 className="loading-header">Loading... </h1>;
  }

  if (error) {
    return <h1 className="error-response"> {error} </h1>;
  }

  return (
    <div>
      {users.map((id, user) => (
        <UserItem id={id} user={user} />
      ))}
    </div>
  );
};

export default UserList;
```

Рис. 3.19. React шаблон для рендеру списка користувачів

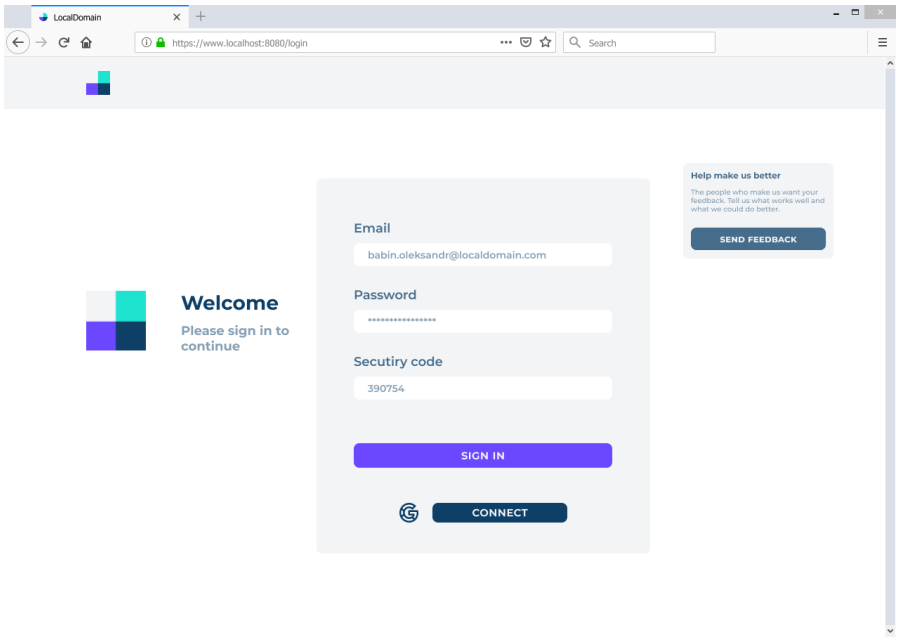

Рис. 3.20. Сторінка авторизації

Розглянемо сторінку, яку користувач бачить відразу після авторизації, це сторінка воркплейсу (англ. Workspace) (див. рис. 3.21). Ми можемо побачити

список груп (англ. Groups), контактів (англ. People), нові пости та повідомлення у лівій частині екрану для навігації. Також, модуль для створення нового посту, с можливістю додавання івенту, файлів та повідомлень. Внизу список постів, які йдуть від груп або контактів. Пости мають можливість коментарів та реакцій.

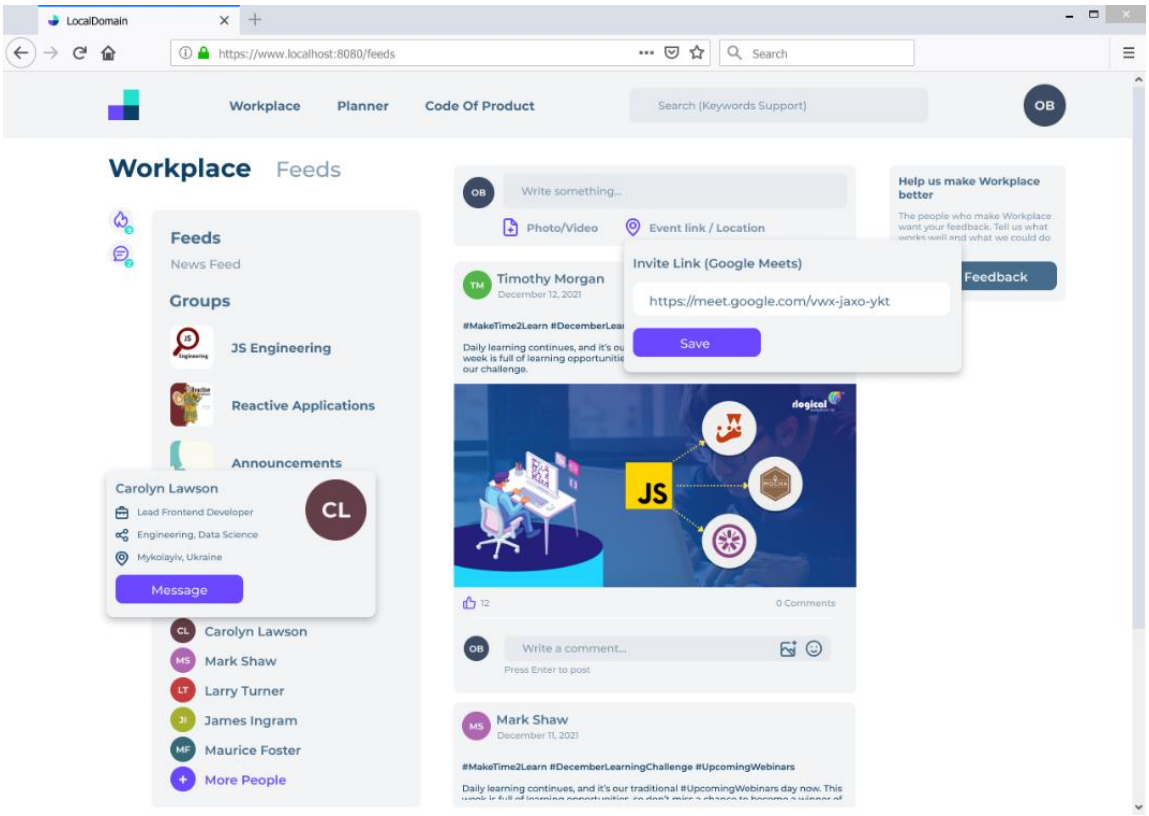

Рис. 3.21. Сторінка воркспейсу (Workspace)

Розглянемо сторінку повідомлень (чату). Поки канал повідомлення не вибраний, то інтерфейс має плейсхолдери у центрі та справа екрану (рис. 3.22). У лівій частині екрану знаходяться чати, які має користувач. Нові повідомлення відмічені значком.

Далі розглянемо кейс, коли вибраний чат (див. рис. 3.23). У центрі екрану можна переглянути повідомлення. Внизу поле для відправлення нового повідомлення, підтримуються файли, документи, текст та посилання. Справа можна побачити інформацію про адресата конкретного чату, а саме персональні дані, позицію у компанії та локацію офісу, роботи, тощо.

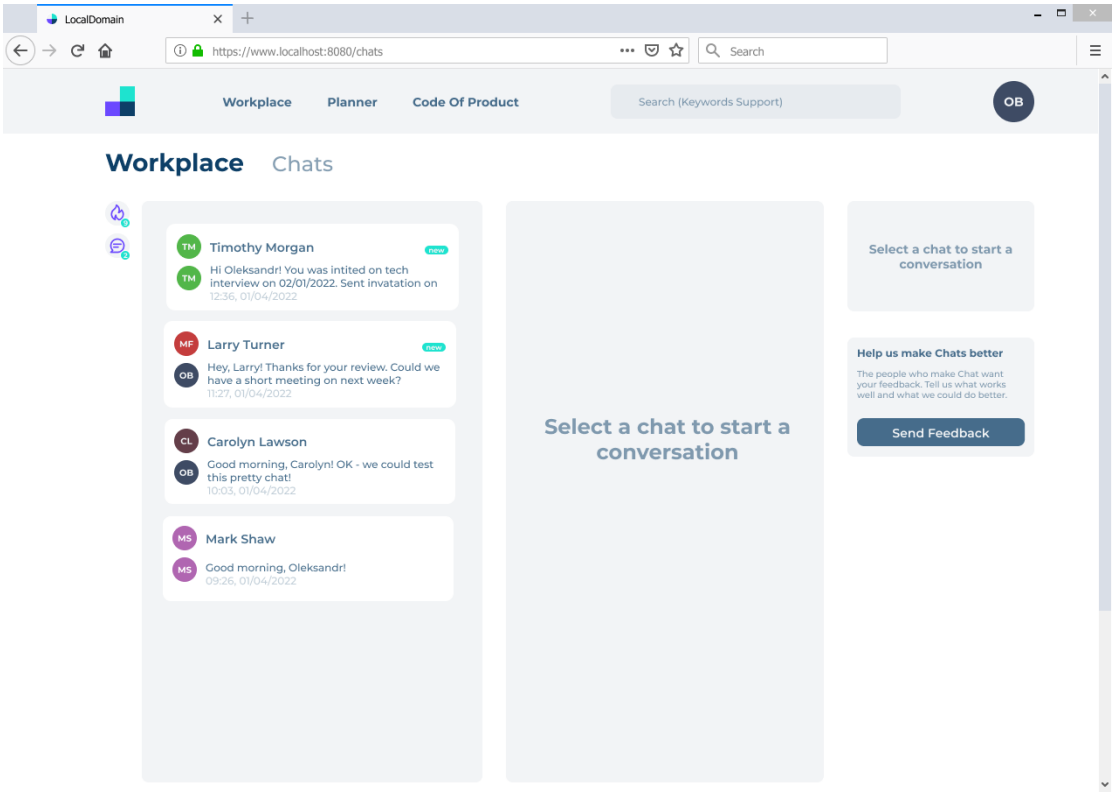

Рис. 3.22. Сторінка чату, доки не вибраний канал повідомлень.

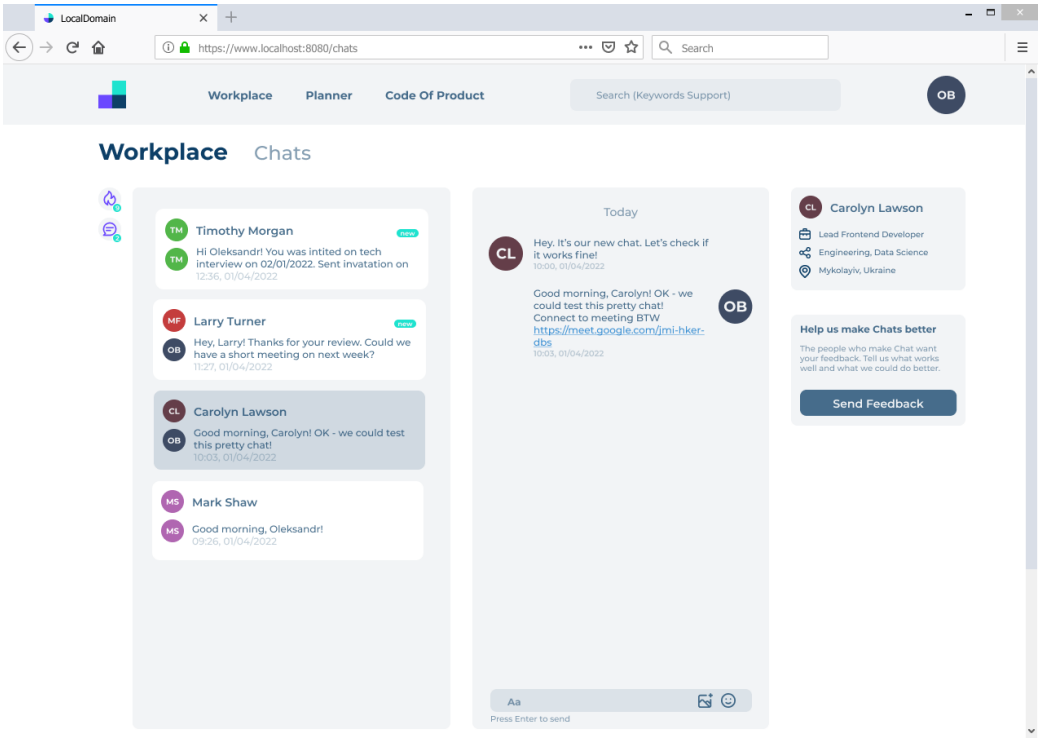

Рис. 3.23. Сторінка чату

Розглянемо сторінку профіля користувача (див. рис. 3.24). Основний функціонал, який представлений:

1. Аватар користувача.
2. Перегляд полей, що включають повне ім’я, назву команди, офіс, позицію, персональні дані, загальну інформацію щодо технологій (Executive Summary), інформації щодо іноземних мов та рівня підготовки, інформацію про вищу освіту та років роботи.
3. Інформацію щодо проекту: назва проекту, Lead та HR/Manager, з можливістю переглянути акаунти, коротку інформацію та відправити повідомлення.
4. Можливість експортувати профіль у необхідний формат (.pdf або .docx).
5. Індикатор статусу акаунту.
6. Можливість редагування власної інформації (див. рис. 3.25).

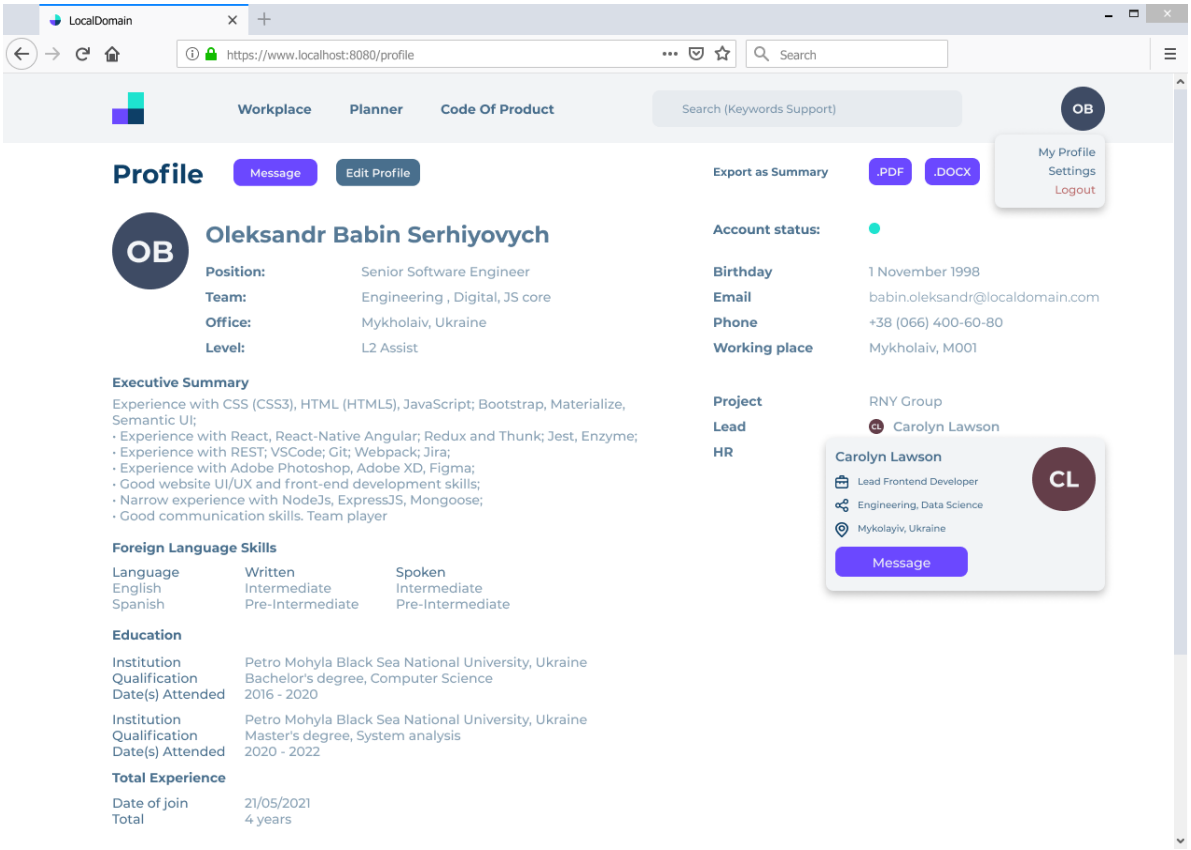

Рис. 3.24. Сторінка профілю користувача

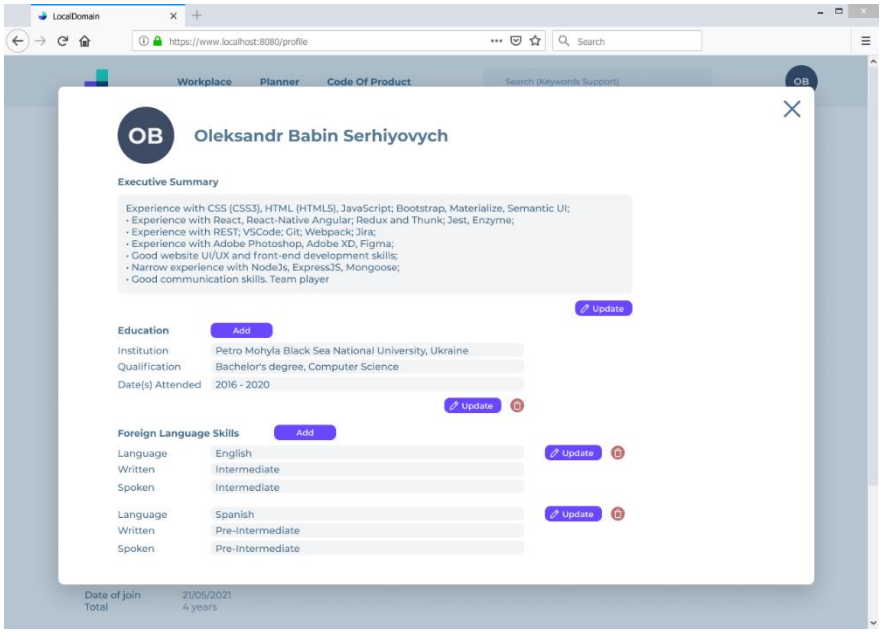

Рис. 3.25. Редагування персональних даних

Розглянемо також сторінку пошуку (див. рис. 3.26), яка включає в собі глубокий пошук не лише по імені або заголовкам, але й по опису (Description, Summary). По введеному значенню можна знайти працівника, групу або проект. Також, можна одразу експортувати профіль створивши CV (резюме).

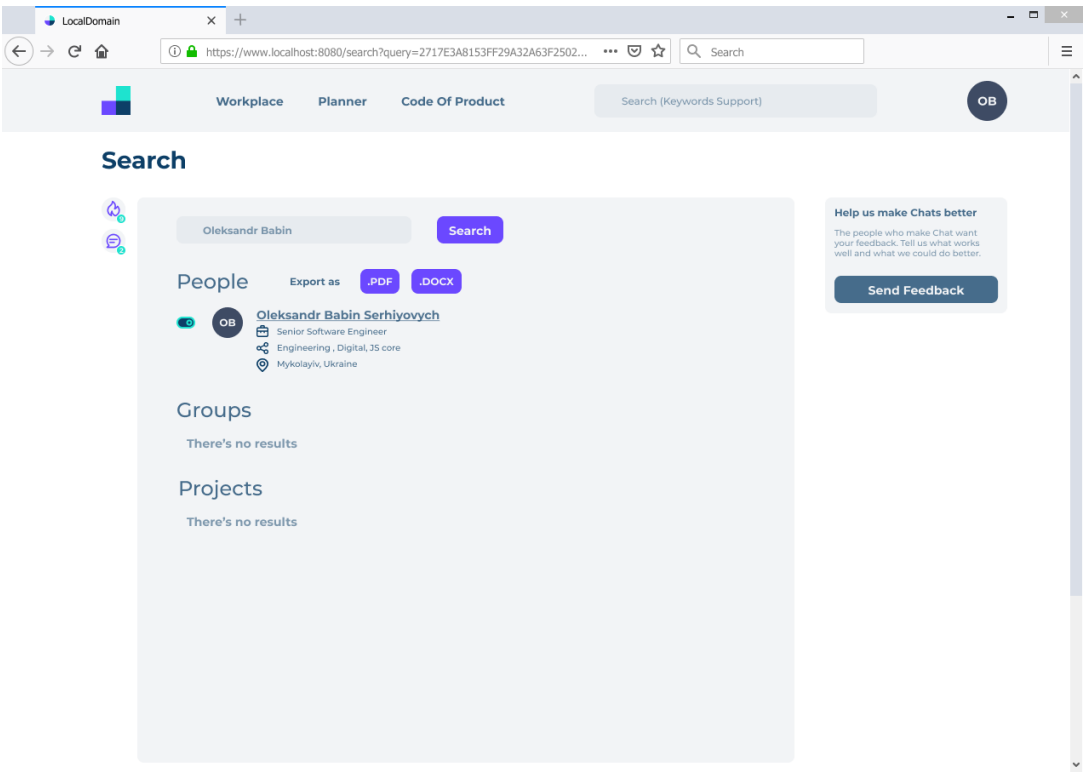

Рис. 3.26. Сторінка пошуку

Висновок до розділу 3

Здійснено розробку, програмну реалізацію та тестування Web-застосунку забезпечення комунікаційної діяльності в IT-компаніях із надійною системою шифрування внутрішньо корпоративної ділової інформації, який підтримує закрите спілкування та обмін діловою інформацією, допомагає усунути вузькі місця у проєкті, забезпечує безперебійну комунікацію з членами команди розробників.

Розроблений Web-застосунок надає можливість для проведення особистих онлайн зустрічей та сумісної роботи віддалено з інструментарієм для переписки, обміну файлами та управління проектами. Застосунок містить чат і соціальну мережу для підтримки спілкування всередині команди розробників, має інструментальні засоби для створення корпоративних каналів, які забезпечують спілкування фахівців під час роботи над проектами, дозволяє формувати та переглядати профайли розробників і створювати на їх основі резюме. Інтеграція з хмарним сервісом Google Calendar надає можливість створювати списки задач із вказівкою термінів їх виконання та відповідальними співробітниками, бот-канали для нагадувань та оповіщень співробітників, інтерактивні опитування, голосування, здійснювати розширений пошук по історії повідомлень. Застосування надійної системи шифрування надає принципово новий підхід до спілкування, дозволяє організувати закритий корпоративний зв’язок та підвищує надійність і безпеку роботи команди розробників IT-компанії, об’єднуючи їх у згуртований колектив.

ВИСНОВКИ

У результаті проведеного дослідження виявлено, що до ключових функцій, які потрібно автоматизувати для підтримки комунікації в IT-компанії, необхідно віднести кадровий облік: автоматизацію всіх процесів роботи з персоналом – від кадрового адміністрування до завдань планування і управління. Сучасні кадровики (PP – співробітники) є HR-фахівцями (HR – людські ресурси, англ. Human resources) й займаються розвитком корпоративної культури, оцінкою ефефктивності співробітників, визначенням вимог до майбутніх співробітників, веденням бази даних співробітників та кандидатів, організацією опитувань, проведенням зустрічей 1:1 і визначенням продуктивності роботи, оцінки співробітників (англ. Performance review).

Використання Інтернет-технологій для розподіленого у просторі спілкування обумовило різке зростання кількості програмних засобів, які спрямовані на ефективне забезпечення комунікаційної діяльності організацій: це корпоративні месенджери Телеграм, Slack, Flock, Skype, Google Meet, Samepage, хмарний сервіс Highfive, цифрове робоче місце, яке об’єднує розрізнені кадри Beekeeper. Їх використання спрощує спілкування між колегами, дозволяє відправляти голосові та текстові повідомлення, забезпечує відеозв’язок, формує канали, які є робочим простором, дозволяє підтримувати та координувати різні графіки діяльності, що сприяє співробітництву та швидкому прийняттю рішень. Однак корпоративні месенджери приділяють недостаню увагу питанням забезпечення безпеки. Єдиного рішення, яке змогло покрити всі потреби аутсортингового розробки програмного забезпечення компаній та мало необхідний функціонал для автоматизації управління й оцінки персмоналу з надійним захистом корпоративної інформації, немає.

Установлено, що для забезпечення надійності захисту використовують часткове або повне шифрування повідомлень та файлів, самознищення повідомлень і цілих чатів, блокування можливостей для скріншотів. Різко знижує витік корпоративної інформації застосування багатофакторної автентифікації.

Наскрізне шифрування (англ. end-to-end encryption), при якому доступ до інформації мають тільки ті користувачі, які приймають участь у спілкуванні, не дозволяє отримати доступ до криптографічних ключів третім особам, є найбільш надійним способом захисту даних. Для обміну ключами при цьому можуть бути застосовані симетричний та асиметричний алгоритми. Симетричне шифрування використовує єдиний ключ, який використовується обома сторонами, що спілкуються, для шифрування та дешифрування. Асиметричне шифрування використовує два ключі, один з яких є приватним, а другий – відкритим. Виявлено найнебезпечніші алгоритми серед найбільш часто використовуваних алгоритмів шифрування: AES, Triple DES, Blowfish, Twofish, RSA.

Застосовування комбінації алгоритму RSA з китайською теоремою про залишки CRT дозволяє суттєво прискорити швидкість шифрування/дешифрування повідомлень і передачу інформації та забезпечує стиснення великих даних. Китайська теорема про залишки розвантажує RSA від піднесення у степінь з дуже великим показником, зменшуючи його розрядність вдвічі. Таким чином застосування криптографічного алгоритму RSA-CRT дозволяє збільшити швидкість алгоритму на практиці втричі та покращити безпеку даних за допомогою двох, а не одного ключового фактора.

Для розробки односторінкового Web-застосунку було використано JavaScript бібліотекку React, стейт менеджери Redux та Thunk, бібліотека для створення інтерфейсу Bootstrap. Для створення серверної частини застосунку було використано програмну платформу Node.js, фреймворк для створення API ExpressJS, бібліотеку Speakeasy для двофакторної авторизації та бібліотеку Moments.js для роботи з датами. Для отримання й зберігання даних було обрано базу даних MongoDB та бібліотеку Mongoose. Для інтеграції з сервісами Google використано Google API.

Здійснено розробку, програмну реалізацію та тестування Web-застосунку забезпечення комунікаційної діяльності в IT-компаніях із надійною системою шифрування внутрішньо корпоративної ділової інформації, який підтримує

закрите спілкування та обмін діловою інформацією, допомагає усунути вузькі місця у проєкті, забезпечує безперебійну комунікацію з членами команди розробників.

Розроблений Web-застосунок надає можливість для проведення особистих онлайн зустрічей та сумісної роботи віддалено з інструментарієм для переписки, обміну файлами та управління проектами. Застосунок містить чат і соціальну мережу для підтримки спілкування всередині команди розробників, має інструментальні засоби для створення корпоративних каналів, які забезпечують спілкування фахівців під час роботи над проектами, дозволяє формувати та переглядати профайли розробників і створювати на їх основі резюме. Інтеграція з хмарним сервісом Google Calendar надає можливість створювати списки задач із вказівкою термінів їх виконання та відповідальними співробітниками, бот-канали для нагадувань та оповіщень співробітників, інтерактивні опитування, голосування, здійснювати розширений пошук по історії повідомлень. Застосування надійної системи шифрування надає принципово новий підхід до спілкування, дозволяє організувати закритий корпоративний зв’язок та підвищує надійність і безпеку роботи команди розробників IT-компанії, об’єднуючи їх у згуртований колектив.

Поставлені завдання виконано повністю, однак дослідження виявило ряд проблем, які потребують подальшого дослідження: допрацювати функції чату, розширити список підтримуваних файлів та інтегрувати можливість онлайн перегляду файлів, додати можливість працівників деялких ролей, таких як HR або менеджерам, редагувати певні поля працівників, можливість деактивувати аккаунт та заповнювати загальну інформацію. Перейти із HTTP запитів на SignalR бібліотеку, що дозволить отримувати сповіщення без оновлення сторінки у реальному часі. Це стосується функцій чату та повідомлень.

Також, необхідно покрити код Unit тестами, щоб не мати проблем з розширенням додатку. У середньому необхідно мати покриття (з англ. Coverage) тестами не нижче 80 відсотків коду.

СПИСОК ПОСИЛАНЬ

1. IT skills: Why communication is key in 2021. URL: https://enterprisersproject.com/article/2021/2/it-skills-communication-key-2021 (дата звернення: 14.01.2022).

2. Ethical issues and public communication in the development of cell-based treatments for COVID-19. URL: https://www.sciencedirect.com/science/article/pii/S2213671121004811 (дата звернення: 14.01.2022).

3. Створення успішної корпоративної соціальної мережі (ESN). URL: https://www.atlassian.com/ru/work-management/social-intranet/enterprise-social-network (дата звернення: 14.01.2022).

4. Сучасні IT-рішення для управління бізнесом. URL: https://www.it.ua/ru (дата звернення: 14.01.2022).

5. Тенденції працевлаштування на IT-ринку. URL: http://www.management.com.ua/tend/tend648.html (дата звернення: 14.01.2022).

6. Сучасний JavaScript. URL: https://www.oreilly.com/library/view/modern-javascript/9781492023548/ (дата звернення: 15.01.2021).

7. JavaScript для веб-дизайнерів. URL: http://backspaces.net/temp/Ebooks/JavaScript-for-Web-Designers/javascript-for-web-designers.pdf (дата звернення: 15.01.2021).

8. JavaScript for Web. URL: http://backspaces.net/temp/Ebooks/JavaScript-for-Web-Designers/javascript-for-web-designers.pdf (дата звернення: 16.01.2021).

9. Web Security. URL: https://www.oreilly.com/library/view/web-security-2016/9781940111452/ (дата звернення: 16.01.2021).

10. Що таке SPA-додатки. URL: https://wezom.com.ua/blog/chto-takoe-spa-prilozheniya (дата звернення: 16.01.2022).

11. Чим насправді має займатися HR. URL: https://mc.today/popytki-hr-

menedzherov-stroit-iz-sebya-psihologov-smeshny-chem-na-samom-dele-dolzhen-zanimatsya-hr/ (дата звернення: 17.01.2022).

12. Redux Thunk vs Redux Saga. URL: https://www.eternussolutions.com/2020/12/21/redux-thunk-redux-saga/ (дата звернення: 17.01.2022).
13. State Management Battle in React 2021: Hooks, Redux, and Recoil. URL: https://dev.to/workshub/state-management-battle-in-react-2021-hooks-redux-and-recoil-2am0 (дата звернення: 17.01.2022).
14. Startups can still feel good about choosing Bootstrap in 2021. URL: https://dev.to/appmapruby/startups-can-still-feel-good-about-choosing-bootstrap-in-2021-15el (дата звернення: 17.01.2022).
15. What Are the Reasons to Learn Express.Js In 2021? URL: https://codersera.com/blog/learn-express-js/ (дата звернення: 17.01.2022).
16. Top trends in Node.js to Watch in 2021. URL: https://medium.com/selleo/top-trends-in-node-js-to-watch-in-2021-d94ff38cc31e (дата звернення: 17.01.2022).
17. Building a REST API with Node and Express. URL: https://stackabuse.com/building-a-rest-api-with-node-and-express/ (дата звернення: 17.01.2022).
18. Organizing your Express.js project structure for better productivity. URL: https://blog.logrocket.com/organizing-express-js-project-structure-better-productivity/ (дата звернення: 18.01.2022).
19. How To Define Routes and HTTP Request Methods in Express. URL: https://www.digitalocean.com/community/tutorials/nodejs-express-routing (дата звернення: 18.01.2022).
20. How to use method functionin Schema. URL: https://www.tabnine.com/code/javascript/functions/mongoose/Schema/method (дата звернення: 18.01.2022).
21. What are the Advantages and Disadvantages of Angular? URL: https://www.edureka.co/blog/advantages-and-disadvantages-of-

angular/#AdvantagesDisadvantages (дата звернення: 18.01.2022).

22. The Good and the Bad of Vue.js Framework Programming. URL: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/ (дата звернення: 18.01.2022).
23. Pros and Cons of ReactJS. URL: https://www.javatpoint.com/pros-and-cons-of-react (дата звернення: 18.01.2022).
24. A New Approach to React Component Design. URL: https://www.freecodecamp.org/news/a-new-approach-to-react-component-design-2bf76a87add1/ (дата звернення: 18.01.2022).
25. How to consume a RESTful API in React. URL: https://pusher.com/tutorials/consume-restful-api-react/ (дата звернення: 18.01.2022).
26. What Is the Best Way to Style React Components? 4 Most Widely-Used Approaches to Styling. URL: https://medium.com/@OPTASY.com/what-is-the-best-way-to-style-react-components-4-most-widely-used-approaches-to-styling-938a1e9f1c4c (дата звернення: 18.01.2022).
27. React Apps: Approaching Organization / Structure / Architecture. URL: https://medium.com/maintainable-react-apps/react-apps-approaching-organization-structure-architecture-49a281bd97eb (дата звернення: 19.01.2022).
28. React Clean Code - Simple ways to write better and cleaner code. URL: https://dev.to/thawkin3/react-clean-code-simple-ways-to-write-better-and-cleaner-code-2loa (дата звернення: 19.01.2022).
29. Dependency Injection /Services pattern for React (inspired by Angular). URL: https://dev.to/dansolhan/simple-dependency-injection-functionality-for-react-518j (дата звернення: 19.01.2022).
30. Approaches to Information Security Implementation. URL: https://blog.box.com/approaches-information-security-implementation (дата звернення: 19.01.2022).
31. RSA - theory and implementation. URL: https://eli.thegreenplace.net/2019/rsa-

theory-and-implementation/ (дата звернення: 19.01.2022).

32. RSA Variant with Improved Computational Performance and Memory Usage. URL: https://www.eurekaselect.com/article/88153 (дата звернення: 19.01.2022).
33. RSA. URL: https://wreferat.baza-referat.ru/RSA (дата звернення: 19.01.2022).
34. Performance-enhancing of RSA public key via three-dimensional hyperchaotic system. URL: https://aip.scitation.org/doi/abs/10.1063/5.0040397 (дата звернення: 19.01.2022).
35. A guide to common types of two-factor authentication. URL: https://venturebeat.com/2017/09/24/a-guide-to-common-types-of-two-factor-authentication/ (дата звернення: 20.01.2022).
36. Google API Design Guide. URL: http://apistylebook.com/design/guidelines/google-api-design-guide (дата звернення: 20.01.2022).
37. Menezes, P. van Oorschot, S. Vanstone. Handbook of Applied Cryptography - www.cacr.math.uwaterloo.ca/hac/. - CRC-Press, 1996. - 816 p. - (Discrete Mathematics and Its Applications). - ISBN 0-8493-8523-7.
38. Венбо Мао Сучасна криптографія. Теорія та практика - Modern Cryptography: Theory and Practice. - М: Вільямс, 2005. - 768 с. - 2 000 прим. - ISBN 5-8459-0847-7, ISBN 0-13-066943-1.
39. Нільс Фергюсон, Брюс Шнайєр Практична криптографія = Practical Cryptography: Designing and Implementing Secure Cryptographic Systems. - М.: "Діалектика", 2004. - 432 с. - 3000 прим. - ISBN 5-8459-0733-0, ISBN 0-4712-2357-3.
40. Шнайєр Б. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти мовою Сі - www.ssl.stu.neva.ru/psw/crypto/appl_ukr/appl_cryp.htm = Applied Cryptography. Protocols, Algorithms and Source Code in C. - М.: Тріумф, 2002. - 816 с. - 3000 прим. - ISBN 5-89392-055-4.
41. Мінімальні вимоги щодо безпеки та охорони здоров’я працівників у робочих зонах. URL: https://medoc.ua/blog/minimalni-vimogi-shhodo-bezpeki-ta-ohoroni-zdorovja-pracivnikiv-u-robochih-zonah-proekt (дата звернення: 23.01.2022).

42. Вимоги безпеки щодо організації робочих місць. URL: https://buklib.net/books/31185/ (дата звернення: 23.01.2022).
43. Охорона праці в офісі. Вимоги до робочого місця офісного працівника. URL: https://gc.ua/uk/oxorona-praci-v-ofisi-vimogi-do-robochogo-miscya-ofisnogo-pracivnika/ (дата звернення: 23.01.2022).
44. ЗАКОН УКРАЇНИ. Про охорону праці. Відомості Верховної Ради України (ВВР), 1992, № 49, ст.668. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 23.01.2022).
45. Навчання та перевірка знань з питань охорони праці. URL: https://oppb.com.ua/news/navchannya-ta-perevirka-znan-z-pytan-ohorony-praci (дата звернення: 23.01.2022).

Додаток А

Лістинг Web-застосунку

userReducer.ts

```
import { UserAction, UserActionTypes, UserType } from "../../types/user";

const initialState: UserType = {
  users: [],
  user: null,
  loading: false,
  loadingList: false,
  fetchError: null,
  removeError: null,
  editShow: false,
  editError: null,
  editLoading: false,
};

export const userReducer = (
  state: UserType = initialState,
  action: UserAction
): UserType => {
  switch (action.type) {
    case UserActionTypes.FETCH_USERS:
      return { ...state, loading: true };

    case UserActionTypes.FETCH_USERS_SUCCESS:
      return { ...state, loading: false, users: action.payload };

    case UserActionTypes.FETCH_USERS_ERROR:
      return { ...state, loading: false, fetchError: action.payload };

    case UserActionTypes.REMOVE_USER:
      return { ...state, loadingList: true };

    case UserActionTypes.REMOVE_USER_SUCCESS:
      return {
        ...state,
        users: state.users.filter((user) => user.id !== action.payload),
        loadingList: false,
      };

    case UserActionTypes.REMOVE_USER_ERROR:
      return { ...state, loadingList: false, removeError: action.payload };

    case UserActionTypes.REMOVE_USER_ERROR_CLEAR:
```

```
      return { ...state, removeError: null };

    case UserActionTypes.EDIT_USER:
      return { ...state, editLoading: true };

    case UserActionTypes.EDIT_USER_SUCCESS:
      const updatedUsers = state.users.map((user) => {
        if (user.id === action.payload.id) {
          user = action.payload;
        }
        return user;
      });
      return {
        ...state,
        users: updatedUsers,
        editShow: false,
        editLoading: false,
      };

    case UserActionTypes.EDIT_USER_ERROR:
      return { ...state, editError: action.payload, editLoading: false };

    case UserActionTypes.EDIT_USER_OPEN:
      const userItem = state.users.filter((user) => user.id === action.payload);
      return {
        ...state,
        editShow: true,
        user: userItem ? userItem[0] : null,
      };

    case UserActionTypes.EDIT_USER_CLOSE:
      return { ...state, editShow: false, user: null };

    default:
      return state;
  }
};
```

userActionCreators.ts

```
import axios from "axios";
import Config from '../../config';
import { Dispatch } from "redux";
import { UserAction, UserActionTypes, UserItemType } from "../../types/todo";

export const fetchUsers = () => {
  return async (dispatch: Dispatch<UserAction>) => {
    try {
      dispatch({ type: UserActionTypes.FETCH_USERS });
```

```
      const res = await axios.get("${Config.APIUrl}");
      dispatch({
        type: UserActionTypes.FETCH_USERS_SUCCESS,
        payload: res.data,
      });
    } catch (error) {
      dispatch({
        type: UserActionTypes.FETCH_USERS_ERROR,
        payload: "Failed to load data from the server. Try again later",
      });
    }
  };
};

export const removeUser = (id: number) => {
  return async (dispatch: Dispatch<UserAction>) => {
    try {
      dispatch({ type: UserActionTypes.REMOVE_USER });
      const res = await axios.delete(
        `${Config.APIUrl}/${id}`
      );
      res &&
        dispatch({
          type: UserActionTypes.REMOVE_USER_SUCCESS,
          payload: id,
        });
    } catch (error) {
      dispatch({
        type: UserActionTypes.REMOVE_USER_ERROR,
        payload:
          "An error occurred while trying to delete the entry. Try again later",
      });
    }
  };
};

export const closeRemoveUserError = (): UserAction => {
  return { type: UserActionTypes.REMOVE_USER_ERROR_CLEAR };
};

export const showEdit = (id: number) => {
  return { type: UserActionTypes.EDIT_USER_OPEN, payload: id };
};

export const closeEdit = () => {
  return { type: UserActionTypes.EDIT_USER_CLOSE };
};

export const editUser = (todoItem: UserItemType) => {
```

```
  return async (dispatch: Dispatch<UserAction>) => {
    try {
      dispatch({
        type: UserActionTypes.EDIT_USER,
      });
      const res = await axios.put(
        `${Config.APIUrl}/${todoItem.id}`,
        { todoItem }
      );
      res &&
        dispatch({
          type: UserActionTypes.EDIT_USER_SUCCESS,
          payload: todoItem,
        });
    } catch (error) {
      dispatch({
        type: UserActionTypes.EDIT_USER_ERROR,
        payload: "An error occurred while trying to edit user. Try again later",
      });
    }
  };
};
```

actionCreators/indes.ts

```
import * as UserActionCreators from "./user";
import * as MessageActionCreators from "./message";
import * as NotificationActionCreators from "./notification";
import * as SearchActionCreators from "./search";
import * as PostActionCreators from "./post";
import * as CommentActionCreators from "./comment";
import * as ReactionActionCreators from "./comment";

export default {
  ... UserActionCreators,
  ... MessageActionCreators,
  ... NotificationActionCreators,
  ... SearchActionCreators,
  ... PostActionCreators,
  ... CommentActionCreators,
  ... ReactionActionCreators,
};
```

store/index.js

```
import { applyMiddleware, createStore } from "redux";
import thunk from "redux-thunk";
import { rootReducer } from "./reducers/index";
```

```
export const store = createStore(rootReducer, applyMiddleware(thunk));
```

components/userList.tsx

```
import React, { useEffect } from "react";
import { useActions } from "../hooks/useActions";
import { useTypesSelector } from "../hooks/useTypesSelector";
import UserItem from "./UserItem";

export const UserList: React.FC = () => {
  const { users, loading, error } = useTypesSelector((state) => state.user);
  const { fetchUsers } = useActions();

  useEffect(() => {
    fetchUsers();
  }, []);

  if (loading) {
    return <h1 className="loading-header">Loading... </h1>;
  }

  if (error) {
    return <h1 className="error-response"> {error} </h1>;
  }

  return (
    <div>
      {users.map((id, user) => (
        <UserItem id={id} user={user} />
      ))}
    </div>
  );
};

export default UserList;
```

components/pagination.tsx

```
import React from "react";

type PaginationProps = {
  currentPage: number;
  itemsPerPage: number;
  totalItems: number;
  paginate: (page: number) => void;
};

export const Pagination: React.FC<PaginationProps> = ({
  currentPage,
```

```
  itemsPerPage,
  totalItems,
  paginate,
}) => {
  const totalPages = totalItems / itemsPerPage;

  const nextPage = () => {
    if (currentPage < totalPages) {
      paginate(++currentPage);
    }
  };

  const prevPage = () => {
    if (currentPage > 1) {
      paginate(--currentPage);
    }
  };

  const firstPage = () => paginate(1);

  const lastPage = () => paginate(totalPages);

  return (
    <PaginationControls next={ nextPage } prev={ prevPage } />
  );
};
```

components/User/UserEdit.tsx

```
import React from "react";
import { useActions } from "../../../hooks/useActions";
import { useTypesSelector } from "../../../hooks/useTypesSelector";
import { UserItemType } from "../../../types/todo";
import { UserForm } from "../TodoForm/TodoForm";

type UserEditType = {
  isEditShow: boolean;
};

export const TodoEdit: React.FC< UserEditType> = ({ isEditShow }) => {
  const { user, editLoading } = useTypesSelector((state) => state.user);
  const { closeEdit, editUser } = useActions();

  const saveUser = (userItem: UserItemType) => {
    editUser(userItem);
  };

  return (
```

```
      < UserForm
        todo={todo}
        isShow={isEditShow}
        isLoading={editLoading}
        saveTodo={saveTodo}
        closeEdit={closeEdit}
      />
    );
};
```

Api/controlles/auth.js

```
const jwt = require('jsonwebtoken')
const User = require('../models/user')
const UserAccess = require('../models/userAccess')
const ForgotPassword = require('../models/forgotPassword')
const utils = require('../middleware/utils')
const uuid = require('uuid')
const { addHours } = require('date-fns')
const { matchedData } = require('express-validator')
const auth = require('../middleware/auth')
const emailer = require('../middleware/emailer')
const HOURS_TO_BLOCK = 2
const LOGIN_ATTEMPTS = 5

/*********************
 * Private functions *
 *********************/

/**
 * Generates a token
 * @param {Object} user - user object
 */
const generateToken = user => {
  // Gets expiration time
  const expiration =
    Math.floor(Date.now() / 1000) + 60 * process.env.JWT_EXPIRATION_IN_MINUTES

  // returns signed and encrypted token
  return auth.encrypt(
    jwt.sign(
      {
        data: {
          _id: user
        },
        exp: expiration
      },
      process.env.JWT_SECRET
```

```
    )
  )
}

/**
 * Creates an object with user info
 * @param {Object} req - request object
 */
const setUserInfo = req => {
  const user = {
    _id: req._id,
    name: req.name,
    email: req.email,
    role: req.role
  }
  return user
}

/**
 * Saves a new user access and then returns token
 * @param {Object} req - request object
 * @param {Object} user - user object
 */
const saveUserAccessAndReturnToken = async (req, user) => {
  return new Promise((resolve, reject) => {
    const userAccess = new UserAccess({
      email: user.email,
      ip: utils.getIP(req),
      browser: utils.getBrowserInfo(req)
    })
    userAccess.save(err => {
      if (err) {
        reject(utils.buildErrObject(422, err.message))
      }
      const userInfo = setUserInfo(user)
      // Returns data with access token
      resolve({
        token: generateToken(user._id),
        user: userInfo
      })
    })
  })
}

/**
 * Blocks a user by setting blockExpires to the specified date based on constant
HOURS_TO_BLOCK
 * @param {Object} user - user object
 */
```

```
const blockUser = async user => {
  return new Promise((resolve, reject) => {
    user.blockExpires = addHours(new Date(), HOURS_TO_BLOCK)
    user.save((err, result) => {
      if (err) {
        reject(utils.buildErrObject(422, err.message))
      }
      if (result) {
        resolve(utils.buildErrObject(409, 'BLOCKED_USER'))
      }
    })
  })
}

/**
 * Saves login attempts to dabatabse
 * @param {Object} user - user object
 */
const saveLoginAttemptsToDB = async user => {
  return new Promise((resolve, reject) => {
    user.save((err, result) => {
      if (err) {
        reject(utils.buildErrObject(422, err.message))
      }
      if (result) {
        resolve(true)
      }
    })
  })
}

/**
 * Checks that login attempts are greater than specified in constant and also that
blockexpires is less than now
 * @param {Object} user - user object
 */
const blockIsExpired = user =>
  user.loginAttempts > LOGIN_ATTEMPTS && user.blockExpires <= new Date()

/**
 *
 * @param {Object} user - user object.
 */
const checkLoginAttemptsAndBlockExpires = async user => {
  return new Promise((resolve, reject) => {
    // Let user try to login again after blockexpires, resets user loginAttempts
    if (blockIsExpired(user)) {
      user.loginAttempts = 0
      user.save((err, result) => {
```

```
        if (err) {
          reject(utils.buildErrObject(422, err.message))
        }
        if (result) {
          resolve(true)
        }
      })
    } else {
      // User is not blocked, check password (normal behaviour)
      resolve(true)
    }
  })
}

/**
 * Checks if blockExpires from user is greater than now
 * @param {Object} user - user object
 */
const userIsBlocked = async user => {
  return new Promise((resolve, reject) => {
    if (user.blockExpires > new Date()) {
      reject(utils.buildErrObject(409, 'BLOCKED_USER'))
    }
    resolve(true)
  })
}

/**
 * Finds user by email
 * @param {string} email - user´s email
 */
const findUser = async email => {
  return new Promise((resolve, reject) => {
    User.findOne(
      {
        email
      },
      'password loginAttempts blockExpires name email role',
      (err, item) => {
        utils.itemNotFound(err, item, reject, 'USER_DOES_NOT_EXIST')
        resolve(item)
      }
    )
  })
}

/**
 * Finds user by ID
 * @param {string} id - user´s id
```

```
 */
const findUserById = async userId => {
  return new Promise((resolve, reject) => {
    User.findById(userId, (err, item) => {
      utils.itemNotFound(err, item, reject, 'USER_DOES_NOT_EXIST')
      resolve(item)
    })
  })
}

/**
 * Adds one attempt to loginAttempts, then compares loginAttempts with the constant
LOGIN_ATTEMPTS, if is less returns wrong password, else returns blockUser function
 * @param {Object} user - user object
 */
const passwordsDoNotMatch = async user => {
  user.loginAttempts += 1
  await saveLoginAttemptsToDB(user)
  return new Promise((resolve, reject) => {
    if (user.loginAttempts <= LOGIN_ATTEMPTS) {
      resolve(utils.buildErrObject(409, 'WRONG_PASSWORD'))
    } else {
      resolve(blockUser(user))
    }
    reject(utils.buildErrObject(422, 'ERROR'))
  })
}

/**
 * Registers a new user in database
 * @param {Object} req - request object
 */
const registerUser = async req => {
  return new Promise((resolve, reject) => {
    const user = new User({
      name: req.name,
      email: req.email,
      password: req.password
    })
    user.save((err, item) => {
      if (err) {
        reject(utils.buildErrObject(422, err.message))
      }
      resolve(item)
    })
  })
}

/**
```

```
 * Builds the registration token
 * @param {Object} item - user object that contains created id
 * @param {Object} userInfo - user object
 */
const returnRegisterToken = (item, userInfo) => {
  const data = {
    token: generateToken(item._id),
    user: userInfo
  }
  return data
}

/**
 * Marks a request to reset password as used
 * @param {Object} req - request object
 * @param {Object} forgot - forgot object
 */
const markResetPasswordAsUsed = async (req, forgot) => {
  return new Promise((resolve, reject) => {
    forgot.used = true
    forgot.ipChanged = utils.getIP(req)
    forgot.browserChanged = utils.getBrowserInfo(req)
    forgot.save((err, item) => {
      utils.itemNotFound(err, item, reject, 'NOT_FOUND')
      resolve(utils.buildSuccObject('PASSWORD_CHANGED'))
    })
  })
}

/**
 * Updates a user password in database
 * @param {string} password - new password
 * @param {Object} user - user object
 */
const updatePassword = async (password, user) => {
  return new Promise((resolve, reject) => {
    user.password = password
    user.save((err, item) => {
      utils.itemNotFound(err, item, reject, 'NOT_FOUND')
      resolve(item)
    })
  })
}

/**
 * Finds user by email to reset password
 * @param {string} email - user email
 */
const findUserToResetPassword = async email => {
```

```
  return new Promise((resolve, reject) => {
    User.findOne(
      {
        email
      },
      (err, user) => {
        utils.itemNotFound(err, user, reject, 'NOT_FOUND')
        resolve(user)
      }
    )
  })
}

/**
 * Checks if a forgot password verification exists
 * @param {string} id - verification id
 */
const findForgotPassword = async id => {
  return new Promise((resolve, reject) => {
    ForgotPassword.findOne(
      {
        verification: id,
        used: false
      },
      (err, item) => {
        utils.itemNotFound(err, item, reject, 'NOT_FOUND_OR_ALREADY_USED')
        resolve(item)
      }
    )
  })
}

/**
 * Checks against user if has quested role
 * @param {Object} data - data object
 * @param {*} next - next callback
 */
const checkPermissions = async (data, next) => {
  return new Promise((resolve, reject) => {
    User.findById(data.id, (err, result) => {
      utils.itemNotFound(err, result, reject, 'NOT_FOUND')
      if (data.roles.indexOf(result.role) > -1) {
        return resolve(next())
      }
      return reject(utils.buildErrObject(401, 'UNAUTHORIZED'))
    })
  })
}
```

```
/**
 * Gets user id from token
 * @param {string} token - Encrypted and encoded token
 */
const getUserIdFromToken = async token => {
  return new Promise((resolve, reject) => {
    // Decrypts, verifies and decode token
    jwt.verify(auth.decrypt(token), process.env.JWT_SECRET, (err, decoded) => {
      if (err) {
        reject(utils.buildErrObject(409, 'BAD_TOKEN'))
      }
      resolve(decoded.data._id)
    })
  })
}

/********************
 * Public functions *
 ********************/

/**
 * Login function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.login = async (req, res) => {
  try {
    const data = matchedData(req)
    const user = await findUser(data.email)
    await userIsBlocked(user)
    await checkLoginAttemptsAndBlockExpires(user)
    const isPasswordMatch = await auth.checkPassword(data.password, user)
    if (!isPasswordMatch) {
      utils.handleError(res, await passwordsDoNotMatch(user))
    } else {
      // all ok, register access and return token
      user.loginAttempts = 0
      await saveLoginAttemptsToDB(user)
      res.status(200).json(await saveUserAccessAndReturnToken(req, user))
    }
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Register function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
```

```
 */
exports.register = async (req, res) => {
  try {
    // Gets locale from header 'Accept-Language'
    const locale = req.getLocale()
    req = matchedData(req)
    const doesEmailExists = await emailer.emailExists(req.email)
    if (!doesEmailExists) {
      const item = await registerUser(req)
      const userInfo = setUserInfo(item)
      const response = returnRegisterToken(item, userInfo)
      emailer.sendRegistrationEmailMessage(locale, item)
      res.status(201).json(response)
    }
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Reset password function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.resetPassword = async (req, res) => {
  try {
    const data = matchedData(req)
    const forgotPassword = await findForgotPassword(data.id)
    const user = await findUserToResetPassword(forgotPassword.email)
    await updatePassword(data.password, user)
    const result = await markResetPasswordAsUsed(req, forgotPassword)
    res.status(200).json(result)
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Refresh token function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.getRefreshToken = async (req, res) => {
  try {
    const tokenEncrypted = req.headers.authorization
      .replace('Bearer ', '')
      .trim()
    let userId = await getUserIdFromToken(tokenEncrypted)
    userId = await utils.isIDGood(userId)
```

```
    const user = await findUserById(userId)
    const token = await saveUserAccessAndReturnToken(req, user)
    // Removes user info from response
    delete token.user
    res.status(200).json(token)
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Roles authorization function called by route
 * @param {Array} roles - roles specified on the route
 */
exports.roleAuthorization = roles => async (req, res, next) => {
  try {
    const data = {
      id: req.user._id,
      roles
    }
    await checkPermissions(data, next)
  } catch (error) {
    utils.handleError(res, error)
  }
}
```

Api/controlles/auth.validate.js

```
const { validationResult } = require('../middleware/utils')
const { check } = require('express-validator')

/**
 * Validates register request
 */
exports.register = [
  check('name')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY'),
  check('email')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY')
    .isEmail()
```

```
    .withMessage('EMAIL_IS_NOT_VALID'),
  check('password')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY')
    .isLength({
      min: 5
    })
    .withMessage('PASSWORD_TOO_SHORT_MIN_5'),
  (req, res, next) => {
    validationResult(req, res, next)
  }
]

/**
 * Validates login request
 */
exports.login = [
  check('email')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY')
    .isEmail()
    .withMessage('EMAIL_IS_NOT_VALID'),
  check('password')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY')
    .isLength({
      min: 5
    })
    .withMessage('PASSWORD_TOO_SHORT_MIN_5'),
  (req, res, next) => {
    validationResult(req, res, next)
  }
]

/**
 * Validates verify request
 */
exports.verify = [
  check('id')
    .exists()
```

```
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY'),
  (req, res, next) => {
    validationResult(req, res, next)
  }
]

/**
 * Validates reset password request
 */
exports.resetPassword = [
  check('id')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY'),
  check('password')
    .exists()
    .withMessage('MISSING')
    .not()
    .isEmpty()
    .withMessage('IS_EMPTY')
    .isLength({
      min: 5
    })
    .withMessage('PASSWORD_TOO_SHORT_MIN_5'),
  (req, res, next) => {
    validationResult(req, res, next)
  }
]
```

Api/controllers/profile.js

```
const model = require('../models/user')
const utils = require('../middleware/utils')
const { matchedData } = require('express-validator')
const auth = require('../middleware/auth')

/**********************
 * Private functions *
 **********************/

/**
 * Gets profile from database by id
 * @param {string} id - user id
```

```
 */
const getProfileFromDB = async id => {
  return new Promise((resolve, reject) => {
    model.findById(id, '-_id -updatedAt -createdAt', (err, user) => {
      utils.itemNotFound(err, user, reject, 'NOT_FOUND')
      resolve(user)
    })
  })
}

/**
 * Updates profile in database
 * @param {Object} req - request object
 * @param {string} id - user id
 */
const updateProfileInDB = async (req, id) => {
  return new Promise((resolve, reject) => {
    model.findByIdAndUpdate(
      id,
      req,
      {
        new: true,
        runValidators: true,
        select: '-role -_id -updatedAt -createdAt'
      },
      (err, user) => {
        utils.itemNotFound(err, user, reject, 'NOT_FOUND')
        resolve(user)
      }
    )
  })
}

/**
 * Finds user by id
 * @param {string} email - user id
 */
const findUser = async id => {
  return new Promise((resolve, reject) => {
    model.findById(id, 'password email', (err, user) => {
      utils.itemNotFound(err, user, reject, 'USER_DOES_NOT_EXIST')
      resolve(user)
    })
  })
}

/**
 * Build passwords do not match object
 * @param {Object} user - user object
```

```
 */
const passwordsDoNotMatch = async () => {
  return new Promise(resolve => {
    resolve(utils.buildErrObject(409, 'WRONG_PASSWORD'))
  })
}

/**
 * Changes password in database
 * @param {string} id - user id
 * @param {Object} req - request object
 */
const changePasswordInDB = async (id, req) => {
  return new Promise((resolve, reject) => {
    model.findById(id, '+password', (err, user) => {
      utils.itemNotFound(err, user, reject, 'NOT_FOUND')

      // Assigns new password to user
      user.password = req.newPassword

      // Saves in DB
      user.save(error => {
        if (err) {
          reject(utils.buildErrObject(422, error.message))
        }
        resolve(utils.buildSuccObject('PASSWORD_CHANGED'))
      })
    })
  })
}

/********************
 * Public functions *
 ********************/

/**
 * Get profile function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.getProfile = async (req, res) => {
  try {
    const id = await utils.isIDGood(req.user._id)
    res.status(200).json(await getProfileFromDB(id))
  } catch (error) {
    utils.handleError(res, error)
  }
}
```

```
/**
 * Update profile function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.updateProfile = async (req, res) => {
  try {
    const id = await utils.isIDGood(req.user._id)
    req = matchedData(req)
    res.status(200).json(await updateProfileInDB(req, id))
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Change password function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.changePassword = async (req, res) => {
  try {
    const id = await utils.isIDGood(req.user._id)
    const user = await findUser(id)
    req = matchedData(req)
    const isPasswordMatch = await auth.checkPassword(req.oldPassword, user)
    if (!isPasswordMatch) {
      utils.handleError(res, await passwordsDoNotMatch())
    } else {
      // all ok, proceed to change password
      res.status(200).json(await changePasswordInDB(id, req))
    }
  } catch (error) {
    utils.handleError(res, error)
  }
}
```

Api/controllers/user.js

```
const model = require('../models/user')
const uuid = require('uuid')
const { matchedData } = require('express-validator')
const utils = require('../middleware/utils')
const db = require('../middleware/db')
const emailer = require('../middleware/emailer')

/*********************
```

```
 * Private functions *
 ********************/

/**
 * Creates a new item in database
 * @param {Object} req - request object
 */
const createItem = async req => {
  return new Promise((resolve, reject) => {
    const user = new model({
      name: req.name,
      email: req.email,
      password: req.password,
      role: req.role
    })
    user.save((err, item) => {
      if (err) {
        reject(utils.buildErrObject(422, err.message))
      }
      // Removes properties with rest operator
      const removeProperties = ({
        // eslint-disable-next-line no-unused-vars
        password,
        // eslint-disable-next-line no-unused-vars
        blockExpires,
        // eslint-disable-next-line no-unused-vars
        loginAttempts,
        ...rest
      }) => rest
      resolve(removeProperties(item.toObject()))
    })
  })
}

/********************
 * Public functions *
 ********************/

/**
 * Get items function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.getItems = async (req, res) => {
  try {
    const query = await db.checkQueryString(req.query)
    res.status(200).json(await db.getItems(req, model, query))
  } catch (error) {
    utils.handleError(res, error)
```

```
  }
}

/**
 * Get item function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.getItem = async (req, res) => {
  try {
    req = matchedData(req)
    const id = await utils.isIDGood(req.id)
    res.status(200).json(await db.getItem(id, model))
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Update item function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.updateItem = async (req, res) => {
  try {
    req = matchedData(req)
    const id = await utils.isIDGood(req.id)
    const doesEmailExists = await emailer.emailExistsExcludingMyself(
      id,
      req.email
    )
    if (!doesEmailExists) {
      res.status(200).json(await db.updateItem(id, model, req))
    }
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Create item function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.createItem = async (req, res) => {
  try {
    // Gets locale from header 'Accept-Language'
    const locale = req.getLocale()
    req = matchedData(req)
```

```
    const doesEmailExists = await emailer.emailExists(req.email)
    if (!doesEmailExists) {
      const item = await createItem(req)
      emailer.sendRegistrationEmailMessage(locale, item)
      res.status(201).json(item)
    }
  } catch (error) {
    utils.handleError(res, error)
  }
}

/**
 * Delete item function called by route
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
exports.deleteItem = async (req, res) => {
  try {
    req = matchedData(req)
    const id = await utils.isIDGood(req.id)
    res.status(200).json(await db.deleteItem(id, model))
  } catch (error) {
    utils.handleError(res, error)
  }
}
```

Middleware/db.js

```
const {
  buildSuccObject,
  buildErrObject,
  itemNotFound
} = require('../middleware/utils')

/**
 * Builds sorting
 * @param {string} sort - field to sort from
 * @param {number} order - order for query (1,-1)
 */
const buildSort = (sort, order) => {
  const sortBy = {}
  sortBy[sort] = order
  return sortBy
}

/**
 * Hack for mongoose-paginate, removes 'id' from results
 * @param {Object} result - result object
```

```
 */
const cleanPaginationID = result => {
  result.docs.map(element => delete element.id)
  return result
}

/**
 * Builds initial options for query
 * @param {Object} query - query object
 */
const listInitOptions = async req => {
  return new Promise(resolve => {
    const order = req.query.order || -1
    const sort = req.query.sort || 'createdAt'
    const sortBy = buildSort(sort, order)
    const page = parseInt(req.query.page, 10) || 1
    const limit = parseInt(req.query.limit, 10) || 5
    const options = {
      sort: sortBy,
      lean: true,
      page,
      limit
    }
    resolve(options)
  })
}

module.exports = {
  /**
   * Checks the query string for filtering records
   * query.filter should be the text to search (string)
   * query.fields should be the fields to search into (array)
   * @param {Object} query - query object
   */
  async checkQueryString(query) {
    return new Promise((resolve, reject) => {
      try {
        if (
          typeof query.filter !== 'undefined' &&
          typeof query.fields !== 'undefined'
        ) {
          const data = {
            $or: []
          }
          const array = []
          // Takes fields param and builds an array by splitting with ','
          const arrayFields = query.fields.split(',')
          // Adds SQL Like %word% with regex
          arrayFields.map(item => {
```

```
            array.push({
              [item]: {
                $regex: new RegExp(query.filter, 'i')
              }
            })
          })
          // Puts array result in data
          data.$or = array
          resolve(data)
        } else {
          resolve({})
        }
      } catch (err) {
        console.log(err.message)
        reject(buildErrObject(422, 'ERROR_WITH_FILTER'))
      }
    })
  },

  /**
   * Gets items from database
   * @param {Object} req - request object
   * @param {Object} query - query object
   */
  async getItems(req, model, query) {
    const options = await listInitOptions(req)
    return new Promise((resolve, reject) => {
      model.paginate(query, options, (err, items) => {
        if (err) {
          reject(buildErrObject(422, err.message))
        }
        resolve(cleanPaginationID(items))
      })
    })
  },

  /**
   * Gets item from database by id
   * @param {string} id - item id
   */
  async getItem(id, model) {
    return new Promise((resolve, reject) => {
      model.findById(id, (err, item) => {
        itemNotFound(err, item, reject, 'NOT_FOUND')
        resolve(item)
      })
    })
  },
```

```
/**
 * Creates a new item in database
 * @param {Object} req - request object
 */
async createItem(req, model) {
  return new Promise((resolve, reject) => {
    model.create(req, (err, item) => {
      if (err) {
        reject(buildErrObject(422, err.message))
      }
      resolve(item)
    })
  })
},

/**
 * Updates an item in database by id
 * @param {string} id - item id
 * @param {Object} req - request object
 */
async updateItem(id, model, req) {
  return new Promise((resolve, reject) => {
    model.findByIdAndUpdate(
      id,
      req,
      {
        new: true,
        runValidators: true
      },
      (err, item) => {
        itemNotFound(err, item, reject, 'NOT_FOUND')
        resolve(item)
      }
    )
  })
},

/**
 * Deletes an item from database by id
 * @param {string} id - id of item
 */
async deleteItem(id, model) {
  return new Promise((resolve, reject) => {
    model.findByIdAndRemove(id, (err, item) => {
      itemNotFound(err, item, reject, 'NOT_FOUND')
      resolve(buildSuccObject('DELETED'))
    })
  })
}
```

```
}
```

Middleware/utils.js

```
const mongoose = require('mongoose')
const requestIp = require('request-ip')
const { validationResult } = require('express-validator')

/**
 * Removes extension from file
 * @param {string} file - filename
 */
exports.removeExtensionFromFile = file => {
  return file
    .split('.')
    .slice(0, -1)
    .join('.')
    .toString()
}

/**
 * Gets IP from user
 * @param {*} req - request object
 */
exports.getIP = req => requestIp.getClientIp(req)

/**
 * Gets browser info from user
 * @param {*} req - request object
 */
exports.getBrowserInfo = req => req.headers['user-agent']

/**
 * Gets country from user using CloudFlare header 'cf-ipcountry'
 * @param {*} req - request object
 */
exports.getCountry = req =>
  req.headers['cf-ipcountry'] ? req.headers['cf-ipcountry'] : 'XX'

/**
 * Handles error by printing to console in development env and builds and sends an
error response
 * @param {Object} res - response object
 * @param {Object} err - error object
 */
exports.handleError = (res, err) => {
  // Prints error in console
  if (process.env.NODE_ENV === 'development') {
```

```
    console.log(err)
  }
  // Sends error to user
  res.status(err.code).json({
    errors: {
      msg: err.message
    }
  })
}

/**
 * Builds error object
 * @param {number} code - error code
 * @param {string} message - error text
 */
exports.buildErrObject = (code, message) => {
  return {
    code,
    message
  }
}

/**
 * Builds error for validation files
 * @param {Object} req - request object
 * @param {Object} res - response object
 * @param {Object} next - next object
 */
exports.validationResult = (req, res, next) => {
  try {
    validationResult(req).throw()
    if (req.body.email) {
      req.body.email = req.body.email.toLowerCase()
    }
    return next()
  } catch (err) {
    return this.handleError(res, this.buildErrObject(422, err.array()))
  }
}

/**
 * Builds success object
 * @param {string} message - success text
 */
exports.buildSuccObject = message => {
  return {
    msg: message
  }
}
```

```
/**
 * Checks if given ID is good for MongoDB
 * @param {string} id - id to check
 */
exports.isIDGood = async id => {
  return new Promise((resolve, reject) => {
    const goodID = mongoose.Types.ObjectId.isValid(id)
    return goodID
      ? resolve(id)
      : reject(this.buildErrObject(422, 'ID_MALFORMED'))
  })
}

/**
 * Item not found
 * @param {Object} err - error object
 * @param {Object} item - item result object
 * @param {Object} reject - reject object
 * @param {string} message - message
 */
exports.itemNotFound = (err, item, reject, message) => {
  if (err) {
    reject(this.buildErrObject(422, err.message))
  }
  if (!item) {
    reject(this.buildErrObject(404, message))
  }
}

/**
 * Item already exists
 * @param {Object} err - error object
 * @param {Object} item - item result object
 * @param {Object} reject - reject object
 * @param {string} message - message
 */
exports.itemAlreadyExists = (err, item, reject, message) => {
  if (err) {
    reject(this.buildErrObject(422, err.message))
  }
  if (item) {
    reject(this.buildErrObject(422, message))
  }
}
```

Models/userAccess.js

```
const mongoose = require('mongoose')
const validator = require('validator')

const UserAccessSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      validate: {
        validator: validator.isEmail,
        message: 'EMAIL_IS_NOT_VALID'
      },
      lowercase: true,
      required: true
    },
    ip: {
      type: String,
      required: true
    },
    browser: {
      type: String,
      required: true
    }
  },
  {
    versionKey: false,
    timestamps: true
  }
)
module.exports = mongoose.model('UserAccess', UserAccessSchema)
```