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-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2022 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ПАКЕТНИЙ МОДУЛЬ ДЛЯ НАЛАШТУВАННЯ АРХІТЕКТУРИ ТА ЛОГІКИ NODE.JS ПРОЄКТУ З ВИКОРИСТАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Спеціальність 124 «Системний аналіз»

124 – МКР – 607.1610320

Студент _____Д. С. Сапонько
«__» _____ 2022 р.

Консультант _____І. В. Кулаковська
к.ф.м.н., доцент кафедри ІІС
«__» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Npm –Менеджер пакетів Node.js.

Node.JS – Платформа для виконання JavaScript коду.

Sli – Інтерфейс З'єднань, що масштабується

Jwt – JSON Web Token

SOA – Сервісно-орієнтована архітектура

СУБД – система управляння базами даних

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«ПАКЕТНИЙ МОДУЛЬ ДЛЯ НАЛАШТУВАННЯ АРХІТЕКТУРИ ТА ЛОГІКИ NODE.JS ПРОЄКТУ З ВИКОРИСТАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ**»**

Спеціальність 124 «Системний аналіз»

124 – МКР – 607.1610320

Студент _____Д. С. Сапонько
«__» _____ 2022 р.

Консультант __ *І. В. Кулаковська*
к.ф.м.н., доцент кафедри ІІС
«__» _____ 2022 р.

Миколаїв – 2022

ВСТУП

На сьогоднішній день інтелектуальні технології займаюсь вагому роль у житті людини.

Інформаційні технології нині розвиваються з швидко зростаючою інтенсивністю, зумовлюючи зміни всіх рівнів буття нашого суспільства та людини, зокрема та її повсякденні. Причому більшість людей навіть не усвідомлюють, наскільки їх повсякденне життя залежить від комп'ютерів та інформаційно-телекомунікаційних систем. Повсякденність навіть тих людей, які жодного разу в житті не включали комп'ютер і не здійснили жодного дзвінка по мобільному телефону, багато в чому пов'язана з розвитком інформаційних технологій, адже за їх допомогою здійснюється управління системою життєзабезпечення. Подача електроенергії та водопостачання, управління системами зв'язку та транспорту, координація фінансових потоків, функціонування медицини та освіти не мислимі в даний час без використання комп'ютерних систем. Інформаційно-комунікаційні технології прямо і безпосередньо впливають на повсякденне життя людини, визначаючи специфіку та якість її праці, побуту, дозвілля, способу життя і навіть мислення. Крім того, інформаційні технології значно розширюють можливості контролю не лише суспільного, а й приватного життя громадян.

Промисловість, транспорт, сільське господарство, індустрія відпочинку та розваг - все залежить від комп'ютерів. Кількість галузей, у яких задіяні цифрові технології, неухильно зростає. Потрібно все більше фахівців для їх обслуговування. Поступово йде у минуле важка людська праця. Його замінює праця інтелектуальна. З'являються нові спеціальності та професії.

Серед відмінних властивостей інформаційних технологій, доцільним виділити такі сім найважливіших.

Інформаційні технології дозволяють активізувати та ефективно використати інформаційні ресурси суспільства, які сьогодні є найважливішим стратегічним фактором його розвитку. Досвід показує, що активізація,

поширення та ефективне використання інформаційних ресурсів (наукових знань, винаходів, технологій, передового досвіду) дозволяють отримати суттєву економію інших видів ресурсів: сировини, енергії, корисних копалин, матеріалів та обладнання, людських ресурсів, соціального часу.

Інформаційні технології дозволяють оптимізувати і в багатьох випадках автоматизувати інформаційні процеси, які останніми роками посідають все більше місце у життєдіяльності людського суспільства. Загальновідомо, що розвиток цивілізації відбувається у напрямі становлення інформаційного суспільства, в якому об'єктами та результатами праці більшості зайнятого населення стають уже не матеріальні цінності, а головним чином інформація та наукові знання.

Інформаційні процеси є важливими елементами інших складніших виробничих або соціальних процесів. Тому дуже часто й інформаційні технології виступають як компоненти відповідних виробничих чи соціальних технологій. У цьому вони, зазвичай, реалізують найважливіші, " інтелектуальні " функції цих технологій.

Інформаційні технології сьогодні відіграють виключно важливу роль у забезпеченні інформаційної взаємодії між людьми, а також у системах підготовки та розповсюдження масової інформації. На додачу до традиційних засобів зв'язку (телефон, телеграф, радіо і телебачення) у соціальній сфері все більш широко використовуються системи електронних телекомунікацій, електронна пошта, факсимільна передача інформації та інші види зв'язку. Ці кошти швидко асимілюються культурою сучасного суспільства, оскільки вони не лише створюють великі зручності, а й знімають багато виробничих, соціальних та побутових проблем.

Інформаційні технології займають сьогодні центральне місце у процесі інтелектуалізації суспільства, розвитку його системи освіти та культури. Практично у всіх розвинених і багатьох країнах що розвиваються комп'ютерна

і телевізійна техніка, навчальні програми на оптичних дисках і мультимедіа - технології стають звичними атрибутами як вищих навчальних закладів, а й традиційних шкіл системи початкової та середньої освіти.

Інформаційні технології відіграють нині ключову роль також у процесах отримання та накопичення нових знань. При цьому, на зміну традиційним методам інформаційної підтримки наукових досліджень шляхом накопичення, класифікації та розповсюдження науково-технічної інформації приходять нові методи, засновані на використанні можливостей інформаційної підтримки фундаментальної та прикладної науки, що знову відкриваються, які надають сучасні інформаційні технології.

Принципово важливе для сучасного етапу розвитку суспільства значення розвитку інформаційних технологій полягає в тому, що їх використання може істотно сприяти у вирішенні глобальних проблем людства і, насамперед, проблем, пов'язаних з необхідністю подолання переживаної світовою спільнотою глобальної кризи цивілізації. Адже саме методи інформаційного моделювання глобальних процесів, особливо у поєднанні з методами космічного інформаційного моніторингу, можуть забезпечити вже сьогодні можливість прогнозування багатьох кризових ситуацій у регіонах підвищеної соціальної та політичної напруги, а також у районах екологічного лиха, у місцях природних катастроф та великих технологічних аварій, які представляють підвищену для суспільства.

В даний час практично всі організації необхідні інформаційне обслуговування, переробка великої кількості інформації. Однією з основних технічним засобом передачі, сприйняття, обробки інформації є комп'ютер. Роль комп'ютера зусилля інтелектуальних можливостей людини і суспільства в цілому служить для зв'язку та передачі інформації.

Мета МКР – зменшення часу та полегшення розробки програмного забезпечення за допомогою розробки npm модуля для генерації структури проєкту з використання мікросервісної архітектури.

Для досягнення мети потрібно виконати такі завдання, як:

- виявити особливості проектування та створення веб серверів;
- проаналізувати та визначити особливості написання та опублікування npm модулів;
- розробити додаток для створення та побудови архітектури проєкту;
- провести тести додатку;

Об’єктом досліджень – технологія розробки npm модуля доступного розробникам за допомогою однієї консольної команди.

Предметом досліджень –алгоритми розробки npm модулів та методи опублікування модуля в загальній npm базі даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕНЬ. ПОСТАНОВКА ЗАДАЧІ

1.1 Історія та розвиток програмного забезпечення

Історія розробки програмного забезпечення та і власне саме програмне забезпечення пережила дуже цікаві та складні часи від перших програм до глобальних корпоративних застосунків які використовуються сьогодні по усьому світі.

Термін «програмне забезпечення» не використовувався до кінця 1950-х років. Протягом цього часу, хоча створювалися різні типи програмного забезпечення програмування, вони, зазвичай, були комерційно доступні. Отже, користувачам - здебільшого вченим та великим підприємствам - часто доводилося писати власне програмне забезпечення.

Червня 1948 року. Том Кілберн, вчений комп'ютерник, пише першу у світі програму для комп'ютера Manchester Baby в Манчестерському університеті в Англії.

На початку 1950-х. General Motors створює першу ОС для електронної машини обробки даних IBM 701. Вона має назву Операційна система General Motors або GM OS.

В 1958 році, статистик Джон Тьюкі використовує слово "програмне забезпечення" у статті про комп'ютерне програмування.

В Кінці 1960-х років. З'являються дискети, які використовуються у 1980-х та 1990-х роках для поширення програмного забезпечення.

Листопад 1971 року. AT&T випускає першу версію ОС Unix.

VisiCorp випускає VisiCalc Apple II, перше програмне забезпечення для роботи з електронними таблицями для персональних комп'ютерів.

Microsoft випускає MS-DOS, ОС, на якій працювало багато ранніх комп'ютерів IBM. IBM починає продавати програмне забезпечення, а

комерційне програмне забезпечення стає доступним для середнього споживача.

Жорсткі диски стають стандартом для ПК і виробники починають об'єднувати програмне забезпечення в комп'ютери.

Рух за вільне програмне забезпечення починається з проекту Linux Річарда Столлмана GNU (GNU is not Unix) зі створення Unix-подібної ОС з вихідним кодом, який можна вільно копіювати, змінювати та розповсюджувати.

Випущено Mac OS, що працює під керуванням лінійки Apple Macintosh.

Середина 1980-х років. Випущено ключові програмні програми, включаючи AutoDesk AutoCAD, Microsoft Word та Microsoft Excel. Випущено Microsoft Windows 1.0.

CD-ROM стають стандартом і містять набагато більше даних, ніж дискети. Великі програми можна розповсюджувати швидко, легко та відносно недорого.

Linux ядро , основа відкритого вихідного коду ОС Linux, звільняється.

З'являються DVD-диски, здатні зберігати більше даних, ніж компакт-диски, що дозволяє розміщувати пакети програм, такі як Microsoft Office Suite, на один диск.

Salesforce.com використовує хмарні обчислення для доставки програмного забезпечення через Інтернет.

Термін "програмне забезпечення як послуга" (SaaS) входить у моду.

Випущено iPhone, і мобільні програми починають завойовувати популярність.

Як ми можемо бачити, історія програмного забезпечення є досить об’ємною та цікавою. Саме програмне забезпечення пережило досить багато і розробника ПЗ довелося пройти дуже складний шлях для отримання таких результатів в розробці які ми маємо на сьогодні.

1.2 Етапи розробки програмного забезпечення

Життєвий цикл розробки програмного забезпечення (SDLC) - це процес створення програмного забезпечення з найвищою якістю та найменшими витратами.

Хоча технології, методи та погляди на створення високопродуктивних та масштабованих програмних сервісів змінилися, обов'язки та дії залишилися незмінними.

Життєвий цикл розробки програмного забезпечення (SDLC) – це серія важливих етапів, визначених для команд, які виробляють та постачають високоякісне програмне забезпечення.

Життєвий цикл розробки програмного забезпечення належить до етапів роботи, пов’язаних із створенням програмних додатків. Кожен етап відповідає ролі або обов'язкам, які учасники розробки програмного забезпечення повинні розуміти, управляти та оптимізувати, щоб надавати свої програмні послуги з високою швидкістю та продуктивністю. Ці етапи роботи включають [1]:

- збір вимог;
- дизайн програмного забезпечення;
- розробка програмного забезпечення;
- тестування та інтеграція;
- розгортання;
- експлуатація та обслуговування;

Щоб краще розуміти як саме проходить розробка програмного забезпечення, необхідно детально розібрати кожен із етапів див. рис. 1.1.

Рис. 1.1 Модель життєвого циклу розробки програмного забезпечення.

Життєвий цикл розробки ПЗ. Збір вимог. На цьому етапі роботи команда виявляє, збирає та визначає поточні проблеми, вимоги, запити та очікування клієнтів, пов'язані з програмою або послугою.

Деякі дії, пов'язані з етапом збору вимог, можуть включати створення специфікацій програмного забезпечення, створення докладного плану, документації, відстеження проблем і планування проекту або продукту, включаючи виділення правильних ресурсів.

Визначення вимог до програмного забезпечення або продукту дає командам можливість передбачати і контекст, необхідних доставки і виробництва своїх програмних рішень.

Життєвий цикл розробки ПЗ. Дизайн програмного забезпечення**.** На цьому етапі проектування команда приймає рішення щодо розробки програмного забезпечення щодо архітектури та створення програмного рішення. Це може включати створення проектної документації, посібників з кодування та обговорення інструментів, практик, середовища виконання або фреймворків, які допоможуть групі виконати специфікацію вимог до програмного забезпечення та цілі, визначені на етапі збору вимог. Також на цьому етапі проходить побудова дизайну програмного забезпечення.

Життєвий цикл розробки ПЗ. Розробка ПЗ. На цьому етапі роботи команди створюють програмні рішення на основі ухвалених проектних рішень. Тут команди досягають цілей та досягають результатів, встановлених на етапі збору вимог до програмного забезпечення, шляхом впровадження рішення.

У процесі розробки можуть бути задіяні команди людей, нові технології та несподівані виклики; однак, групи розробників зазвичай працюють з технічними керівниками та менеджерами з продуктів або проектів, щоб розблокувати процес, прийняти рішення або надати підтримку.

Життєвий цикл розробки ПЗ. Тестування та інтеграція. На цьому етапі роботи програмна реалізація упаковується та тестується для забезпечення якості. Тестування чи гарантія якості гарантує, що реалізовані рішення відповідають стандартам якості та продуктивності. Це може включати модульне тестування, виконання інтеграційних та наскрізних тестів, верифікацію/валідацію, а також створення звітів або виявлення помилок чи дефектів у програмному вирішенні.

Життєвий цикл розробки ПЗ. Розгортання. На цьому етапі роботи програмне забезпечення розгортається у виробничому середовищі. Зібрана, спроектована, розроблена та протестована робота надається споживачам та користувачам програмного сервісу. Цей процес включає забезпечення інфраструктури всередині локального або хмарного провайдера і визначення стратегії розгортання програмного забезпечення для доставки змін замовнику.

Життєвий цикл розробки ПЗ. Експлуатація та обслуговування**.** На цьому етапі роботи програмне забезпечення вводиться в експлуатацію, щоб гарантувати відсутність проблем чи інцидентів, пов'язаних із розгортанням. Цей етап роботи може включати аналіз, розуміння і моніторинг мережевих налаштувань, конфігурацій інфраструктури та продуктивності сервісів додатків у виробничому середовищі. Цей процес може включати дозвіл або

керування інцидентами в ході будь-яких проблем або змін, внесених з метою вплинути на клієнтів або на базу користувачів.

1.3 З чого складається програмне забезпечення

Кожен проєкт має мінімум дві частини, frontend та backend, а якщо проєкт доволі великий то ще додається одна частина, це Data. Щоб краще розуміти про, що іде мова необхідно детально проаналізувати кожна ух частин і за що вона відповідає.

Frontend. Розробка зовнішнього інтерфейсу — це стиль комп'ютерного програмування, який фокусується на кодуванні та створенні елементів та функцій веб-сайту, які буде видно користувачеві. Йдеться про те, щоб переконатися, що візуальні аспекти веб-сайту є функціональними. Ви також можете думати про зовнішній інтерфейс як про «клієнтську сторону» програми. Допустимо, ви фронтенд-розробник. Це означає, що ваша робота полягає в тому, щоб кодувати та втілювати у життя візуальні елементи веб-сайту. Ви б більше зосередилися на тому, що бачить користувач, коли відвідує веб-сайт або програму. І ви хотіли б переконатись, що з сайтом легко взаємодіяти, а також він працює без збоїв.

Ці розробники беруть візуальний дизайн від дизайнерів UX та UI та пожвавлюють веб-сайт, стежачи за тим, щоб він добре працював для користувача. Один з багатьох способів використання навичок інтерфейсу користувача - це створення статичного веб-сайту, тобто веб-сайту з фіксованим вмістом, яке доставляється в браузер користувача так само, як воно зберігається. Якщо ви зустрінете просту цільову сторінку або веб-сайт для малого бізнесу, який не дозволяє користувачам виконувати будь-які інтерактивні завдання [2].

Backend. Бекенд - це серверна частина веб-сайту. Він зберігає та впорядковує дані, а також забезпечує належну роботу всього на стороні

клієнта веб-сайту. Це частина веб-сайту, яку ви не можете бачити та з якою взаємодіяти. Це частина програмного забезпечення, яка не контактує безпосередньо з користувачами. До частин і характеристик, розроблених бекенд-дизайнерами, користувачі опосередковано отримують доступ через інтерфейсний додаток. Діяльність, як-от написання API, створення бібліотек і робота з системними компонентами без користувацьких інтерфейсів або навіть систем наукового програмування, також включені до бекенда [3].

Дані**.** Робота на цій стороні ведеться в базі даних. Інженери по даним працюють с величезною купою даних та приводять ці дані в нормальний вигляд, який потім потрапляє на backend сторону.

1.4 Backend. Технології для розробки серверної частини

Backend - це програмно-апаратна частина сервісу. Це набір засобів, за допомогою яких відбувається реалізація логіки веб-сайту. Це те, що приховано від наших очей, тобто відбувається поза комп'ютером та браузером.

Як тільки ви введете запит на сторінці пошукової системи та натисніть клавішу Введення, frontend закінчиться і почнеться backend. Ваш запит відправиться на сервер, тобто за місцем розташування алгоритмів пошуку. Саме там і відбувається вся магія. Але на моніторі з'являються дані, які ви запитували, - це відбувається повернення в frontend.

Також можна сказати, що backend це процес об'єднання користувача з сервером.

Що стосується backend-розробника, він використовує будь-які інструменти, які доступні на його сервері [4].

Термін «серверні технології» може охоплювати низку програмних рішень, переважно: серверні мови сценаріїв; Системи управління базами даних (СУБД); програмне забезпечення веб-сервера, таке як Apache; та багато інших технологій залежно від створюваної програми. Необхідна комбінація

технологій, необхідних для створення служби, відома як «стек програмних рішень», а вихідний та найчастіше використовуваний стек програмних рішень для веб-служб відомий як LAMP. (Акронім для Linux, Apache, MySQL та PHP). Компоненти стека взаємозамінні, і є сотні інших скорочень, що охоплюють різні технології (наприклад, WAMP використовує Windows, MAMP використовує Mac OS X). Варто зазначити, що до стека повинні входити чотири компоненти: операційна система; екземпляр веб-сервера; система управління базами даних та серверна мова сценаріїв [5].

Вебтехнології постійно розвиваються. Так, поява HTML5 перевернула дуже багато буквально з ніг на голову. Браузери з підтримкою HTML5 - дуже мінливі, кожне оновлення вносить свої корективи. І все це вимагає сучасних інструментів для розробки інтерактивних рішень.

Одним із найбільш відомих засобів для розробників програмних рішень на JavaScript вважається платформа Node.js. Вона дуже проста і зрозуміла інтуїтивно навіть для початківців. При цьому Node.js дозволяє працювати з серверними технологіями, реалізовувати інтерактивну роботу з використанням комп'ютерних потужностей користувачів. У числі іншого, ця платформа дозволяє запускати код з командного рядка в будь-якій з поширених ОС .

Сьогодні Node по праву вважається однією з лідируючих платформ для веброзробки, і велика частина вебінструментів, серверних і клієнтських, працюють з цим інструментом.

Як, одна з найпопулярніших платформ, Node.js має декілька хороших сторін, таких, як:

- платформа Node.js широко використовує одну з найпопулярніших мов програмування JavaScript;
- платформа містить безліч стандартних бібліотек;

– велика кількість зовнішніх бібліотек та готових модулів; Саме тому, у якості технології розробки серверу було обрано платформу Node.js. Дана платформа була обрана перш за все через свою потужність, популярність, і низький поріг входу. Також великою перевагою є використання стрімко прогресуючої мови JavaScript.

Більшість застосунків на Node.JS розробляється за допомогою архітектурного шаблону MVC.

MVC (Model-View-Controller) – шаблон проектування програмної архітектури додатку, основною ідеєю якого є відділення бізнес логіки від представлення (користувацького інтерфейсу). Сутність шаблону полягає в розділенні об’єктів програми на три основні «ролі»: модель, представлення та контролер таким чином, що реалізація кожної з трьох частин може здійснюватися незалежно одна від одної [6].

Термін «компонент» в даному випадку не має нічого спільного з компонентами популярних фреймворків. В даному випадку під компонентом слідує розуміти якусь окрему частину коду , кожна з яких має одну з ролей Контролера, Моделі чи Представлення. Архітектура сервера будується на трьох частинах: Контролер, Модель та Представлення.

Сервер, працює таким чином, що вся інформація перебуває в базі даних с якою працює модель. Сама ж модель працює тільки з контролером, який являється з’єднанням між представленням та моделлю. Для розуміння, як працює патерн MVC (див. рис. 1.2).

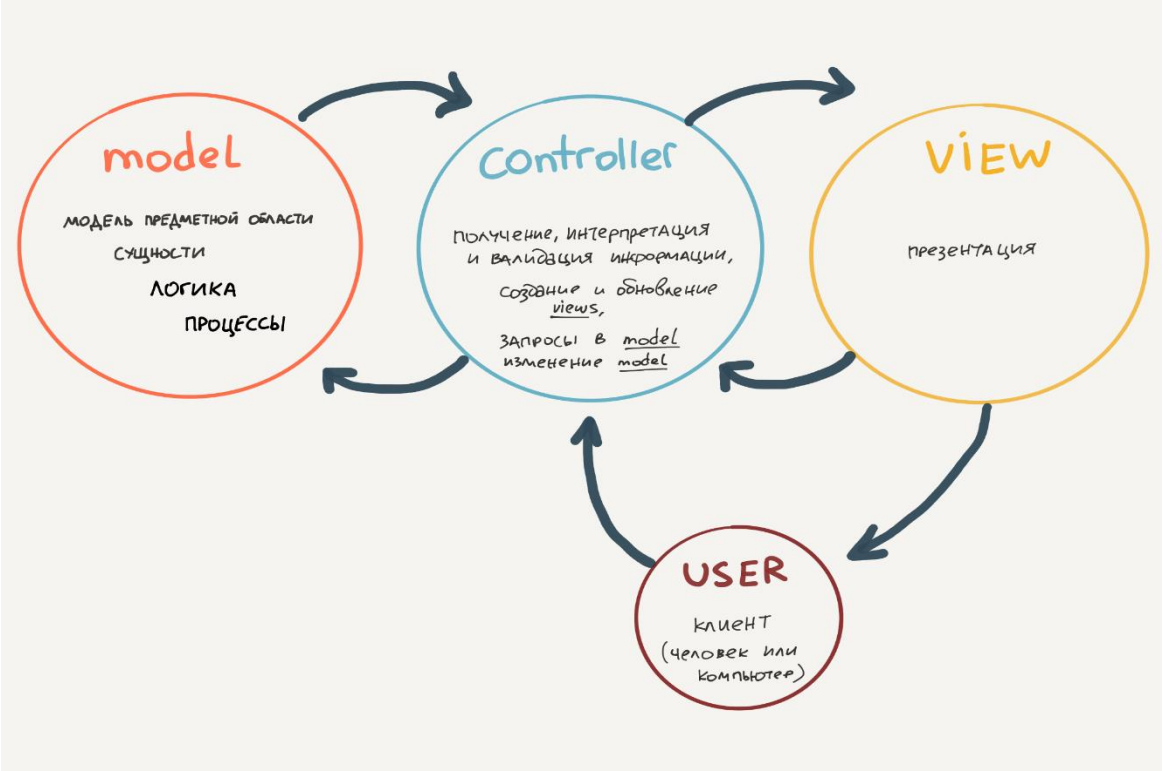

Рис. 1.2 Схема шаблону MVC.

Модель – об’єкт або сукупність об’єктів, які інкапсулюють дані та логіку доступу, обчислення та маніпуляції над ними. В ідеалі об’єкти моделі не повинні «знати» нічого про об’єкти представлення, які використовують дані з моделі, і мати з ними будь які зв’язки.

Представлення – об’єкт або сукупність об’єктів, які описують користувацький інтерфейс програми. Дані об’єкти повинні «знати» як відображати графічний інтерфейс, та реагувати на взаємодію з користувачем.

Контролер – об’єкт, який діє як посередник між одним або декількома об'єктами перегляду та одним або кількома об’єктами моделі, тобто описує логіку залежності відображення графічного інтерфейсу від даних, та маніпулює даними при взаємодії користувача з графічним інтерфейсом.

Основними перевагами використання шаблону MVC є:

– чітке відділення логіки користувацького інтерфейсу від логіки програми;

- зменшення складності великих додатків. Код виходить набагато більш структурованим, і, тим самим, полегшується підтримка, тестування і повторне використання рішень;
- легкість в адаптуванні додатку під різні пристрої, за рахунок повного відділення логіки користувацького інтерфейсу від бізнес логіки програми, що надає можливість легко замінювати дані частини;
- зручність у проектуванні додатків, у яких постійно відбуваються зміни в інтерфейсі при взаємодії з користувачем.

До недоліків шаблону, можна віднести те, що контролер практично неможливо повторно використовувати, а при розробці додатку зі складним інтерфейсом, він зазвичай розростається до великих об’ємів.

1.5 Менеджери пакетів. Огляд сфери

Менеджер пакетів - це комплекс програмного забезпечення в Linux, яке виконує установку, налаштування, видалення, а також оновлення як окремих пакетів (програм), так і всієї системи.

Пакетні менеджери спрощують використання чужого коду, надаючи цей код як незалежних модулів — пакетів. Ці пакети підключаються до свого коду за принципом чорних ящиків - це коли ми не знаємо і нам не важливо, як все влаштовано всередині цієї ящика, але ми знаємо, що він робить. Завдяки такій слабко пов'язаній архітектурі з'являється можливість легко оновлювати чужий код або замінювати один пакет іншим зі схожою функціональністю.

У кожного пакетного менеджера є файл із налаштуваннями, в якому нам потрібно вказати, від яких пакетів залежить наш код, щоб пакетний менеджер їх скачав і встановив до нас у систему. У цьому кожен пакет може залежати від інших пакетів. Пакетний менеджер розплутує цю систему залежностей і встановлює все, що потрібно, тому їх ще називають менеджерами

залежностей. Ось приклад: для роботи я використовую фреймоворк Twitter Bootstrap і для його роботи потрібний jQuery. Тому якщо я вкажу своєму менеджеру пакетів встановити Twitter Bootstrap, він автоматично встановить і jQuery [6].

На сьогодні існує велика кількість пакетних менеджерів які підходять лише під сої задачі.

NPM. найбільший у світі реєстр програмного забезпечення. Розробники з відкритим кодом з усіх континентів використовують npm для спільного використання та запозичування пакетів, і багато організацій також використовують npm для управління приватною розробкою [7].

NPM складається з двох основних частин:

- Інструмент CLI (інтерфейс командного рядка) для публікації та завантаження пакетів.
- Онлайн-сховище, яке містить пакети JavaScript.

Для більш наочного пояснення ми можемо уявити репозиторій npmjs.com як центр виконання, який отримує пакети товарів від продавців (автори пакетів npm) і роздає ці товари покупцям (користувачам пакетів npm).

Він служить онлайн-платформою . Реестр NPM, в якому люди, включаючи вас, можуть створювати, загружати, публікувати та ділитися інструментами (пакети Node.js). Ці пакети з відкритим вихідним кодом. Кожен може шукати та використовувати інструменти, опубліковані на цій онлайн-платформі (reestr NPM).

Це інструмент командної строки . Це допомагає вам взаємодіяти з онлайн-платформою, що я тільки що упомянув. Несколько речей, які можна зробити за допомогою інструментів командної строки, які включають встановлення та видалення пакетів.

Yarn. Yarn — це менеджер пакетів, який одночасно виконує роль менеджера проекту. Незалежно від того, чи працюєте ви над одноразовими

проектами чи великими монорепозиторіями, як любитель чи корпоративний користувач, ми допоможемо вам [8].

Bower. Bower — це менеджер пакетів, такий як npm, який керує фреймворками, бібліотеками, активами та утилітами, встановлює їх та слідкує за їх актуальністю.

Традиційно багато проектів веб-розробки об'єднували npm та Bower. npm використовувався для управління внутрішніми залежностями, а Bower використовувався для зовнішніх залежностей. Насправді, вам потрібно було використовувати npm, щоб встановити Bower в першу чергу.

Основна перевага Bower перед npm полягала в тому, що він мав плоский графік залежностей. npm відстежує залежність пакетів і може автоматично встановлювати тисячі залежностей і подзависимостей, включаючи безліч дублікатів однієї й тієї ж пакета. Як ви можете уявити, це не дуже добре для інтерфейсних проектів, оскільки може призвести до дуже великого корисного навантаження.

З іншого боку, Бауер надав користувачеві можливість керувати залежностями. Наприклад, якщо в проекті було багато бібліотек, що залежать від jQuery, користувач міг вирішити, яку версію jQuery встановити, і вказати цю версію як залежність для інших бібліотек [9].

Пакетні менеджери є дуже популярними в наш час тому що роблять розробку простішою та пришвидшують процес.

1.5.1 Аналіз існуючих аналогів npm модулів

Для того, щоб розуміти, як краще потрібно побудувати архітектуру свого програмного забезпечення, а саме npm модуля для побудови архітектури проєкту, необхідно проаналізувати існуючі аналоги та як вони працюють.

На сьогоднішній день, npm модулі створені мабуть на всі випадки життя, а саме:

– для роботи із датами;
– для роботи із даними;
– для створення файлів із різними розширеннями (word, pdf, csv);
– для роботи із http;
– да багато інших;

Існує велика кількість npm модулів із відкритим вихідним кодом. За аналоги які необхідно проаналізувати були взяті такі модулі як:

– npm модуль «Jwt»;
– npm модуль «Socket.io»;

Npm модуль Socket.io. Socket.io – npm модуль який забезпечує двосторонній зв'язок на основі подій у реальному часі. Модуль складається з:

– сервер Node.js (цей репозиторій);
– бібліотека Java – клієнт для браузера (або клієнт Node.js);

Також цей модуль має реалізації не лише для платформи Node.js а також для таких мов, як java, C++, Python, .Net.

Його основні особливості:

Надійність. З'єднання встановлюються навіть за наявності:

– проксі та балансувальників навантаження;
– персональних між-мережевих екранів та антивірусного програмного забезпечення;

Для цього він покладається на Engine.IO, який спочатку встановлює з'єднання з тривалим опитуванням, а потім намагається перейти на більш досконалі транспорти, які тестуються на стороні, наприклад, WebSocket. Для отримання додаткових відомостей див. розділ « Цілі ».

– підтримка автоматичного перепідключення**.** Якщо не вказано інше, вимкнений клієнт намагатиметься повторно підключитися назавжди, доки сервер знову не стане доступним;

– виявлення відключення. На рівні Engine.IO реалізований механізм контрольних сигналів, що дозволяє як серверу, і клієнту знати, коли інший більше відповідає;

Ця функціональність досягається з таймерами, встановленими як на сервері і клієнті, зі значеннями тайм-аута, що спільно використовуються під час з'єднання. Ці таймери вимагають, щоб всі наступні клієнтські виклики направлялися на один і той же сервер, отже, ця sticky-session вимога потрібна при використанні декількох вузлів [11].

Npm модуль Jwt. Це модуль якій необхідний для авторизації користувача програмного забезпечення. Аутентифікація дозволяє додатку знати, що людина, яка надсилає запит додатку, насправді той, ким вона себе називає. Веб-токен JSON (JWT) - це один із методів автентифікації без фактичного зберігання будь-якої інформації про користувача в самій системі.

JWT складається із 3 частин а саме:

– Заголовок. Заголовок визначає таку інформацію, як алгоритм, який використовується для генерації підпису. Ця частина досить стандартна і однакова для будь-якого JWT, який використовує той самий алгоритм.
– Корисне навантаження. Ця містить інформацію про додаток. Додаток може містити різну інформацію, яку може відправляти користувач, а також інформацію про закінчення терміну дії та дійсності токена.
– Підпис. Підпис створюється шляхом об'єднання та хешування перших двох частин разом із секретним ключем [12].

1.6 Постановка задачі

Метою МКР – є розробка та публікація npm модуля, який буде доступний кожному розробнику для поліпшення процесу розробки проєктів на платформі node.js.

Цей npm модуль має містити в собі:

– backend частина. Сам модуль написаний за допомогою мови програмування JavaScript;
– frontend частина. Сторінка вибору компонентів проєкту, які розробник хоче використовувати;

В результаті ми маємо отримати легкий у використанні npm модуль який буде створювати проєкт із готовою структурою, яку вибере користувач на етапі створення.

Висновки до розділу 1

У цьому розділі було проаналізовано предметну сферу створення ПЗ. Проаналізовано етапи розробки програмного забезпечення та з чого складається кожен застосунок. Розглянуто новітні технології, які використовуються для розробки серверної та клієнтської частини, та як ці технології співпрацюють. Також було розглянуто сфери менеджерів пакетів для розробки програмного забезпечення та їх допомогу під час розробки.

2 ТЕХНОЛОГІЇ ТА ПІДХОДИ ДЛЯ СТВОРЕННЯ ДОДАТКУ

Для успішної реалізації проекту та подальшого його просування, об’єкт проектування повинен бути на сам перед детально описаний, побудовані функціональні та інформаційні моделі додатку. За досвідом який набутий при виконанні МКР показує, що розробка такого проекту трудомістка і довготривала робота, яка вимагає високої кваліфікації від розробників, які займаються розробкою проекту. Дуже велику роль під час розробки проєкту грає – архітектура.

2.1 Типи архітектури які використовуються в розробці програмного забезпечення.

Архітектура програмного забезпечення - сукупність рішень щодо організації програмної системи [12].

Архітектура включає:

- вибір структурних елементів та його інтерфейсів, з допомогою яких складено система, і навіть їх поведінки у межах співробітництва структурних елементів;
- з'єднання обраних елементів структури та поведінки, у все більші системи;
- архітектурний стиль, який спрямовує всю організацію - всі елементи, їх інтерфейси, їх співробітництво та їх з'єднання;

2.1.1 Багатошарова архітектура

Цей підхід ділить відповідальність між шарами які лежать один на одному. Архітектура ділить програмне забезпечення на 3 шари, а саме:

- шар представлення (Presentation Layer) містить інтерфейс користувача і відповідає за забезпечення хорошого користувальницького досвіду;

– шар бізнес-логіки (Business Logic Layer), як випливає з назви, містить бізнес-логіку програми. Він відокремлює UI/UX від обчислень, пов'язаних із бізнесом. Це дозволяє легко змінювати логіку залежно від постійно мінливих бізнес-вимог, ніяк не впливаючи на інші шари;
– шар передачі даних (Data Link Layer) відповідає за взаємодію Космосу з постійними сховищами, такими як бази даних, та іншу обробку інформації, яка не пов'язана з бізнесом;

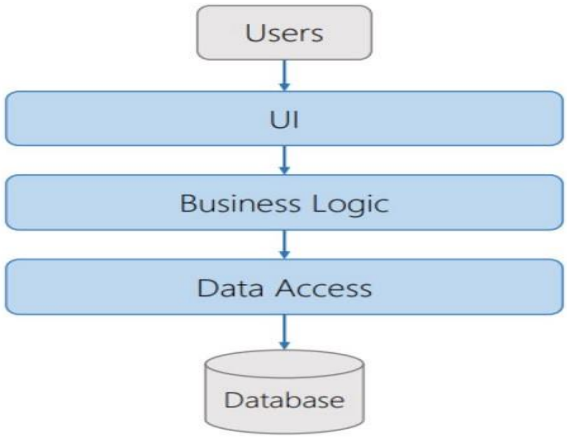

Рис. 2.1 Схема багатошарової архітектури.

Переваги**:**

– простіша реалізація порівняно з іншими підходам;
– пропонує абстракцію завдяки розподілу відповідальності між рівнями;
– ізолювання захищає одні шари від інших змін;
– підвищує керованість програмного забезпечення з допомогою слабкої пов'язаності;

Недоліки:

– не пропонує великої масштабованості;
– програмне забезпечення, створене з таким підходом, матиме монолітну структуру, що ускладнює внесення модифікацій;
– дані повинні проходити по кожному шару, навіть якщо не потрібно передавати їх з певних шарів;

2.1.2 Багаторівнева архітектура

Це клієнт-серверна архітектура, в якій поділяються функції представлення, обробки та зберігання даних. Такий архітектурний патерн пом'якшує зростаючу складність додатків, спрощує і робить більш гнучким їх доопрацювання.

Така архітектура складається з різних рівнів, кожен із яких відповідає окремій функції. Таким чином, вносити зміни до кожного окремого рівня простіше, ніж займатися всією архітектурою. Розробники можуть створювати гнучкі та повторно використовувані програми. Це значно спрощує весь процес керування системою.

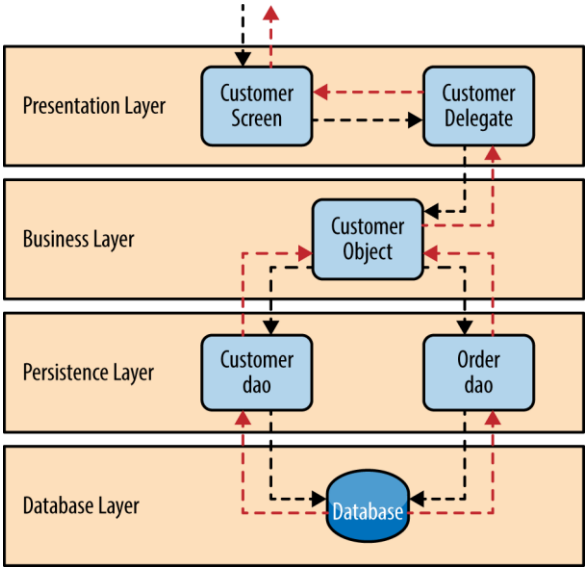

Рис. 2.2 Схема багаторівневої архітектури.

Переваги:

- багаторівнева архітектура підвищує гнучкість, зручність супроводу та масштабованість;
- декілька програм можуть повторно використовувати компоненти;
- легший спосіб тестування компонентів незалежно один від одного;

Недоліки:

- розробка ресурсоємних програм іноді може зайняти більше часу;
- ускладнює прості додатки;

2.1.3 Сервіс-орієнтована архітектура

Це стиль розробки програмного забезпечення. У концепції SOA послуги надаються ззовні іншим компонентам як компоненти програми через протокол зв'язку через мережу. Основний принцип SOA не залежить від технологій, продуктів та постачальників.

Переваги**:** масштабованість; надійність; незалежність від інших сервісів;

Недоліки**:** висока вартість;

2.1.4 Мікросервісна архітектура

При такому підході додаток розробляється як набір невеликих сервісів, кожен з яких працює у власному процесі та зв'язується з легковажними механізмами, як правило, API для HTTP-ресурсу.

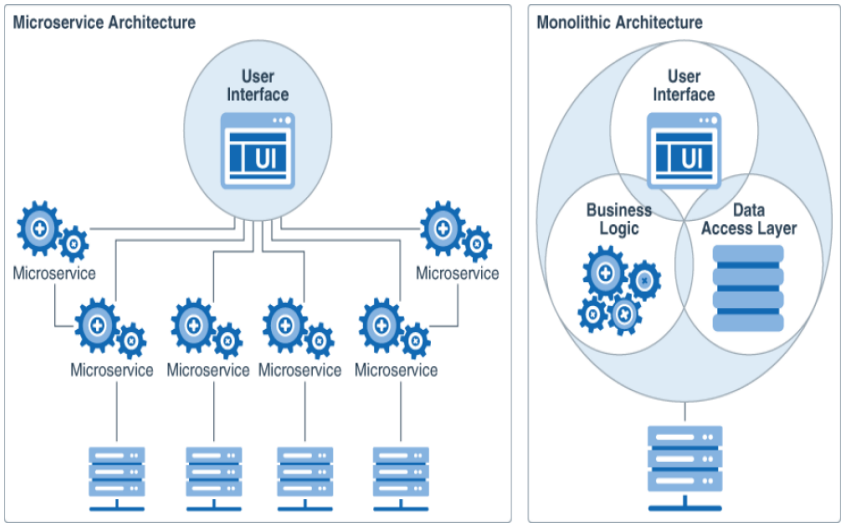

Рис. 2.3 Схема мікросервісної архітектури.

Ці сервіси ґрунтуються на бізнес-можливостях і можуть розвиватися незалежно один від одного за допомогою повністю автоматизованого механізму.

Централізоване керування між сервісами мінімальне. Вони можуть бути написані різними мовами, використовувати різні технології зберігання даних.

Архітектура працює за принципом компонентизації послуг. Вона поділяє програмне забезпечення на різні ізольовані компоненти (сервіси), кожен із яких несе єдину відповідальність. Зміни в одному сервісі не повинні торкатися інших.

Переваги:

– пропонує слабку пов'язаність завдяки високому ступеню ізоляції.
– підвищує модульність;
– збій в одному сервісі не торкнеться всієї системи, оскільки вони ізольовані;
– пропонує високу гнучкість та масштабованість;
– простота модифікації може прискорити ітерацію;
– дозволяє реалізувати покращену систему обробки помилок;
– вирішує проблеми з потоками даних, які бувають багатошаровою архітектурою;

Недоліки:

– підвищений ризик збою під час обміну даними між сервісами.
– великою кількістю сервісів важко керувати;
– вимагає вирішення таких проблем, як затримки в мережі, балансування навантаження та інші труднощі, властиві розподіленій архітектурі;
– потребує комплексного тестування в розподіленому середовищі;
– на реалізацію потрібно набагато більше часу;

2.1.5 Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA) – тип архітектури, який пропонує метод багаторазового використання програмних компонентів з допомогою інтерфейсів сервісів. Ці інтерфейси використовують загальні комунікаційні

стандарти, щоб їх можна було швидко включити до нових програм, не вдаючись щоразу до глибокої інтеграції.

Ця архітектурна модель складається з компонентів та додатків, що зв'язуються один з одним за допомогою чітко визначених сервісів.

Вона складається з 5 елементів:

– послуги (Services);
– сервісна шина (Service Bus);
– сервісний репозиторій (Service Repository catalogue of services);
– безпека SOA (SOA Security);
– керування SOA (SOA Governance).

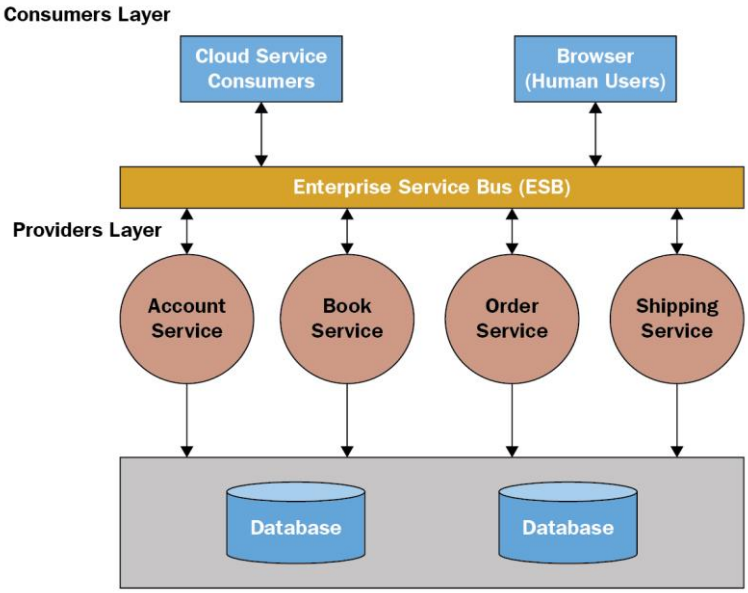

Рис. 2.4 Схема сервіс-орієнтованої архітектури.

Переваги:

– незалежність від обраних технологій;
– незалежність від особливостей передачі/зв'язку;

Недоліки:

– заблоковані канали зв'язку;
– складна та неоднозначна специфікація;

2.2 Вибір стеку технологій

Вибір стеку технологій є дуже важливим етапом та складовою під час розробки програмного забезпечення, бо від цього не рідно залежить шлях та складність самої розробки та написання коду.

Технологічний стек - це екосистема даних, що включає низку ключових інструментів, бібліотек, фреймворків, мов програмування та технологій, які працюють разом для розробки та запуску програмного продукту чи послуги. Наприклад, повномасштабний технічний стек Facebook включає Cassandra, PHP, Hadoop, React, Swift, GraphQL та низку інших фреймворків.

У технологічному стеку є два однаково важливі елементи: інтерфейс (на стороні клієнта/користувача) та бекенд (на стороні сервера). У той час як інтерфейсні технології складаються з JavaScript, CSS, HTML, бібліотек інтерфейсу користувача і фреймворків, серверні технології включають сервери і операційні системи, мови програмування, веб-фреймворки і бази даних.

Вибір застосовуваного технологічного стека - це важливе рішення, яке необхідно ухвалити перед запуском програмного проекту. Впливи такі:

– як працює програмне забезпечення;
– як воно працюватиме у майбутньому;
– масштабованість програмного забезпечення;
– ціль проекту;
– вибір програми та бази даних, локального та хмарного;
– можливості серверних систем;

Існує велика кількість стеків технологій, але як за стек для розробки npm пакету був обраний стек PETN.

PEJN – це стек який складається з таких технологій, як:

– postgresQL (база даних MySQL);
– express.js (бекенд фреймворк);

– javascript (мова програмування);
– node.js (платформа для побудови і виконання мови typescript);

PostgreSQL. Це потужна об'єктно-реляційна база даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з безліччю функцій, що забезпечують безпечне зберігання та масштабування найскладніших робочих навантажень з даними. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES у Каліфорнійському університеті в Берклі та понад 30 років активно розвиваються на базовій платформі.
PostgreSQL заробив міцну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та прихильності спільноти відкритого вихідного коду, що стоїть за програмним забезпеченням, для послідовної доставки ефективних та інноваційних рішень. PostgreSQL працює у всіх основних операційних системах, відповідає вимогам ACID з 2001 року та має потужні надбудови, такі як популярний розширювач гео-просторових баз даних PostGIS. Не дивно, що PostgreSQL стала реляційною базою даних з відкритим вихідним кодом, яку вибирають багато людей та організацій [13].

Node.js**.** Це платформа, побудована на середовищі виконання JavaScript Chrome для простого створення швидких та масштабованих мережних додатків. Node.js використовує керовану подіями неблокуючу модель вводу-виводу, яка робить його легким і ефективним, що ідеально підходить для додатків з інтенсивним використанням даних у реальному часі, які працюють на розподілених пристроях. Node.js - це кросплатформове середовище виконання з відкритим вихідним кодом для розробки серверних та мережевих програм. Програми Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js у OS X, Microsoft Windows та Linux [14].

Node.js також надає багату бібліотеку різних модулів JavaScript, яка значно спрощує розробку веб-додатків з використанням Node.js.

В node.js є багато переваг.

Асинхронний та керований подіями – всі API бібліотеки Node.js є асинхронними, тобто неблокуючими. По суті це означає, що сервер на основі Node.js ніколи не чекає, поки API поверне дані. Сервер переходить до наступного API після його виклику, і механізм повідомлення подій Node.js допомагає серверу отримати відповідь від попереднього виклику API.

Дуже швидкий – бібліотека Node.js, створена на базі ядра Google Chrome V8 JavaScript Engine, який дуже швидко виконує код.

Однопотоковий, але добре масштабований - Node.js використовує однопоточну модель із циклом подій. Механізм подій допомагає серверу реагувати неблокуючим чином і забезпечує високу масштабованість сервера, на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує однопоточну програму, і та сама програма може обслуговувати набагато більше запитів, ніж традиційні сервери, такі як HTTP-сервер Apache.

Без буферизації – програми Node.js ніколи не буферизують дані. Ці програми просто виводять дані частинами.

2.3 Вибір СУБД

База даних – це інформація, яка налаштована для легкого доступу, керування та оновлення. Комп'ютерні бази даних зазвичай зберігають сукупність записів даних або файлів, що містять таку інформацію, як транзакції продажів, дані про клієнтів, фінансові показники та інформацію про продукти.

Бази даних використовуються для зберігання, обслуговування та доступу до будь-яких даних. Вони збирають інформацію про людей, місця або речі. Ця інформація збирається в одному місці, щоб її можна було спостерігати та аналізувати. Бази даних можна як організований набір інформації.

Компанії використовують дані, що зберігаються в базах даних, для прийняття обґрунтованих бізнес-рішень. Деякі способи, якими організації використовують бази даних.

Поліпшити бізнес-процеси. Компанії збирають дані про бізнес-процеси, такі як продаж, обробка замовлень та обслуговування клієнтів. Вони аналізують ці дані, щоб покращити ці процеси, розширити свій бізнес та збільшити дохід.

Слідкуйте за клієнтами. Бази даних часто зберігають інформацію про людей, таких як клієнти чи користувачі. Наприклад, платформи соціальних мереж використовують бази даних для зберігання інформації про користувачів, таких як імена, адреси електронної пошти та поведінка користувачів. Ці дані використовуються для рекомендації контенту користувачам та покращення взаємодії з користувачем.

Захист особисту інформацію щодо здоров'я. Постачальники медичних послуг використовують бази даних для безпечного зберігання особистих даних про здоров'я, щоб інформувати та покращувати догляд за пацієнтами.

Зберігати особисті дані. Бази даних також можуть бути використані для зберігання особистої інформації. Наприклад, особисте хмарне сховище доступне для окремих користувачів для зберігання мультимедіа, наприклад фотографій, у керованій хмарі.

2.3.1 Порівняння популярних типів баз даних

На сьогодні існує велика кількість різних баз даних, які відрізняються одне від одної дуже сильно а іноді не відрізняється взагалі [15].

Таблиця 1.1

Порівняння найпопулярніших типів баз даних

	Тип ДБ	Ліцензія	Документ-тація	Масштаб-ованість	Тип даних	Скл-ть вивчення
MySQL	SQL	Ліц. на GNI	Нормальна	вертикальний комплекс	Структурований, напівструктурований	Середня
Maria DB	SQL	Ліц. на GNI	Чудова	вертикальний	Структурований, напівструктурований	Середня
Oracle	Мульти-модель, SQL	Власність	Чудова	вертикальний	Структурований, напівструктурований, неструктурований	Складна
PostgresQL	Об'єктно-орієнтована модель. SQL	Відкрита	Нормальна	вертикальний	Структурований, напівструктурований, неструктурований	Складна
MSSQL Server	T-SQL	Власність	Чудова	вертикальний комплекс	Структурований, напівструктурований, неструктурований	Складна
MongoDB	NoSQL, документо орієнтована модель	SSPL	Чудова	Горизонтальна	Структурований, напівструктурований, неструктурований	Середня
Redis	NoSQL	Відкрита, BSD 3-клас	Чудова	Горизонтальна	Структурований, напівструктурований, неструктурований	Середня
Cassandra	NoSQL	Відкрита	Чудова	Горизонтальний	Структурований, напівструктурований, неструктурований	Складна
Elastic-Serch	NoSQL, документо орієнтована модель	Відкрита	Нормальна	Горизонтальний	Структурований, напівструктурований, неструктурований	Складна

Як ми можемо бачити, існує велика кількість баз даних, які мають свої переваги та недоліки, але була обрана postrgresQL через свою гнучкість та легкість використання.

Висновки до розділу 2

У цьому розділі було детально розглянуто та проаналізовано найпопулярніші типи архітектурних патернів. Також було проаналізовано та порівняно існуючі, популярні типи СУБД та обрано postgresQL.

Було проаналізовано технології які б підійшли для реалізації та розробки проекту та проаналізовано існуючі типи пакетних менеджерів які використовуються для платформи Node.js.

3 ОПИС ЗАСТОСУНКУ. ПРОГРАМНА РЕАЛІЗАЦІЯ

Перед розробкою будь-якого застосунку, потрібно поставити чітку мету та головні вимоги для того, щоб застосунок став зручним та користувався популярністю у користувачів.

Метою розробки npm модуля для побудови архітектури проєкту, безкоштовне та швидке, зрозуміле використання. Перед розробкою будь-якого застосунку, потрібно поставити чітку мету та головні вимоги для того, щоб застосунок став зручним та користувався популярністю у користувачів.

3.1 Створення дизайну frontend частини

Дизайн сайту – це оформлення контенту, сукупність всіх графічних елементів на вебсторінці. Раніше під вебдизайном розуміли виключно візуальне оформлення, але тепер на перший план вийшло зручність користувача, тому до завдань вебдизайнера додалися, аналітика і грамотне структурування інформації на сайті.

Дизайн сайту грає велике значення в процесі його просування та розкрутки. Перший контакт відвідувача з сайтом здійснюється на основі візуального сприйняття сторінки. Якщо зовнішній вигляд сайту буде приємним і цікавим, то користувач затримається тут і захоче більше дізнатися про вміст вебресурсу.

Дизайн сайту та його якісне оформлення – це показник іміджу компанії, вираз її відносини до потенційних користувачів. Якщо сайт виглядає красиво і оригінально, значить, його власникові важливо справити хороше враження на відвідувачів, і він зацікавлений у вашій увазі і оцінці.

Розробка дизайну сайту повинна здійснюватися за особливими правилами. Використовуючи сучасні технології можна створити якісний сайт, акуратно і красиво оформлений, приємний для очей. Що стосується азів, то тут важливо врахувати вагу графічних файлів, оскільки при високій вазі графічних

об’єктів, сторінки сайту будуть довго завантажуватися, а це негативно відіб'ється на відвідуваності ресурсу. Не кожен користувач захоче чекати кілька хвилин до повного завантаження сторінки.

Для розробки макетів та дизайнів різного типу, існує велика кількість графічних редакторів, таких як Photoshop, Figma або Sketch. В даному випадку було використано графічний редактор такий як Adobe XD.

Adobe XD – програма від компанії Adobe. Чудово підходить для проектування та презентацій. Також широко використовується в сфері вебпрограмування [16].

Також Adobe XD являється потужною та простою платформою для командної роботи. Допомагає створювати різні проекти, такі як проекти для вебсайтів, мобільних застосунків, голосових інтерфейсів та багато іншого.

Найбільшим конкурентом для Adobe XD являється такий графічний редактор , як Sketch. Проте Sketch з самого початку був розроблений під операційну систему Mac OS, а Adobe XD підходить під різні системи.

В Adobe XD, ми одразу можемо обрати роздільну здатність дисплею, під який ми збираємося розробляти макет (див. рис. 3.1). На вибір нам дається, розробка макета під пристрої з роздільною здатністю 375x812 пікселів, тобто розробка буде виконуватися під мобільні пристрої.

Другим варіантом є розробка проектів під дисплеї з роздільною здатністю 768x1024 пікселі, тобто під планшетні пристрої.

Також на вибір дається розробка для вебпроектів роздільною здатністю 1920x1080 пікселів.

На останок, на вибір нам дається розробка проектів де ми можемо самі обрати розмір дисплею під який будемо розробляти проект.

В даному випадку було обрано роздільну здатність дисплею 1920x1080 та , як створення та розробка проекту ведеться під браузери.

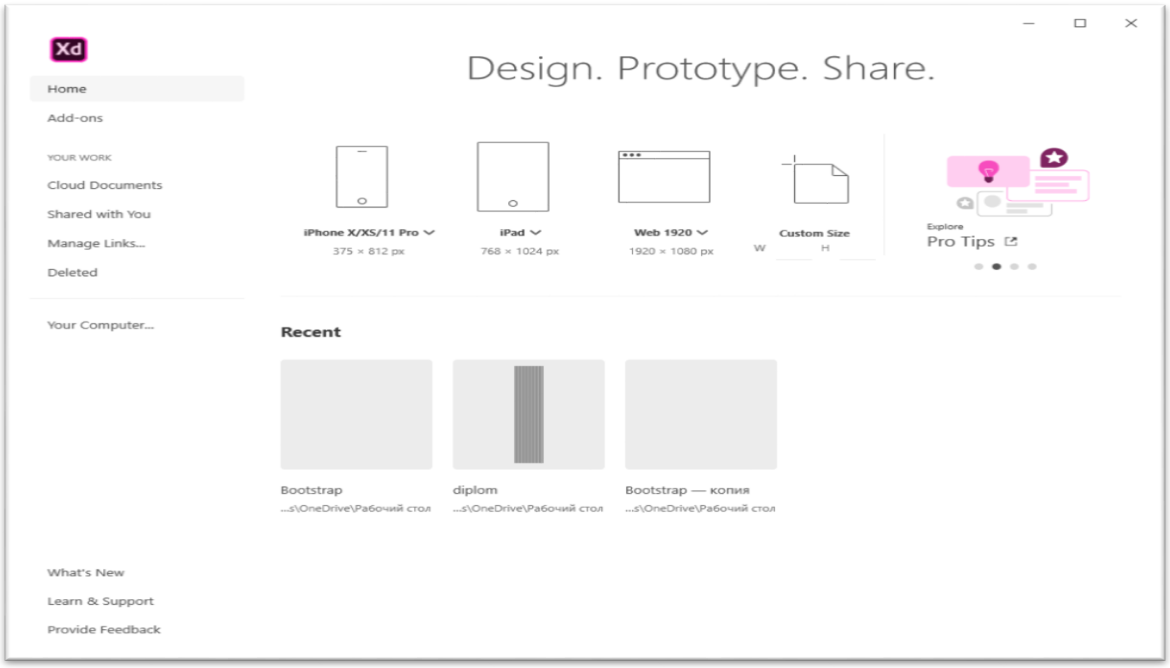

Рис. 3.1 Головна сторінка редактору Adobe XD.

Adobe XD з коробки, дає багато можливостей для комфортної розробки макету.

З лівого боку знаходиться панель компонентів, які допомагають створити макет. З правого боку знаходиться панель редагування, наприклад, коли ми обираємо якийсь компонент, панель стає активною і ми можемо змінити такі параметри як колір, розмір, положення на шарі. В разі, якщо ми хочемо редагувати тест, то ми можемо змінити колір, положення, розмір шрифту та змінити сам шрифт.

Клієнтська частина модуля має містити що найменше 3 сторінку, а саме сторінку реєстрації, сторінку аутентифікації та сторінку вибору модулів які мають бути завантажені разом з самим створення проєкту.

Макет Сторінки реєстрації (див. рис. 3.2). Сторінка реєстрації має нескладну форму, в якій користувач має ввести:

– ім’я;
– логін;
– пароль (двічі);

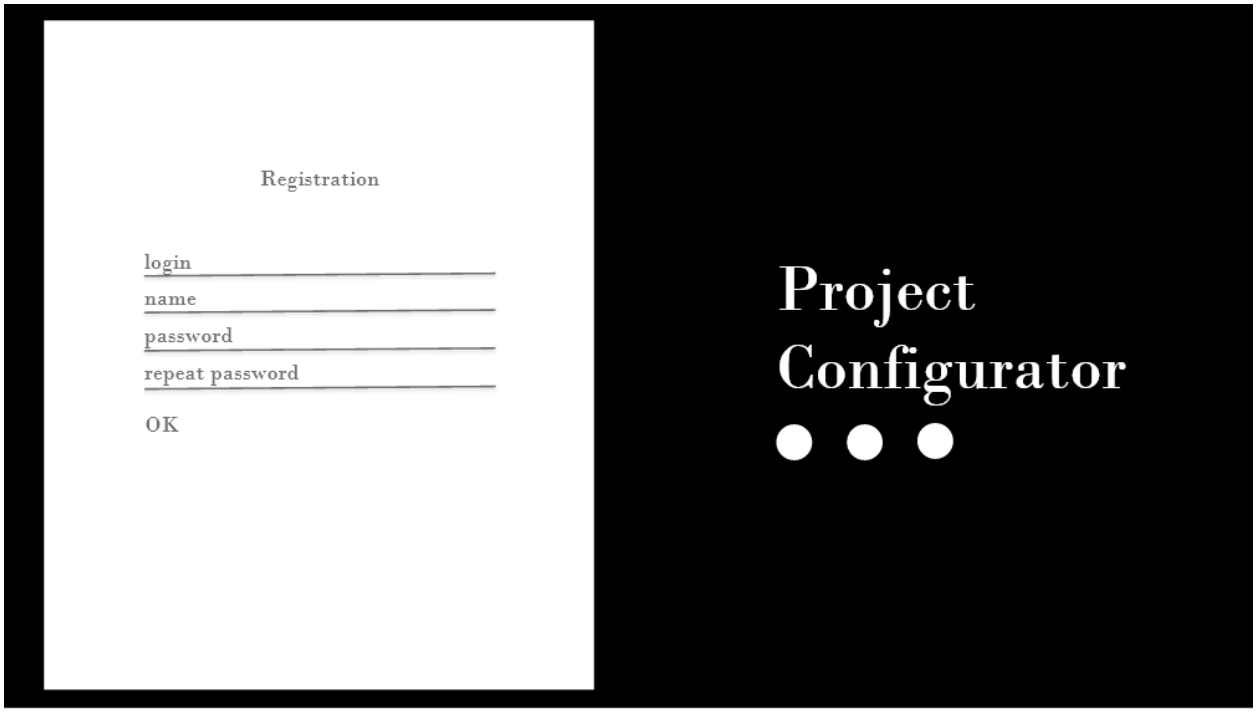

Рис. 3.2 Макет сторінки реєстрації.

Після того, як користувач зареєструється, відбудеться редірект на сторінку аутентифікації (див. рис. 3.3). На цій сторінці користувач повинен буде ввести:

– логін;
– пароль;

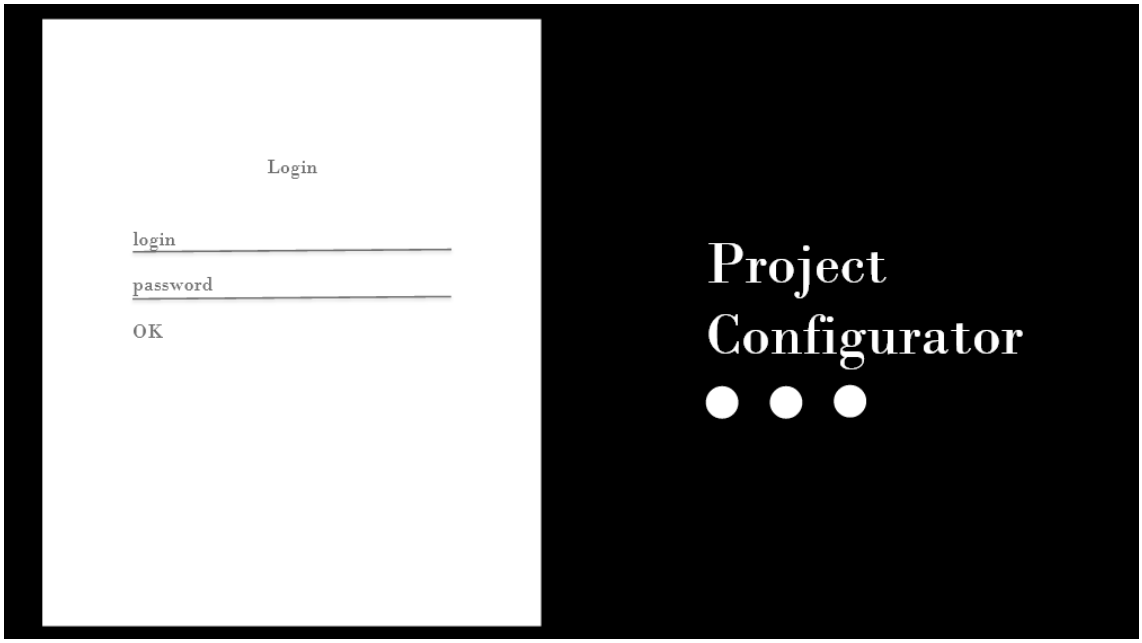

Рис. 3.3 Макет сторінки аутентифікацуії.

Після того як користувач заповнить форму користувача, і якщо пройде валідація форми, то виконається редірект на сторінку з вибором модулів для проєкту.

Макет головної сторінки (див. рис. 3.4.) тобто сторінка вибору конфігурації проєкту має:

– поле для вибору бази даних;
– поле для вибору стилізації коду;
– поле для вибору докера;
– поле для вибору методів тестування;
– поле для вибору допоміжних модулів;

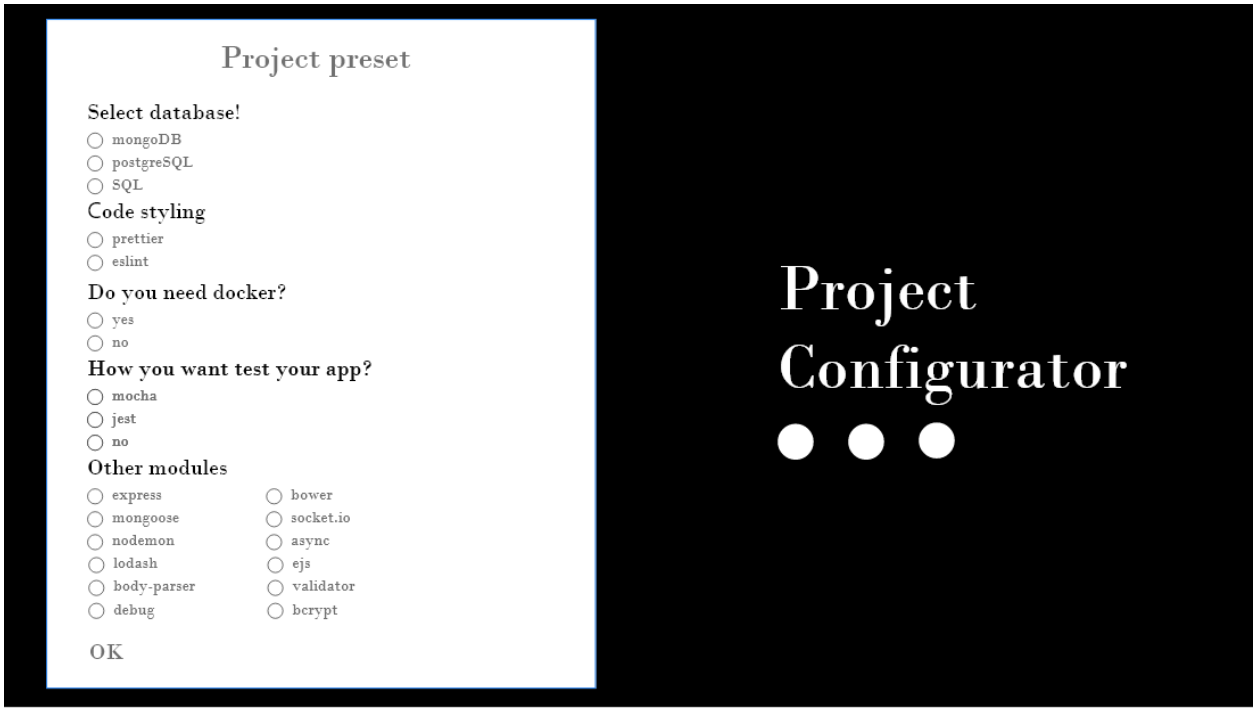

Рис. 3.4 Макет сторінки вибору конфігурації для проєкту.

Із рис. 3.2. ми може бачити макет головної сторінки, на якому знаходиться форма , в яку користувач повинен обрати компоненти які він хоче використовувати під час написання проєкту. Після того як користувач обире конфігурацію, об’єкт з обраними даними буде автоматично відправлено на backend частину.

3.2 Налаштування середовища розробки

Налаштування середовища розробки є важливою частино під час розробки проєкту, то му, що добре налаштоване середовище розробки може зберегти багато часу та зробити процес розробки продукту більш зручним.

3.2.1 IDE WebStorm. Налаштування

Огляд WebStorm. WebStorm — це інтегроване середовище розробки для кодування в JavaScript і пов’язаних з ним технологій, включаючи TypeScript, React, Vue, Angular, Node.js, HTML і таблиці стилів. Так само, WebStorm робить досвід розробки більш приємним, автоматизуючи рутинну роботу та допомагаючи з легкістю справлятися зі складними завданнями [12].

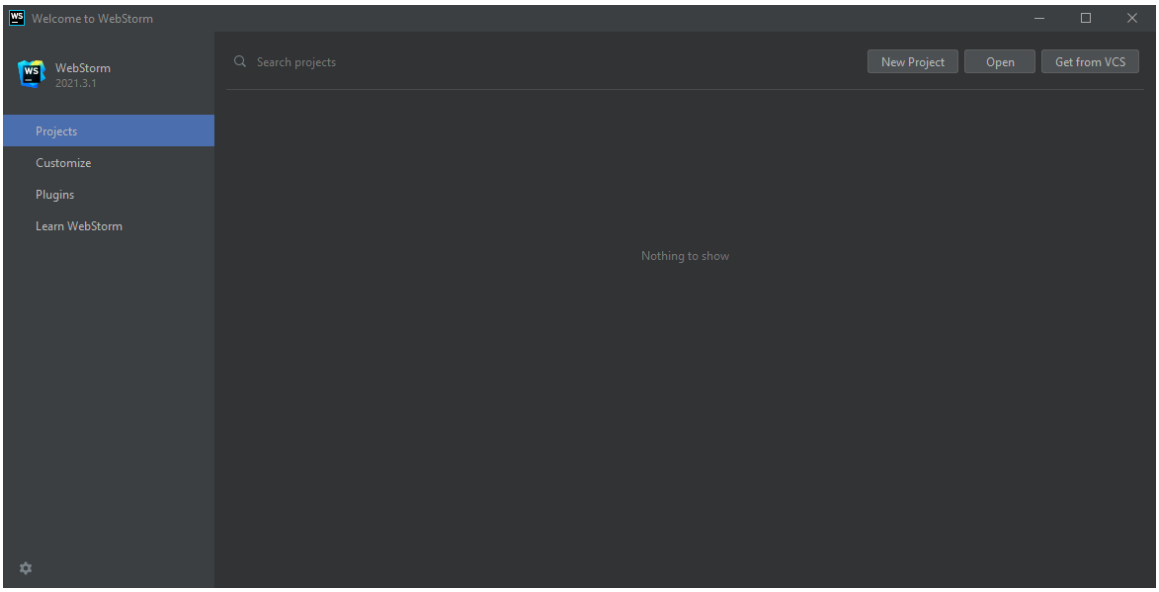

Рис. 3.3 Початкова сторінка IDE WebStorm.

На рисунку 3.3 ми можемо бачити сторінку яку користувач може бачити одразу після ввімкнення програми.

На цій сторінці ми можемо:

– створити проєкт;
– змінити зовнішній вигляд ide;
– скачати додаткові плагіни;

– документація WebStorm;
– загальні налаштування;

Для створення проєкту необхідно перейти на вкладку projects та обрати тип проєкту який користувач хоче розробляти див. рис. 3.4.

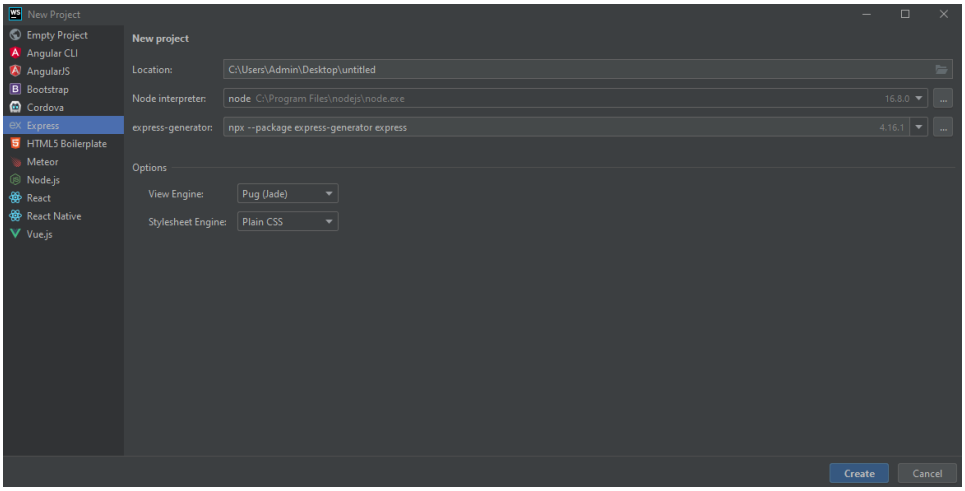

Рис. 3.4 Сторінка створення проєкту.

На сторінці створення проєкту ми можемо бачити:

– тип проєкту;
– місце зберігання проєкту;
– версія node;
– версія express;

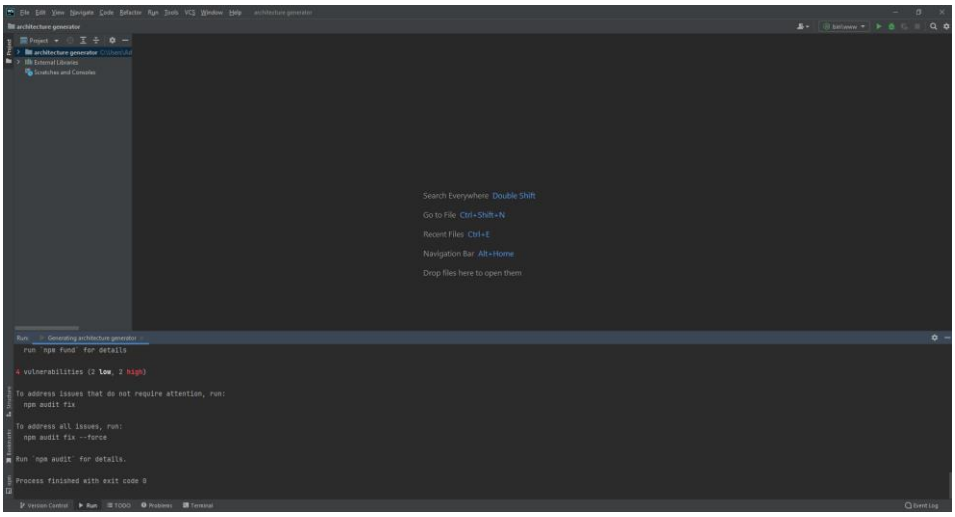

Рис. 3.5 Сторінка створення проєкту.

На рис. 3.5 ми можемо бачити:

– панель налаштувань;

– панель проєкту;

– консоль;

Плагіни для WebStorm. Плагін — це програмне забезпечення, яке встановлюється на програму, розширюючи її можливості. Наприклад, якщо ви хочете подивитися відео на веб-сайті, вам може знадобитися плагін для цього. Якщо плагін не встановлено, ваш браузер не зрозуміє, як відтворити відео.

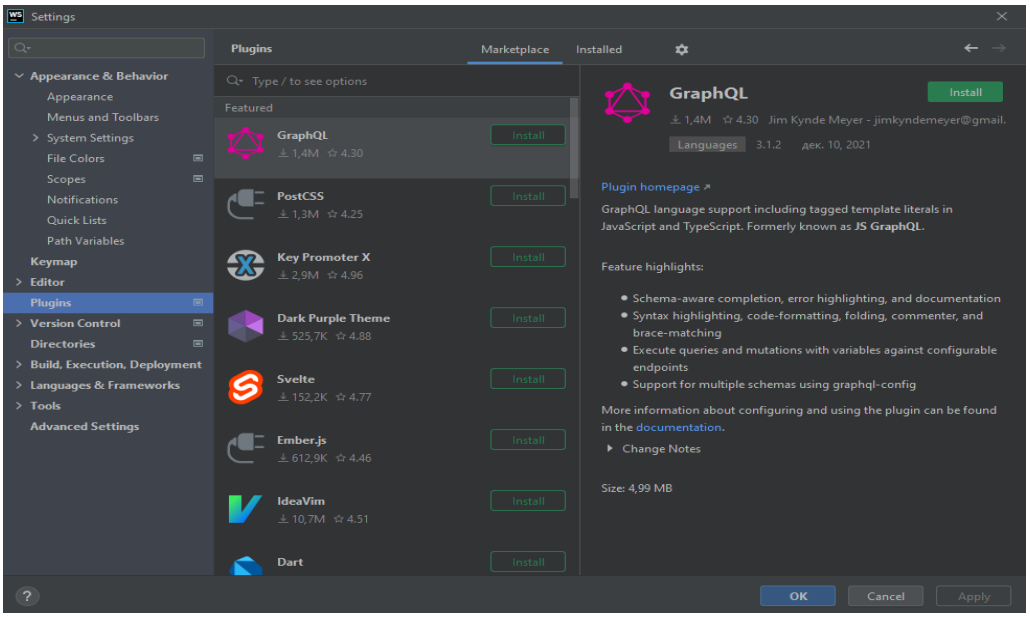

Рис. 3.6 Сторінка завантаження плагінів.

Rainbow brackets. Це чудовий плагін для IDE, заснований на IntelliJ IDEA, який розфарбовує фігурні, квадратні, круглі та кутові дужки різним кольором відповідно до їх пари [18].

Key Promoter X**.** Цей плагін допомагає отримати необхідні ярлики під час роботи над проектами. Коли розробники використовують мишу на кнопці всередині IDE, Key Promoter X представляє комбінацію клавіш, яку ви повинні були використовувати в якості альтернативи. Key Promoter X пропонує простий спосіб вивчити, як замінити виснажливу роботу миші на клавіші клавіатури, і допомагає перейти до швидшої розробки без використання миші. Вікно інструмента Key Promoter X містить список дій миші, які найчастіше

використовуються розробниками, і швидко надає ярлик, який розробники можуть використовувати в якості альтернативи. Кнопки, які не мають ярлика, Key Promoter X підказує з можливістю безпосереднього створення [18].

Material Icon Theme. Розширення, що допомагає встановити дизайн іконок для директорій та файлів, що дає більшу зручність під час роботи. Також, автоматично ставиться виділення, а саме зручно бачити де дочірні директорії, а де батьківські директорії [19].

3.2.2 NodeJS. Установка та налаштування.

NodeJS - це платформа розробки з відкритим вихідним кодом для виконання коду JavaScript на стороні сервера. Node корисний для розробки додатків, які вимагають постійного підключення браузера до сервера, і часто використовується для додатків у режимі реального часу, таких як чат, стрічки новин та веб-повідомлення [18].

Завантажити платформу nodejs можна с офіційного сайту. Перейшовши на офіційний сайт ми бачимо сторінку для завантаження платформи [20]:

Рис. 3.7 Сторінка завантаження Node.JS.

Після завантаження платформи, можна буде відкрити командний рядок node.js.

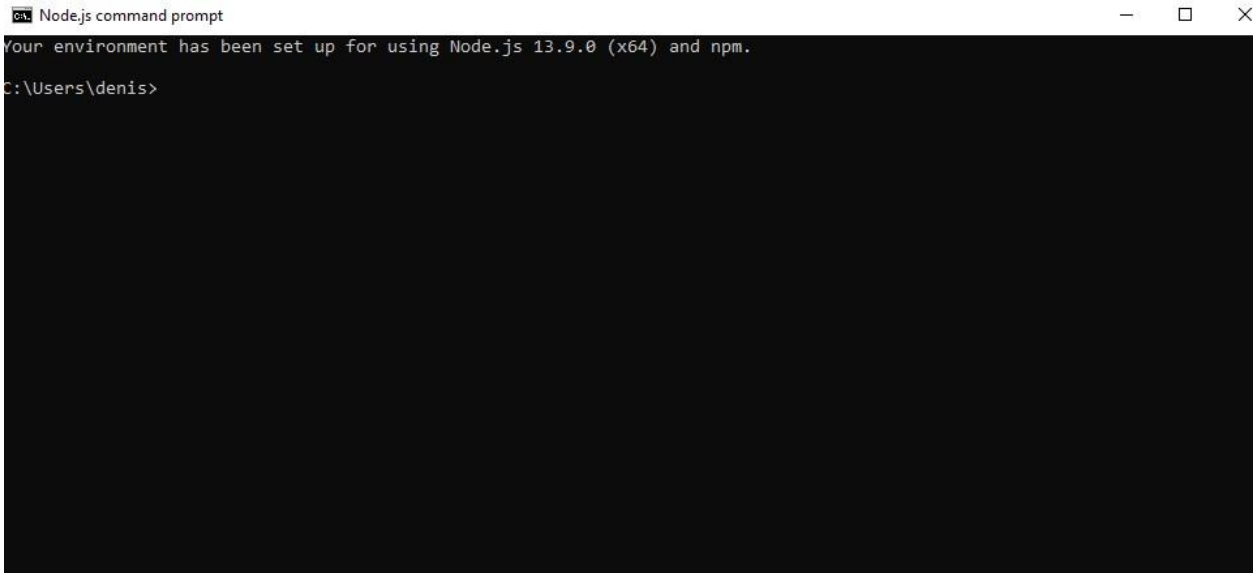

Рис. 3.7 Командний рядок Node.JS.

3.2.3 Налаштування Node.js для роботи з typescript

Так як мова розробки typescript, не є стандартною мовою для платформи node.js, як мова програмування javascript, необхідно додаткове налаштування.

За додаткове налаштування відповідає файл tsconfig.json, в якому розписана конфігурація для роботи

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "sourceMap": true,
    "outDir": "./dist",
    "rootDir": "./src",
    "moduleResolution": "node",
    "allowSyntheticDefaultImports": true
  },
  "exclude": [
    "node_modules"
  ]
}
```

Рис. 3.8 Конфігураційний файл tsconfig.json.

В файлі tsconfig.json знаходиться конфігурація, тобто команди для для нормальної роботи в середовищі node.js.

На даному етапі, в конфігураційному tsconfig.json файлі знаходиться 2 об’єкти, а саме compileOptions, exclude.

CompileOptions – це об’єкт містить властивості для нормально компіляції typescript в javascript код.

- module. Ця команда відповідає за те, як саме будуть імпортуватися файли в проекті;
- target. Ця команда відповідає за те, яка версія javascript буде використовуватись в проєкті;
- sourceMap. Ця команда відповідає за те, чи створювати source файли .map;
- outDir. Ця команда відповідає за те, в яку папку буде вкладено скомпільовані файли. Якщо задана опція "outFile", то опція "outDir" буде проігнорована;
- rootDir. Шлях до папки, з якої треба починати пошук вхідних файлів. Зазвичай коренева директорія визначається за списком вхідних файлів. Ця опція необхідна для перевірки, що всі знайдені файли TypeScript знаходяться всередині кореневої папки.
- moduleResolution. Ця команда відповідає за те, чи визначити спосіб пошуку модулів у папках: як у Node.js або класичний, як у TypeScript 1.5 та нижче.
- allowSyntheticDefaultImports. Ця команда відповідає за те, чи дозволяти імпортувати модулі які не мають "import default";

3.3 Розробка клієнтської частини

Клієнтська частина грає велику роль під час користування продуктом, бо саме від неї залежить простота та приємність від користування. Для розробки клієнтської частини було використано модуль ejs. Проєкт має 3 основні сторінки:

– сторінка реєстрації;
– сторінка аутентифікації;
– сторінка вибору конфігурації;

3.3.1 Розробка клієнтського сервісу

Цей сервіс відповідає за взаємодію клієнта з застосунком. Сервіс повинен взаємодіяти. Роботу сервісу показано на рис. 3.9.

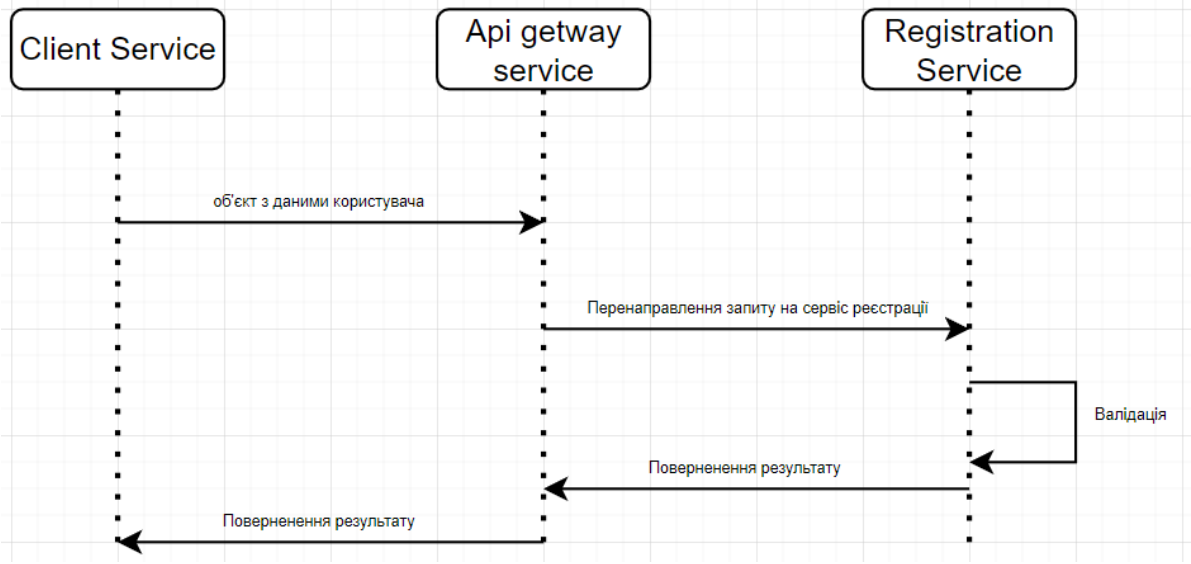

Рис. 3.9 Діаграма зв’язку між сервісом клієнтом та сервісом авторизації.

Для реалізації клієнтської частини було використано node.js модуль – «ejs».

Модуль ejs – це механізм шаблонів, який використовується Node.js. Механізм шаблонів допомагає створити HTML-шаблон із мінімальним кодом. Крім того, він може вводити дані в HTML-шаблон на стороні клієнта та створювати остаточний HTML-код. EJS — це проста мова шаблонів, яка використовується для створення HTML-розмітки за допомогою простого JavaScript. Це також допомагає вбудовувати JavaScript у HTML-сторінки [21].

Перша сторінка – це сторінка аутентифікації, тобто перша сторінка яку бачить клієнт після завантаження застосунку див. рис. 3.10.

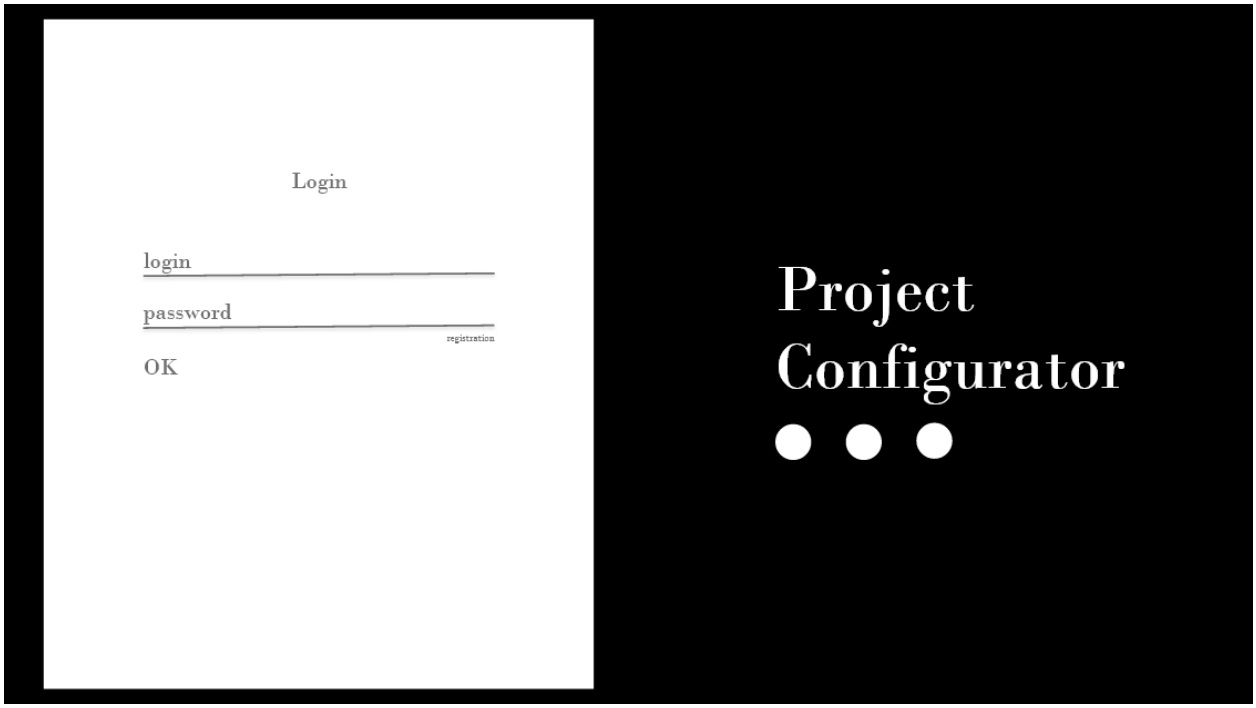

Рис. 3.10 Вигляд сторінки аутентифікації.

На цій сторінці знаходиться форма, яка необхідна для отримання даних від користувача.

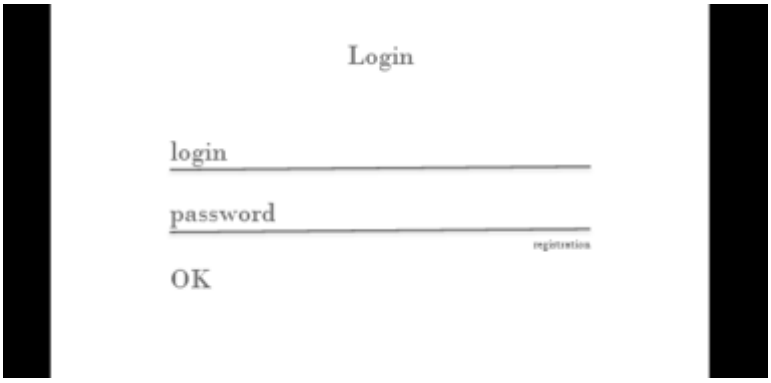

Рис. 3.11 Форма аутентифікації.

При спробі авторизуватися, користувач повинен ввести логін та пароль, які від вказав під час реєстрації, якщо ж користувач є новим та ще не реєструвався раніше, користувач може зареєструватися натиснувши на кнопку реєстрації яка знаходиться під полем для вводу пароля.

Якщо користувач не введе якесь із двох полів обов’язкових полів, то кнопка буде не активною та з’явиться підказка що поле необхідно заповнити див. рис. 3.12.

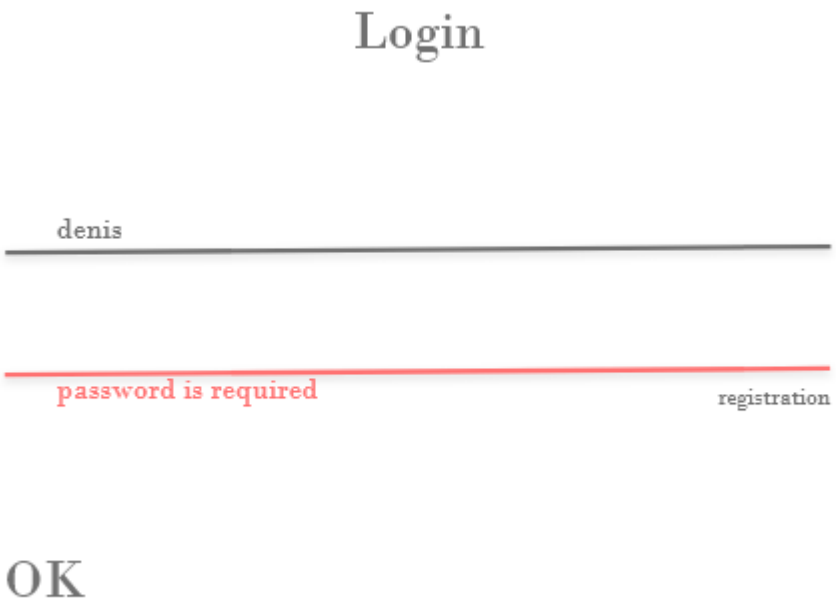

Рис. 3.12 Повідомлення про помилку.

Якщо користувач вперше використовує застосунок, тоді він повинен буде перейти до реєстрації див. рис. 3.13.

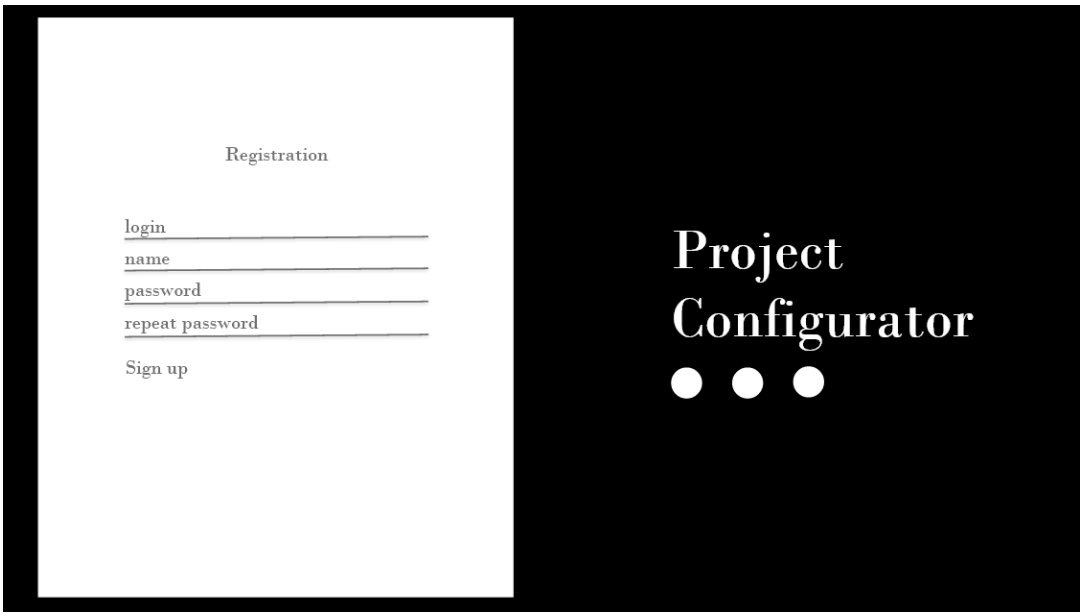

Рис. 3.13 Сторінка реєстрації.

На цій сторінці знаходиться форма реєстрації. Щоб зареєструватися користувач повинен ввести такі дані, як:

– логін;

– ім’я;
– пароль;
– поле для підтвердження пароля;

Якщо, дані введені користувачем, не будуть підходити по заданим критерієм то з’явиться повідомлення про отриману помилку (див. рис. 3.14).

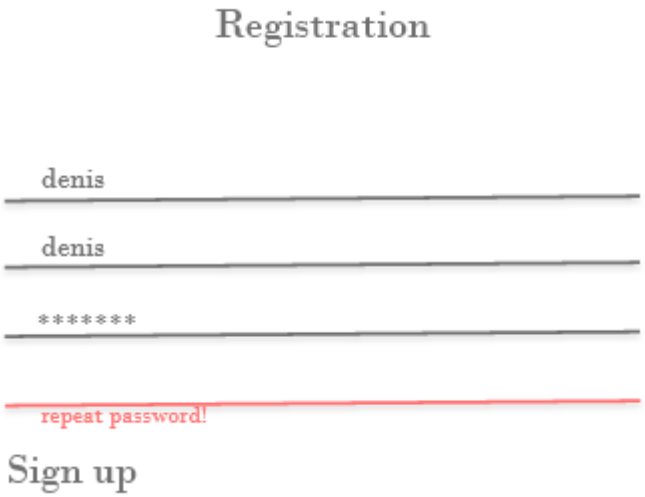

Рис. 3.14 Вивід помилки під час заповнення форми реєстрації.

Після того, як всі дані будуть введені, та валідація даних успішно пройдена, користувач зможе повернутися на сторінку авторизації, та ввести свої дані. Що клієнт вдало створився, дані користувача збереглися в базі даних. То користувач зможе авторизуватися.

Після авторизації, користувач повинен потрапити на сторінки вибору конфігурації проекту див. рис. 3.14.

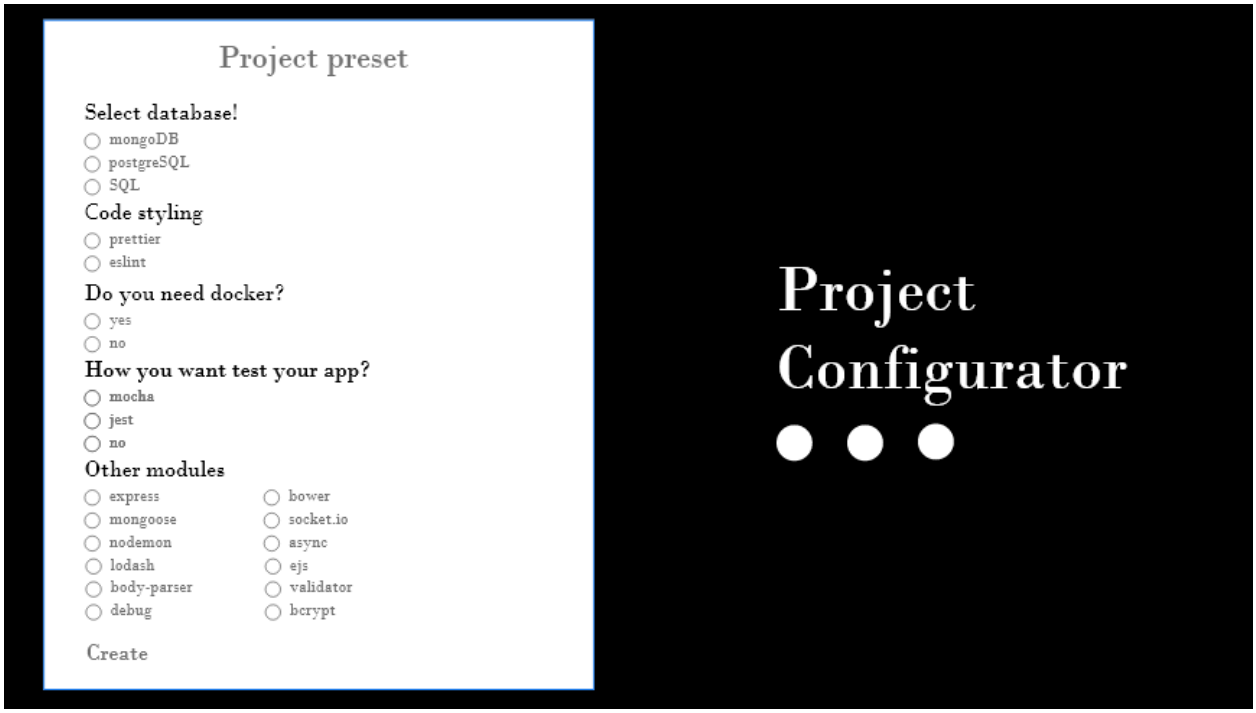

Рис. 3.15 Сторінка вибору конфігурації.

На цій сторінці знаходиться величезна форма, яка містить велику кількість пунктів для створення власної конфігурації проєкту.

Під такі пункти належить:

– вибір бази даних;
– вибір стилізації коду;
– вибір стилю тестування;
– встановлення додаткових модулів;

Рис. 3.16 Вибір критеріїв конфігурації.

В цій формі немає ніяких критеріїв для валідації, окрім того, що користувач в деяких категоріях не зможе обрати декілька критеріїв із одного списку, наприклад декілька баз даних одночасно.

3.4 Розробка серверної частини

Перед початком розробки серверної частини проєкту, тобто розробка самих мікросервісів.

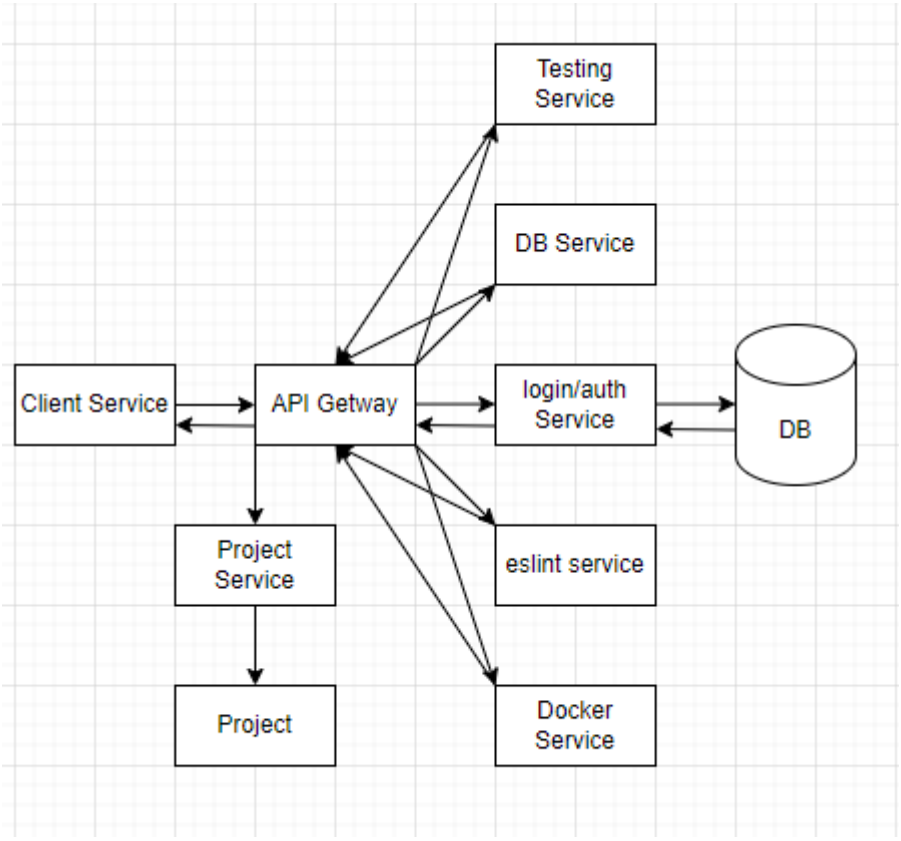

Рис. 3.17 Діаграма взаємодії між мікросервісами.

З цієї діаграми ми можемо бачити повну взаємодію між мікросервісами. Робота починається з клієнтського сервісу, а всі запити проходять через сервіс api Getway, який необхідний для перенаправлення запитів.

3.4.1 Розробка сервісу для аутентифікації

Сервіс для аутентифікації необхідний для того, щоб надавати користувачу право на використовування проєкта.

Як критерії для цього сервісу були обрані такі критерії, як:

– можливість реєстрації нового користувача;
– можливість авторизації;

Повну директорії з сервісом можна подивитися на рис. 3.17. В директорії знаходиться:

– директорія з базою даних;
– директорія з проміжним програмним забезпеченням;
– директорія з бізнес логікою самого сервісу;

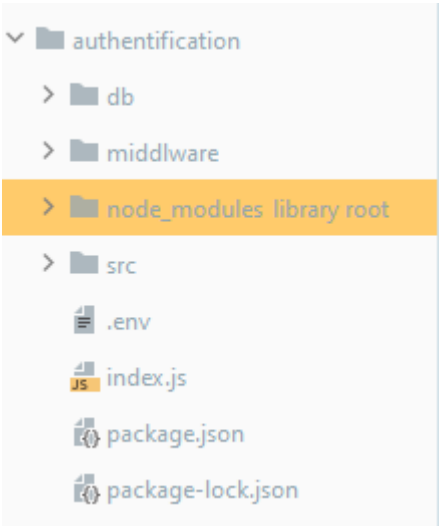

Рис. 3.17 Вигляд директорії сервісу для авторизації/реєстрації.

В директорії для роботи з базою даних знаходиться один файл який відповідає за зв’язок з базою даних, в нашому випадку це postgress.

```
const {Sequelize} = require('sequelize');

module.exports = new Sequelize(
    'pr-architecture',
    'postgres',
    'F62RGT2F',
    {
        dialect: 'postgres',
        host: 'localhost',
        port: 5433,
    }
);
```

Рис. 3.18 Файл для підключення до posgresQL.

Для підключення та роботи з базою даних, було використано бібліотеку sequelize. Для підключення до posgresQL необхідно завантажити саму базу

даних на персональний комп'ютер а також знати такі параметри, як логін та пароль який користувач вводить при завантаженні бази даних.

В директорії мідлваре, знаходиться логіка яка необхідна для виконання по усьому проєкту, тобто вона може бути використання в різних місцях. В нашому випадку в мідлваре зберігається модуль для обробки HTTP помилок див. рис. 3.19.

```
const ApiError = require('../src/errors/ApiError.handler');

module.exports = (err, req, res, next) => {
    if( err instanceof ApiError) {
        return res.status(err.status).json({message: err.message});
    }

    return res.status(500).json({message: 'Unknown Error'})
}
```

Рис. 3.19 Middlware для обробки http помилок.

Так, як говорилось раніше для архітектури самої бізнес логіки проєкту використовувався патерн MVC, тож вся бізнес логіка розділена на контролери, моделі та відображення див. рис. 3.20.

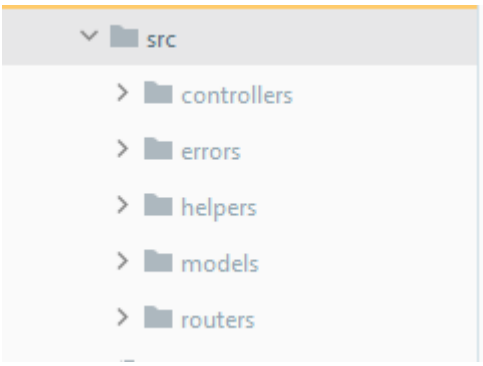

Рис. 3.21 Вміст директорії src.

Так як це сервіс для реєстрації, в директорії контролерів знаходиться лише один контролер, а саме UserController. Цей контролер має два методи, а саме метод для реєстрації та авторизації.

```
async registration (req, res, next) {
    const { name, email, password } = req.body;
    if(!name || !email || !password) {
        return next(ApiError.badRequest('Incorrect user data!'));
    }
    const candidate = await User.findOne ({where: {email}});
    if(candidate) {
        return next(ApiError.badRequest('User with this email  already created!'));
    }
    const user = await User.create( values: {name, email, password});
    if(!user) {
        return next(ApiError.badRequest('User was not created, something went wrong!'));
    }
}
```

Рис. 3.22 Метод для реєстрації користувача.

Ми можемо бачити, що щоб користувача зареєструватися, йому необхідно унікальну пошту, в інакшому випадку спрацює мідлваре в яку ми передаємо статус помилки та повідомлення, що пошта повинна буди унікальною для реєстрації.

```
async login (req, res, next) {
    const { email, password } = req.body;
    const user = await User.findOne({where: {email}});
    if(!user) {
        return next(ApiError.badRequest('User is not founded'));
    }
   if(password !== user.password) {
       return next(ApiError.badRequest('Password is incorrect, try again!'));
   }

   res.json(user);
}
```

Рис. 3.23 Метод для аутентифікації користувача.

Із цього методу ми можемо бачити, що логінація проходить за допомогою пошти та пароля.

Також в директорії знаходиться логіка для моделі, тобто вигляд даних в якому ми будемо працювати з користувачем.

```
const sequelize = require('../../db');
const {DataTypes} = require('sequelize');

const User = sequelize.define( modelName: 'user',  attributes: {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    name: {type: DataTypes.STRING, unique: true},
    email: { type: DataTypes.STRING, unique: true},
    password: { type: DataTypes.STRING }
})
```

Рис. 3.24 Модель користувача.

З цієї моделі ми можемо бачити, що користувач має 4 поля, а саме:

– унікальний id;
– ім’я;
– пошту;
– пароль;

Також дуже важливий компонент кожного застосунку, це роутінг який використовується як скелет застосунку, тобто відповідає за роботу із запитами.

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/UserController');

router.post( path: '/registration', userController.registration);
router.post( path: '/login', userController.login);
```

Рис. 3.25 Директорія з роутами.

3.4.2 Розробка сервісу для налаштування бази даних

Одна із функцій застосунку, це можливість налаштування бази даних. За допомогою цієї функції, можна налаштовувати бази даних, такі як:

– mongoDB;

– sql;
– postgress;

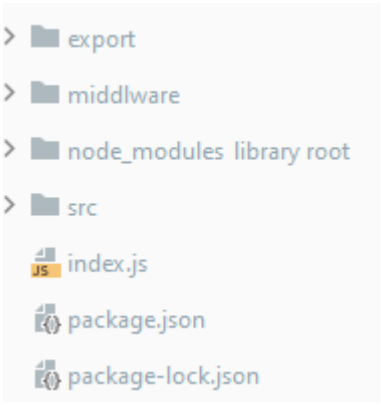

Рис. 3.26 Вміст сервісу з налаштуванням бд.

В директорії export знаходяться файли з налаштуванням бази даних, які вже готові для відправки.

Для налаштування бази даних необхідно два файли, а саме модель з даними та файл з налаштуванням та підключенням до субд.

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

async function dbConnect() {
    await mongoose.connect( uri: 'mongodb://localhost:27017/test');
}
```

Рис. 3.27 Підключення до mongoDB.

```
const {Sequelize} = require('sequelize');

module.exports = new Sequelize(
    'DB_Name',
    'postgres',
    'password',
    {
        dialect: 'postgres',
        host: 'localhost',
        port: 5433,
    }
);
```

Рис. 3.28 Підключення до postgresQL.

Також необхідна обробка помилок, для цього використовується мидлваре з обробкою помилок.

```
const ApiError = require('../src/errors/ApiError.handler');

module.exports = (err, req, res, next) => {
    if( err instanceof ApiError) {
        return res.status(err.status).json({message: err.message});
    }

    return res.status(500).json({message: 'Unknown Error'})
}
```

Рис. 3.29 Обробка помилок для сервісу налаштування бд.

Для бізнес логіки було використано контролер який містить методи для роботи із різними типами субд див. рис. 3.30.

```
async mongoDB(req, res, next) {
    let dbConnectFile;
    let modelFile;
    try {
        dbConnectFile = await fs.readFileSync( path: './export/mongoDB/index.js');
        modelFile = await fs.readFileSync( path: './export/mongoDB/model/model.js');
    } catch (e) {
        return next(ApiError.badRequest('Can not read file!'));
    }

    let dbResponse = {
        title: 'Mongo DB',
        dbFile: dbConnectFile,
        dbModel: modelFile
    };

    res.json(dbResponse);
}
```

Рис. 3.30 Контролер для роботи з mongoDB.

В цьому контролері проводиться зчитування файлів які вже були підготовлені та створення об’єктів для відповідей. Об’єкт респонсу містить в

собі 3 поля, а саме, назву бд, буфер файлу для підключення до бази даних та файл з моделі.

```
async postgresQL(req, res, next) {
    let dbConnectFile;
    let modelFile;
    try {
        dbConnectFile = await fs.readFileSync( path: './export/postgres/index.js');
        modelFile = await fs.readFileSync( path: './export/postgres/model/model.js');
    } catch (e) {
        return next(ApiError.badRequest('Can not read file!'));
    }

    let dbResponse = {
        title: 'PostgresQL',
        dbFile: dbConnectFile,
        dbModel: modelFile
    };

    res.json(dbResponse);
}
```

Рис. 3.31 Контролер для роботи з postgresQL.

```
const express = require('express');
const router = express.Router();
const dbController = require('../controllers/DBController');
const fs = require("fs");

router.get( path: '/mongo', dbController.mongoDB);
router.get( path: '/postgres', dbController.postgresQL);

module.exports = router;
```

Рис. 3.32 Модуль для роутів.

	id [PK] integer	name character varying (255)	email character varying (255)	password character varying (255)	createdAt timestamp with time zone	updatedAt timestamp with time zone
1	2	John Collins	jColl@gmail.com	$2a$04$lafrA1AqUhb/SIljgREUO.0EETOxAO7/59hHv7gTI7JruqK9TDSRa	2022-02-17 00:29:34.264+03	2022-02-17 00:29:34.264+03
2	1	Denis Saponko	dSaponko@gmail.com	$2a$04$o0Gsx3qbPTO0QjYWoZOoqeFtRySR7e3050A.rqA/a1NNAo5JbtHcm	2022-02-17 00:27:34.844+03	2022-02-17 00:27:34.844+03

Рис. 3.33 Вигляд таблиці users.

3.4.3 Розробка сервісу для налаштування стилізації коду

Дуже велику роль грає якість коду та простота написання, допомагає слідкувати за цим – eslint.

ESLint — це утиліта JavaScript з відкритим вихідним кодом, спочатку створена Ніколасом у червні 2013 року. Лінтинг коду — це тип статичного аналізу, який часто використовується для пошуку проблемних шаблонів або коду, який не відповідає певним рекомендаціям щодо стилю. Існують лінтери коду для більшості мов програмування, і компілятори іноді включають лінтинг у процес компіляції.

JavaScript, будучи динамічною мовою з нечіткими типами, особливо схильний до помилок розробника. Без переваги процесу компіляції код JavaScript зазвичай виконується, щоб знайти синтаксичні чи інші помилки. Інструменти Linting, такі як ESLint, дозволяють розробникам виявляти проблеми з їх кодом JavaScript, не виконуючи його.

Основна причина створення ESLint полягала в тому, щоб дозволити розробникам створювати власні правила лінтингу. ESLint розроблено так, щоб усі правила повністю підключалися. Правила за замовчуванням написані так само, як і будь-які правила плагінів. Усі вони можуть діяти за одним шаблоном, як для самих правил, так і для тестів. Хоча ESLint буде поставлятися з деякими вбудованими правилами, щоб зробити його корисним із самого початку, ви зможете динамічно завантажувати правила в будь-який момент часу [24].

Цей сервіс необхідний для того, що генерувати eslint файл, а потому повертати для зберігання в конфігурації користувача.

```
module.exports = {
  env: {
    browser: true,
    commonjs: true,
    es2021: true
  },
  extends: [
    'standard'
  ],
  parser: '@typescript-eslint/parser',
  parserOptions: {
    ecmaVersion: 'latest'
  },
  plugins: [
    '@typescript-eslint'
  ],
  rules: {
  }
}
```

Рис. 3.34 Вміст файлу .eslintrc.

Цей файл містить налаштування, за допомогою яких ядро буде дивитися за стилем та шаблоном написання коду, і якщо з’явиться якась проблема чи не точність, а бо щось таке, що не відповідає налаштуванням, то компілятор інтерпритатор не правильний код.

3.4.4 Розробка сервісу для налаштування docker

Даний мікросервіс, мабуть є однім із найпростіших тому, що має лише настройку одного файлу, докер файлу.

Docker — це відкрита платформа для розробки, доставки та запуску програм. Docker дає змогу відокремити ваші програми від інфраструктури, щоб ви могли швидко постачати програмне забезпечення. За допомогою Docker ви можете керувати своєю інфраструктурою так само, як і своїми

програмами. Використовуючи переваги методології Docker для швидкої доставки, тестування та розгортання коду, ви можете значно скоротити затримку між написанням коду та його запуском у виробництві [23].

Якщо користувач вибере пункт для завантаження докера в свою архітектуру проєкту, то на цей сервіс прийде запит для генерування докер файлу.

```
FROM node:9-slim
WORKDIR /app
COPY package.json ./app
RUN npm nstall
COPY . ./app
CMD ["npm", "start"]
```

Рис. 3.34 Згенерований докер файл.

```
async getDockerFile(req, res, next) {
    let dockerFile;
    try {
        dockerFile = await fs.readFileSync( path: './Dockerfile');
    } catch (e) {
     return next(ApiError.badRequest('Can not read file!'));
    }

    res.json({
        title: 'docker',
        data: {
            dockerFile
        }
    });
}
```

Рис. 3.35 Функція для отримання докер файлу.

```
{
  title: 'docker',
  data: {
    dockerFile: <Buffer 46 52 4f 4d 20 6e 6f 64 65 3a 39 2d 73 6c 69 6d 0d 0a 57 4f 52 4b 44 49 52 20 2f 61 70 70 0d 0a 43 4f 50 59 20 70 61 63 6b 61 67 65 2e 6a 73 6f 6e 20 ... 63 more bytes>
  }
}
```

Рис. 3.36 Об’єкт для відповіді.

3.4.5 Розробка сервісу для побудови архітектури

Цей сервіс безпосередньо відповідає за побудову самої архітектури на базі буферів згенерованих файлів.

```
const express = require('express')
const cors = require('cors')
const path = require('path')
const router = require('./src/router')
const app = express()

app.use(cors())
app.use(express.json());
app.use('/api', router);

app.use(express.static(path.join(__dirname, 'public')))

app.listen( port: 3006,  hostname: () => {
    console.log('Server running on port 3004')
})
```

Рис. 3.37 Головний файл сервісу.

Після того як виконаються всі запити, і будуть отримані всі файли, на базі отриманих даних на робочому столі буде створено директорії «project».

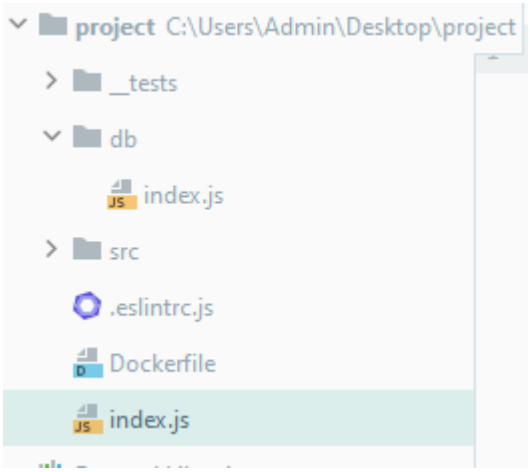

Рис. 3.38 Згенерована архітектура проєкту.

В даному прикладі було створено архітектуру проєкту, яка мімтить налаштований докер файл, налаштований еслінт, базу даних постгрес, та тести.

Висновки до розділу 3

В даному розділі МКР було розроблено npm модуль з використанням мікросервісної архітектури. За допомогою модуля можна генерувати архітектуру застосунку. Було проаналізовано методи розробки npm модулів. Також було створено макет клієнтської частини, детально описано функціонал кожної із сторінок. Було проаналізовано структуру веб застосунків та методи її створення. Було описано роботу з базою даних PostgresQL.

ВИСНОВОК

У магістерській кваліфікаційній роботі було розроблено застосунок для генерації архітектури проектів з використанням мікросервісної архітектури.

Мета МКР – зменшення часу та полегшення розробки програмного забезпечення за допомогою розробки npm модуля для генерації структури проєкту з використання мікросервісної архітектури досягнута..

Були виконані такі завдання як, виявлено особливості проектування та створення веб серверів; проаналізовано та виділено особливості написання та опублікування npm модулів; розроблено додаток для створення та побудови архітектури проекту; проведено тестування додатку.

Для цього проекту було окреслені чіткі задачі, які було виконано. Було показано крок за кроком етапи розробки та складності. Було створено та підключено базу даних postgresQL. За підключення та роботу з базою відповідає бібліотека sequelize. Для розробки клієнтської частини використовувалась бібліотека ejs та мова javascript. Для розробки бекенда використовувались сучасні технології, nodejs, Express, postgresQl.

У процесі роботи над проектом були отримані практичні навички в дослідженній предметній області, в описі проектного рішення, побудові моделей. У роботі була використана методологія об'єктно-орієнтованого проектування. В результаті, розроблений додаток відповідає всім вимогам, які були поставлені перед початком розробки проекту і має практичну значимість.

Спеціальний розділ МКР містить певні розрахунки, рекомендації та положення щодо використання інформаційних пристроїв, які є актуальними і для розробників програмного забезпечення, і для користувачів. Ці рекомендації дозволяють уникати небезпек для здоров’я та життя, що є найважливішим аспектом життєдіяльності та благополуччя громадян.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Моделі життєвого циклу програмного забезпечення: вебсайт URL: https://evergreens.com.ua/ru/articles/software-development metodologies.html (дата звернення: 16.11.2021).

2. С чого складається програмне забезпечення: вебсайт URL: https://kenzie.snhu.edu/blog/front-end-vs-back-end-whats-the-difference/ (дата звернення: 17.11.2021).

3. С чого складається програмне забезпечення: вебсайт URL: https://www.geeksforgeeks.org/frontend-vs-backend/ (дата звернення: 17.11.2021).

4. Backend сторона : вебсайт URL: https://otus.ru/nest/post/943/ (дата звернення: 18.11.2021).

5. Backend сторона: вебсайт URL: http://www.geog.leeds.ac.uk/courses/postgrad/web/lectures/server-side-technologies/1_introduction/index.html (дата звернення: 18.11.2021).

6. MVC: вебсайт : вебсайт URL: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem- prostymi-slovami (дата звернення: 06.06. 2020).

7. Пакетні менеджери : вебсайт URL: https://blog.vistro.ru/dependency-management/ (дата звернення: 19.11.2021).

8. NPM : вебсайт URL: https://docs.npmjs.com/about-npm (дата звернення: 19.11.2021).

9. Yarn : вебсайт URL: https://yarnpkg.com/ (дата звернення: 27.11.2021).

10. Bower : вебсайт URL: https://snyk.io/blog/bower-is-dead/ (дата звернення: 27.11.2021).

11. Bower : вебсайт URL: https://snyk.io/blog/bower-is-dead/ (дата звернення: 28.11.2021).

12. Npm модуль Socket.io : вебсайт URL: https://www.npmjs.com/package/socket.io (дата звернення 01.12.2021)

13.Архітектура програмного забезпечення : вебсайт URL: https://searchapparchitecture.techtarget.com/definition/software (дата звернення: 02.12.2021).

14. База даних PostgresQL : вебсайт URL: https://www.postgresql.org/ (дата звернення: 15.12.2021).

15. Платформа Node.js : вебсайт URL: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js (дата звернення: 19.12.2021).

16. СУБД : вебсайт URL:https://searchdatamanagement.techtarget.com/definition/database (дата звернення: 24.12.2021).

17. Adobe XD : вебсайт URL: https://helpx.adobe.com/ru/xd/ (дата звернення: 02.01.2022).

18. Rainbow plagin : вебсайт URL: https://www.kobzarev.com/soft/rainbow-brackets/ (дата звернення: 03.01.2022).

19. Key Promoter X : вебсайт URL: geeksforgeeks.org/integrating-key-promoter-x-plugin-with-android-studio/ (дата звернення: 05.01.2022).

20. Material Icon Theme : вебсайт URL:https://marketplace.visualstudio.com/items?itemName=PKief.material-icon-theme (дата звернення: 07.01.2022).

21. Node.js : вебсайт URL: https://nodejs.org/uk/ (дата звернення: 09.01.2022).

22. Node.js модуль – ejs : вебсайт URL: https://www.geeksforgeeks.org/use-ejs-as-template-engine-in-node-js/ (дата звернення: 20.01.2022).

23. Docker : вебсайт URL: https://docs.docker.com/get-started/overview/ (дата звернення: 20.01.2022).

24. Eslint : вебсайт URL: https://eslint.org/docs/about/ (дата звернення: 20.01.2022).

25. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: Підручник. 5-е вид. / За ред. М.П. Гандзюка. - К.: Каравела, 2011. - 384 с.

26. Гігієнічна класифікація умов праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу. – К.: МОЗ України, 1998. – 34 с.