МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Органайзер на основі веб системи

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-41

_____Хилько Д.М._____
(прізвище та ініціали)

Керівник _____Сайко В.Г._____
(прізвище та ініціали)
_____д.т.н., проф._____
(науковий ступінь, звання)

Унікальність тексту 92%

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *прикладних інформаційних систем*
Протокол № від р.

зав. кафедри_  Плескач В.Л.

Київ – 2023

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	22.05.2023	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	Виконано

14.	Захист кваліфікаційної роботи бакалавра	27.06.2023	

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Складові частини кваліфікаційної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план кваліфікаційної роботи	1
Відомість кваліфікаційної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	14
2	28
3	13
Висновки	1
Перелік використаних джерел	1

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Хилько Д.М.		26.05.2023	Відомість кваліфікаційної роботи	Лист	Листів
Керівн.	Сайко В.Г.					
Н/контр.	Макаренко С.А.		26.05.2023			
Зав.каф.	Плескач В.Л.					

АНОТАЦІЯ

Кваліфікаційна робота: 67 с., 10 рис., 14 джерел.

Кваліфікаційна робота присвячена розробці органайзера на основі веб системи.

Метою кваліфікаційної роботи бакалавра є результативне та оперативне управління нотатками на основі створеного органайзера.

Для досягнення поставленої мети потрібно вирішити такі **завдання**:

- дослідити підходи до розроблення органайзерів на основі веб системи;
- здійснити аналіз архітектурних рішень і вибрати програмні засоби для реалізації органайзера на основі веб системи;
- описати реалізацію на впровадження органайзера на основі веб системи.

Об’єктом дослідження є процеси організації нотаток.

Предметом дослідження кваліфікаційної роботи бакалавра є програмні, технічні, організаційні засади, принципи, концептуальні підходи до побудови органайзера.

У роботі використано такі **методи дослідження:** системний підхід і системний аналіз, проектний підхід — SCRUM методологію управління проектами.

Ключові слова: веб система, органайзер, нотатки, FastAPI, Angular.

ABSTRACT

Thesis: 67 pages, 10 figures, 14 sources.

The thesis is devoted to the development of an organizer based on a web system.

The purpose of the bachelor's qualification work is effective and efficient management of notes based on the created organizer.

To achieve the set goal, the following **tasks** must be solved:

- to investigate approaches to the development of organizers based on the web system;
- carry out an analysis of architectural solutions and choose software tools for implementing an organizer based on a web system;
- describe the implementation of the organizer based on the web system.

The object of study is the process of organizing notes.

The subject of the thesis study is programmatic, technical, organizational principles, principles, and conceptual approaches to building an organizer.

The following **research methods** were used in the work: system approach and system analysis, project approach — SCRUM project management methodology.

Keywords: web system, organizer, notes, FastAPI, Angular.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API - прикладний програмний інтерфейс
CMS - система управління контентом
LCDP - платформи розробки з низьким кодом
PWA - прогресивна веб-програма
ORM - object relational mapping

ВСТУП

Актуальність теми органайзера на основі веб системи є важливим елементом менеджменту. У сучасному стрімкому та інформаційно керованому світі люди та організації стикаються зі зростаючою потребою в ефективних інструментах організації та продуктивності. Веб-система органайзера надає централізовану платформу для керування нотатками, завданнями, розкладами, документами та співпрацею, що дозволяє користувачам залишатися організованими та підвищувати продуктивність.

Із зростанням віддаленої роботи та розподілених команд попит на онлайн-інструменти для співпраці та координації значно зріс. Веб-система органайзера полегшує віддалену співпрацю, забезпечує безперебійне спілкування та допомагає членам команди залишатися узгодженими та організованими незалежно від їх фізичного розташування.

Метою кваліфікаційної роботи бакалавра є результативне та оперативне управління нотатками на основі створеного органайзера, для цього необхідно вирішити такі завдання:

- дослідити підходи до розроблення органайзерів на основі веб системи;
- здійснити аналіз архітектурних рішень і вибрати програмні засоби для реалізації органайзера на основі веб системи;
- описати реалізацію на впровадження органайзера на основі веб системи.

Об’єктом дослідження є процеси організації нотаток.

Предметом дослідження кваліфікаційної роботи бакалавра є програмні, технічні, організаційні засади, принципи, концептуальні підходи до побудови органайзера.

У роботі використано такі **методи дослідження:** системний підхід і системний аналіз, проектний підхід — SCRUM методологію управління проектами.

У процесі виконання кваліфікаційної роботи бакалавра було використано:

для бекенду:

- мова програмування Python
- фреймворк FastAPI
- об’єктно-реляційну проекцію SQLAlchemy
- міграцію бази даних Alembic

для фронтенду:

- мова програмування TypeScript
- фреймворк Angular

Дипломна робота складається зі вступу, трьох розділів, висновку і списку використаних джерел.

РОЗДІЛ 1 ПІДХОДИ ДО РОЗРОБЛЕННЯ ОРГАНАЙЗЕРІВ НА ОСНОВІ ВЕБ СИСТЕМИ

1.1 Дослідження підходів

Для побудови веб системи з розвитком технологій з’явилося багато різних можливостей і способів, які можуть відповідно до свого рівня справитись з поставленою задачею. Вибір може бути як платформи, які пропонують розробку з мінімальним написанням коду або взагалі без написання коду, а з допомогою візуальної взаємодії - це так звані low-code або no-code платформи.

Існують системи управління контентом (CMS) які дозволяють користувачам створювати, керувати та публікувати цифровий вміст у мережі. Вони забезпечують зручний інтерфейс і набір інструментів, які дозволяють користувачам без технічних знань легко створювати та оновлювати вміст веб-сайту.

Серед варіантів розробки є поняття як індивідуальна розробка з використанням чистого HTML, CSS і JavaScript. Цей підхід забезпечує максимальну гнучкість і налаштування, але вимагає значних зусиль і досвіду розробки.

API-орієнтована розробка фокусується на створенні надійного серверного API, який надає функціональні можливості різним клієнтам, таким як веб-переглядачі, мобільні програми чи інші системи. Такий підхід забезпечує гнучкість інтерфейсних технологій і забезпечує легку інтеграцію зі сторонніми службами.

Популярним є використання розробниками фреймворків і бібліотек, що значно підвищує ефективність і покращує загальну якість системи.

1.2 Платформи розробки з низьким кодом (LCDP)

Платформи розробки з низьким рівнем коду (LCDP) — це платформи розробки програмного забезпечення, які дозволяють створювати програми з мінімальним кодуванням вручну. Вони забезпечують візуальні інтерфейси, інструменти перетягування та готові компоненти, які дозволяють

розробникам, навіть тим, хто має обмежений досвід програмування, створювати програми швидко та ефективно.

LCDP пропонують середовище візуальної розробки, де розробники можуть проектувати інтерфейси програм, робочі процеси та логіку за допомогою графічних інтерфейсів і візуальних інструментів. Це зменшує потребу в написанні великого коду та спрощує процес розробки програми.

Вони надають бібліотеку попередньо створених компонентів і шаблонів, які можна легко перетягувати в інтерфейс програми. Це дозволяє розробникам швидко зібрати користувальницький інтерфейс програми та функціональність без написання коду з нуля. Вони можуть створювати застосунки набагато швидше порівняно з традиційними підходами кодування. Візуальна розробка та готові до використання компоненти спрощують процес розробки та скорочують час, необхідний для впровадження.

Платформи з низьким кодом розроблені таким чином, щоб бути зручними та доступними для розробників різного рівня кваліфікації. Вони забезпечують інтуїтивно зрозумілі інтерфейси, візуальні редактори та документацію, щоб спростити процес розробки програми та скоротити час навчання. Хоча дані платформи пропонують готові компоненти, вони також забезпечують гнучкість для налаштування, що дозволяє змінювати зовнішній вигляд, поведінку та функціональність компонентів відповідно до вимог конкретної програми.

Платформи з низьким кодом часто включають функції співпраці, які дозволяють кільком розробникам працювати над одним проектом одночасно. Це сприяє командній роботі, забезпечує швидші ітерації та сприяє зворотному зв’язку та спілкуванню між членами команди.

Дані платформи пропонують можливості інтеграції для підключення до зовнішніх систем, API та баз даних. Це дозволяє розробникам використовувати існуючі служби та джерела даних, підвищуючи функціональність і універсальність застосунків.

Low-code платформи надають інструменти та фреймворки для створення мобільних застосунків і підтримки кількох каналів, таких як веб, мобільний і робочий стіл. Це дозволяє розробляти кросплатформні програми, які можуть працювати на різних пристроях і платформах.

Платформи з низьким кодом часто забезпечують функції масштабованості, такі як автоматичне масштабування та балансування навантаження, щоб відповідати зростаючим вимогам користувачів. Крім того, вони зазвичай пропонують можливості обслуговування та контролю версій, що полегшує керування оновленнями та виправлення помилок.

Однією з найбільших переваг даних платформ є те, що вони дають змогу людям, які не мають досвіду розробки або відповідних знань, створювати програми. Це розширює коло тих, хто може внести свій внесок у розробку програм, і сприяє інноваціям в організаціях.

Платформи розробки з низьким кодом пропонують баланс між швидкістю та гнучкістю, дозволяючи організаціям швидко створювати програми, дотримуючись індивідуальних вимог. Вони особливо корисні для проектів з часовими обмеженнями, обмеженими ресурсами розробки або потребою в частих ітераціях і оновленнях. Однак при виборі платформи з низьким кодом важливо оцінити масштабованість, розширюваність і аспекти довгострокового обслуговування, щоб переконатися, що вона відповідає конкретним цілям і вимогам вашого проекту.

1.3 Системи управління контентом (CMS)

Системи керування вмістом (CMS) — це програмні платформи, які дозволяють користувачам створювати, керувати та публікувати цифровий вміст у мережі. Вони забезпечують зручний інтерфейс і набір інструментів, які дозволяють користувачам без технічних знань легко створювати та оновлювати вміст веб-сайту.

Платформи CMS пропонують інтуїтивно зрозумілі можливості створення та редагування вмісту. Користувачі можуть створювати та впорядковувати такий вміст, як статті, публікації в блогах, сторінки,

зображення, відео та інші засоби масової інформації за допомогою зручного редактора. CMS зазвичай надають такі функції, як редактори WYSIWYG (що бачиш, те й отримаєш), медіа-бібліотеки та версії вмісту для ефективного керування вмістом.

Вони відокремлюють вміст від презентації за допомогою шаблонів і тем. Користувачі можуть вибирати з ряду попередньо розроблених шаблонів або створювати власні дизайни за допомогою вбудованих інструментів або інтеграції з програмним забезпеченням для веб-дизайну. Шаблони забезпечують узгодженість веб-сайту, забезпечуючи гнучкість для налаштування.

Платформи CMS надають функції керування користувачами, що дозволяє адміністраторам створювати облікові записи користувачів і керувати ними. Можна визначити ролі та дозволи користувачів, щоб контролювати рівні доступу та обмежувати певні дії на основі ролей користувачів. Це дозволяє співпрацювати та делегувати завдання керування вмістом між кількома користувачами.

Системи управління контентом часто містять вбудовані функції пошукової оптимізації (SEO) для покращення видимості веб-сайту в рейтингу пошукових систем. Вони надають інструменти для додавання метатегів, оптимізації URL-адрес, керування перенаправленнями та створення карт сайту, які допомагають пошуковим системам розуміти та індексувати вміст веб-сайту.

Платформи CMS пропонують розширюваність за допомогою плагінів, модулів або розширень, які розширюють основну функціональність системи. Вони дозволяють інтегруватись зі сторонніми службами, функціями електронної комерції, інтеграцією соціальних мереж, аналітикою та іншими функціями на основі конкретних потреб.

Дані платформи надають можливості адаптивного дизайну, гарантуючи оптимізацію веб-сайтів для різних пристроїв і розмірів екрана. Крім того,

CMS часто підтримують багатомовне керування вмістом, що дозволяє створювати та керувати вмістом кількома мовами.

CMS пропонують робочий процес і функції співпраці, які полегшують процеси затвердження вмісту, керування версіями та робочі процеси публікації вмісту. Ці функції спрощують процес створення та перегляду вмісту, забезпечуючи точність і послідовність вмісту.

Системи управління контентом розроблені для роботи з веб-сайтами різного розміру та обсягів трафіку. Вони забезпечують такі функції масштабованості, як кешування, мережі доставки контенту (CDN) і оптимізацію бази даних для підвищення продуктивності та задоволення зростаючих вимог до веб-сайтів.

Вони надають пріоритет безпеці та пропонують такі функції, як автентифікація користувачів, контроль доступу на основі ролей, шифрування даних і захист від поширених вразливостей. Зазвичай надаються регулярні оновлення системи безпеки та виправлення для усунення нових загроз і вразливостей.

CMS часто мають активні спільноти, форуми та канали підтримки, де користувачі можуть звернутися за допомогою, поділитися знаннями та знайти ресурси. Ця підтримка, керована спільнотою, надає доступ до документації, навчальних посібників, плагінів і тем, допомагаючи користувачам використовувати весь потенціал CMS.

Серед популярних платформ CMS – WordPress, Drupal, Joomla та TYPO3. Кожна CMS має свої сильні сторони та особливості, тому вибір правильної залежить від таких факторів, як вимоги до проекту, потреби в масштабованості, бюджет, простота використання та наявний досвід.

1.4 Розробка на основі інтерфейсів прикладного програмування (API)

Розробка на основі API або розробка, орієнтована на API, — це підхід до побудови веб-систем, де процес розробки обертається навколо проектування та впровадження API (інтерфейсів прикладного програмування) як основного фокусу. Веб-система побудована як клієнт-

серверна архітектура, де сервер надає API для надання даних і функцій, а клієнт використовує ці API для взаємодії з сервером і представлення інформації користувачам.

У розробці дизайн API є критичним кроком. Це передбачає визначення кінцевих точок, структур даних, механізмів автентифікації та форматів запитів/відповідей. Добре розроблені API є інтуїтивно зрозумілими, узгодженими та відповідають таким стандартам, як REST (Representational State Transfer) або GraphQL, що забезпечує взаємодію та легкість інтеграції.

Даний вид розробки сприяє розділенню завдань між серверними та зовнішніми компонентами веб-системи. Серверна частина зосереджена на забезпеченні надійних і масштабованих API, які обробляють дані, бізнес-логіку та зберігання даних. Інтерфейс, який часто створено за допомогою таких технологій, як HTML, CSS і JavaScript, використовує ці API для отримання даних і представлення їх користувачам.

Розробка орієнтована на API пропонує гнучкість у виборі технологій для інтерфейсу та серверної частини. Це дозволяє розробникам використовувати різні фреймворки, мови або платформи, які найкраще підходять для кожного компонента. Це також дає змогу масштабувати систему шляхом впровадження додаткових інтерфейсних клієнтів або підтримки кількох платформ, таких як веб-інтеграція, мобільна або стороння інтеграція, усі з використанням тих самих внутрішніх API.

Модульність і можливість багаторазового використання. Проектуючи API як компоненти для багаторазового використання, розробка на основі API сприяє модульності та повторному використанню коду. API інкапсулюють певні функції та дані, що полегшує створення нових функцій або розширення наявних. Цей модульний підхід також забезпечує командну співпрацю, коли розробники серверної частини зосереджуються на створенні API незалежно, а розробники зовнішньої частини можуть одночасно працювати над використанням цих API.

API забезпечують чіткий договір між інтерфейсом і сервером, що дозволяє командам працювати незалежно. Розробники бекенда можуть створювати та тестувати API за допомогою таких інструментів, як Postman або фреймворків автоматизованого тестування, до початку розробки інтерфейсу. Подібним чином розробники інтерфейсу можуть імітувати або імітувати відповіді API на етапі розробки, забезпечуючи паралельну розробку та ефективне тестування.

Розробка, керована API, забезпечує бездоганну інтеграцію зі сторонніми службами та системами. Завдяки добре розробленим API веб-система може легко взаємодіяти із зовнішніми службами, такими як платіжні шлюзи, платформи соціальних мереж або постачальники даних. Це дозволяє веб-системі використовувати існуючі функції та джерела даних, розширюючи свої можливості та надаючи більше цінності користувачам.

API потребують вичерпної документації, яка допоможе розробникам ефективно їх використовувати. Розробка, керована API, наголошує на чіткій та оновленій документації, що дозволяє розробникам зрозуміти доступні кінцеві точки, формати запитів/відповідей, вимоги до аутентифікації та обробки помилок. Хороший досвід розробника сприяє співпраці, прискорює розробку та заохочує до прийняття сторонніми розробниками.

Розробка на основі API набула популярності завдяки своїй гнучкості, модульності та здатності полегшувати інтеграцію з різними технологіями та службами. Це забезпечує відокремлену архітектуру, де інтерфейсні та бекенд-компоненти можуть розвиватися незалежно, покращуючи масштабованість, зручність обслуговування та розширюваність веб-системи в майбутньому. Однак це вимагає ретельного проектування API, стратегій управління версіями, міркувань безпеки та ретельного тестування для забезпечення надійності та сумісності системи.

1.5 Прогресивні веб-програми (PWA)

Прогресивні веб-програми (PWA) — це веб-програми, які забезпечують покращену взаємодію з користувачем, подібну до рідних

мобільних програм. Вони використовують сучасні веб-технології для надання функцій, схожих на застосунки, швидкості реагування та можливостей офлайн.

Взаємодія, схожа на застосунок: PWA пропонують користувацький досвід, схожий на нативний, із такими функціями, як плавна анімація, push-сповіщення та повноекранний режим. Вони забезпечують послідовний і привабливий інтерфейс, який дуже нагадує рідні мобільні програми.

Прогресивні веб-програми створено з використанням таких веб-технологій, як HTML, CSS і JavaScript, що робить їх сумісними з кількома платформами та пристроями. Вони можуть працювати в будь-якому сучасному веб-браузері, включаючи комп’ютери, смартфони, планшети та навіть смарт-телевізори. Їх розроблено для швидкого реагування та адаптації до різних розмірів і орієнтацій екрана. Вони забезпечують безперебійну роботу на різних пристроях, забезпечуючи гарний вигляд і роботу програми незалежно від пристрою користувача.

Однією з значних переваг PWA є їх здатність працювати в автономному режимі або за поганого підключення до Інтернету. Вони можуть кешувати дані та ресурси, дозволяючи користувачам отримувати доступ до програми та виконувати певні завдання, навіть якщо вони не підключені до Інтернету.

PWA використовують такі методи, як сервіс-воркери та кешування для підвищення продуктивності. Сервісні працівники діють як проміжний рівень між застосунком і мережею, забезпечуючи роботу в автономному режимі та кешування ресурсів для швидшого завантаження. Серед переваг PWA можна зазначити можливість виявлення та спільного використання. Їх можна легко виявити за допомогою пошукових систем, оскільки вони є мережевими. Ними також можна ділитися через URL-адреси, що дозволяє користувачам отримувати прямий доступ до програми без необхідності завантажувати її з магазину програм.

Розробка PWA може бути рентабельною порівняно зі створенням окремих нативних програм для різних платформ. Завдяки єдиній кодовій базі можна охопити ширшу аудиторію на кількох пристроях, скоротивши зусилля з розробки та обслуговування. PWA можна оновлювати автоматично, не вимагаючи від користувачів вручну завантажувати та встановлювати оновлення. Це гарантує, що користувачі завжди матимуть доступ до останньої версії програми та будь-яких виправлень помилок або вдосконалених функцій.

Прогресивні веб застосунки користуються перевагами функцій безпеки, які надають веб-браузери, наприклад шифрування HTTPS. Це допомагає захистити дані користувача та гарантує безпечний досвід перегляду.

Розповсюдження в магазині застосунків (необов'язково): хоча PWA є мережевими, їх також можна поширювати через магазини застосунків, такі як Google Play Store або Apple Store. Це дозволяє охопити користувачів, які віддають перевагу установці програм із магазинів застосунків, водночас користуючись перевагами PWA.

Загалом, PWA надають привабливе рішення для надання багатих і привабливих користувальницьких можливостей в Інтернеті, поєднуючи в собі найкраще з Інтернету та рідних програм. Вони пропонують розширене охоплення, покращену продуктивність і посилене залучення користувачів, що робить їх популярним вибором для багатьох веб-застосунків.

1.6 Serverless (безсерверний) підхід

Безсерверний підхід до розробки органайзера на основі веб системи передбачає використання безсерверних обчислювальних платформ і служб для створення та розгортання веб-застосунків без керування основною інфраструктурою. Замість того, щоб налаштовувати та підтримувати сервери, розробники можуть зосередитися виключно на написанні коду для логіки та функціональності програми.

У безсерверній архітектурі серверна логіка програми розбита на окремі функції або мікросервіси. Ці функції керуються подіями та запускаються певними подіями чи запитами, такими як HTTP-запити, оновлення бази даних або заплановані завдання. Безсерверна платформа автоматично масштабує ресурси, необхідні для виконання цих функцій, забезпечуючи оптимальну продуктивність і економічну ефективність.

Завдяки безсерверній розробці розробники звільняються від керування серверами, операційними системами та мережевою інфраструктурою. Хмарний постачальник піклується про надання, масштабування та керування базовою інфраструктурою на основі попиту та використання функцій. Це дозволяє розробникам зосередитися на основній функціональності веб органайзера, не турбуючись про проблеми інфраструктури.

Дані платформи зазвичай дотримуються моделі ціноутворення з оплатою за використання, де розробникам виставляються рахунки на основі фактичного часу виконання та ресурсів, спожитих функціями. Ця економічно ефективна модель позбавляє від необхідності оплачувати час простою сервера та дозволяє організаціям оптимізувати свої витрати на основі фактичних моделей використання. Це також полегшує масштабування програми без попередніх інвестицій в інфраструктуру.

Архітектура, керована подіями розробка навколо подій і тригерів. Для веб-органайзерів події можуть включати дії користувача, оновлення даних або заплановані завдання. Коли відбувається подія, пов’язана функція запускається, виконує свою певну логіку та виробляє відповідь. Ця керована подіями архітектура забезпечує обробку в реальному часі, повну інтеграцію із зовнішніми службами та масштабованість для обробки різноманітних робочих навантажень.

Безсерверні архітектури добре піддаються інтеграції зі сторонніми службами та API. Веб-органайзерам часто потрібні такі функції, як автентифікація, зберігання файлів, обмін повідомленнями або аналіз даних. Безсерверні платформи забезпечують готову інтеграцію з різними службами,

що дозволяє розробникам легко включати ці функції у свої органайзери без необхідності складного налаштування чи обслуговування.

Швидка розробка та розгортання. Розробники можуть зосередитися на написанні модульних багаторазових функцій, які інкапсулюють певні функції. Ці функції можна розробляти, тестувати та розгортати незалежно, сприяючи гнучкості та забезпечуючи безперервну інтеграцію та розгортання (CI/CD). Безсерверна платформа керує розгортанням, масштабуванням і доступністю функцій, гарантуючи, що веб-органайзер завжди доступний і швидко реагує.

Масштабованість і стійкість: безсерверна архітектура за своєю суттю підтримує автоматичне масштабування на основі робочого навантаження. Оскільки попит на веб платформу зростатиме, безсерверна платформа динамічно надаватиме ресурси для обробки збільшеного трафіку та запитів. Ця масштабованість гарантує, що застосунок зможе впоратися з раптовими сплесками використання без зниження продуктивності. Крім того, безсерверні платформи часто забезпечують вбудовану відмовостійкість і стійкість, автоматично обробляючи збої та забезпечуючи високу доступність програми.

Зауваження щодо прив’язки до постачальника. При застосуванні безсерверного підходу важливо враховувати потенційну прив’язаність до постачальника. Хоча безсерверні платформи пропонують зручність і ефективність, вони можуть мати власні функції або залежності, які ускладнюють перехід до іншого постачальника. Бажано оцінити довгострокові наслідки та вибрати безсерверну платформу, яка відповідає вимогам проекту та забезпечує гнучкість і портативність.

Вцілому, безсерверний підхід до розробки веб системи пропонує численні переваги, зокрема скорочене управління інфраструктурою, оптимізацію витрат, масштабованість і швидші цикли розробки. Використовуючи безсерверні платформи та зосереджуючись на написанні модульних функцій, розробники можуть створити ефективні та масштабовані

веб системи, насолоджуючись перевагами автоматичного масштабування, ціноутворення за використання та спрощених процесів розгортання.

1.7 Фреймворки і бібліотеки

Підхід до розробки веб органайзера на основі фреймворків і бібліотек передбачає використання вже існуючих фреймворків і бібліотек програмного забезпечення для оптимізації процесу розробки, підвищення продуктивності та використання повторно використовуваних компонентів. Ці фреймворки та бібліотеки надають набір інструментів, попередньо створених функціональних можливостей і шаблонів розробки, які спрощують реалізацію веб-органайзерів.

Фреймворки веб-розробки забезпечують структурований спосіб створення веб систем, пропонуючи основу багаторазового коду, стандартні угоди та найкращі практики. Ці структури часто включають такі функції, як маршрутизація, створення шаблонів, інтеграція бази даних, аутентифікація та керування сеансами. Приклади популярних фреймворків веб-розробки включають Django (Python), Ruby on Rails (Ruby), Laravel (PHP) і Express.js (JavaScript/Node.js).

Фронтенд-фреймворки зосереджуються на вдосконаленні інтерфейсу користувача та взаємодії з веб-органайзером. Вони надають інструменти, компоненти та шаблони для створення адаптивних, інтерактивних і візуально привабливих інтерфейсів. Фронтенд-фреймворки, такі як React, Angular і Vue.js, пропонують ефективні компонентні архітектури, рішення для керування станом і потужні можливості візуалізації. Вони дозволяють розробникам створювати динамічні користувацькі інтерфейси та ефективніше обробляти складну інтерфейсну логіку.

Бібліотеки компонентів інтерфейсу користувача пропонують попередньо розроблені та готові компоненти, які можна легко інтегрувати у веб-органайзери. Ці бібліотеки забезпечують узгоджений і візуально приємний інтерфейс користувача, заощаджуючи час і зусилля розробників на

проектуванні та реалізації елементів інтерфейсу користувача. Приклади популярних бібліотек компонентів інтерфейсу користувача включають Bootstrap, Material-UI, Tailwind CSS і Semantic UI.

Внутрішні бібліотеки та модулі надають готові до використання функціональні можливості для певних вимог у веб-органайзерах. Ці бібліотеки можуть виконувати такі завдання, як підключення до бази даних, перевірка даних, керування файлами, інтеграція електронної пошти та розробка API. Вони дозволяють розробникам зосередитися на основній функціональності органайзера, не винаходячи велосипеда. Приклади включають SQLAlchemy (Python ORM), Mongoose (бібліотека MongoDB для Node.js) і nodemailer (бібліотека Node.js для електронної пошти).

Засоби для запуску завдань та інструменти для створення автоматизують повторювані завдання та спрощують робочий процес розробки. Вони можуть виконувати такі завдання, як компіляція препроцесорів CSS, об’єднання файлів JavaScript, оптимізація активів, виконання тестів і розгортання органайзера. Популярні інструменти в цій категорії включають Gulp, Grunt, webpack і Parcel. Вони підвищують продуктивність і забезпечують ефективне керування кодом і процеси розгортання.

Фреймворки для тестування допомагають забезпечити якість і надійність веб-органайзера, надаючи інструменти та методології для автоматизованого тестування. Ці фреймворки дозволяють розробникам писати модульні тести, інтеграційні тести та наскрізні тести для перевірки функціональності та продуктивності органайзера. Приклади тестових фреймворків включають Jest, PHPUnit, Selenium і Cypress.

Інструменти керування залежностями спрощують процес встановлення, оновлення та керування зовнішніми бібліотеками та залежностями у веб-органайзера. Ці інструменти автоматично обробляють конфлікти керування версіями та гарантують, що необхідні залежності є

легкодоступними. Поширені інструменти керування залежностями включають npm (Node Package Manager), Composer (PHP) і pip (Python).

Використовуючи фреймворки та бібліотеки, розробники можуть пришвидшити процес розробки, підтримувати узгодженість коду та отримувати користь від усталених передових практик. Вони пропонують готові рішення для стандартних вимог до органайзера на основі веб системи, що скорочує час і зусилля на розробку. Однак дуже важливо ретельно вибирати відповідні фреймворки та бібліотеки на основі вимог проекту, підтримки спільноти, документації та довгострокової придатності до обслуговування.

РОЗДІЛ 2 АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ОРГАНАЙЗЕРА НА ОСНОВІ ВЕБ СИСТЕМИ

2.1 Архітектурні рішення органайзера на основі веб системи

При проектуванні веб-системи необхідно враховувати кілька архітектурних рішень. Вибір архітектури залежить від таких факторів, як масштабованість, продуктивність, безпека, досвід команди розробників і конкретні вимоги до системи.

2.1.1 Монолітна архітектура

У монолітній архітектурі вся програма побудована як єдине самостійне ціле. Усі компоненти, включаючи інтерфейс користувача, бізнес-логіку та рівень доступу до даних, тісно інтегровані. Цю архітектуру відносно просто розробити та розгорнути, але її масштабування та обслуговування може стати складним у міру зростання системи.

2.1.2 Мікросервісна архітектура

У мікросервісній архітектурі система розділена на набір слабко пов’язаних і незалежно розгорнутих сервісів. Кожна послуга зосереджена на певних бізнес-можливостях і може бути розроблена, розгорнута та масштабована незалежно. Зв’язок між службами зазвичай здійснюється за допомогою спрощених протоколів, таких як REST або черги обміну повідомленнями. Ця архітектура сприяє масштабованості, гнучкості та дозволяє командам працювати над різними службами одночасно.

2.1.3 Serverless архитектура

Безсерверна архітектура, також відома як функція як послуга (FaaS), абстрагується від управління інфраструктурою та дозволяє розробникам зосередитися на написанні окремих функцій або мікросервісів. Хмарний постачальник керує інфраструктурою та автоматично масштабує функції на основі попиту. Ця архітектура зменшує операційні накладні витрати та може бути економічно ефективною для систем зі змінним навантаженням.

2.1.4 Сервісно-орієнтована архітектура (SOA)

Сервісно-орієнтована архітектура зосереджується на розділенні функціональних можливостей застосунків на слабозв’язані служби, які можна багаторазово використовувати. Служби спілкуються один з одним за допомогою стандартизованих протоколів, таких як SOAP або REST. Ця архітектура сприяє багаторазовому використанню, гнучкості та сумісності між різними платформами та технологіями.

2.1.5 Архітектура, керована подіями

Архітектура, керована подіями (EDA), фокусується на зв’язку між компонентами через події. Коли відбувається подія, вона запускає виконання певних дій або процесів. Ця архітектура корисна для систем з асинхронними робочими процесами, оновленнями в реальному часі та складними вимогами до обробки подій.

2.1.6 Багаторівнева архітектура

Багаторівнева архітектура поділяє систему на окремі рівні, такі як рівень презентації, рівень бізнес-логіки та рівень доступу до даних. Кожен рівень має певні обов’язки та взаємодіє з сусідніми рівнями відповідно до визначених протоколів. Ця архітектура сприяє поділу завдань, модульності та зручності обслуговування.

2.1.7 Архітектура клієнт-сервер

Архітектура клієнт-сервер передбачає поділ системи на клієнтську програму, відповідальну за взаємодію користувачів, і серверну програму, яка обробляє бізнес-логіку та зберігання даних. Ця архітектура дозволяє створювати масштабовані та розподілені системи за рахунок використання кількох серверів для обробки запитів клієнтів.

2.1.8 Архітектура однорангового зв’язку

Архітектура однорангового зв’язку забезпечує децентралізований зв’язок і обмін даними між рівноправними партнерами. Кожен одноранговий вузол може діяти як клієнт і сервер, дозволяючи прямий зв’язок і спільне використання ресурсів, не покладаючись на централізований сервер.

Важливо відзначити, що ці архітектурні рішення можна комбінувати або адаптувати для задоволення конкретних вимог веб-системи. Крім того, нові архітектурні шаблони та технології, такі як джерело подій, CQRS (Command Query Responsibility Segregation) і GraphQL, пропонують альтернативні підходи для вирішення конкретних проблем у розробці веб-систем. Вибір найбільш прийнятної архітектури вимагає ретельного аналізу потреб системи, вимог до масштабованості, ресурсів розробки та довгострокових цілей.

Серед проаналізованих видів архітектури, для побудови бажаної системи я зупинив свій вибір на клієнт-серверній архітектурі адже це фундаментальний архітектурний шаблон, який зазвичай використовується в веб-системах. Він передбачає поділ системи на два основні компоненти: клієнт і сервер.

Клієнт - відноситься до пристрою або програмного забезпечення, яке надсилає запити серверу та отримує відповіді. Клієнт, як правило, є компонентом, зверненим до користувача, відповідальним за ініціювання зв’язку з сервером для доступу до послуг, даних або ресурсів:

- клієнти можуть приймати різні форми, включаючи веб-браузери, мобільні програми, настільні програми або пристрої IoT. Вони призначені для взаємодії з сервером і надання інформації або виконання певних завдань для задоволення потреб користувачів.
- клієнти надсилають запити на сервер, вказуючи бажану операцію або дані, які їм потрібні. Запити можуть бути у формі HTTP-запитів або інших протоколів залежно від вимог програми.
- клієнти часто надають інтерфейс користувача (UI) для полегшення взаємодії з користувачем. Вони можуть візуалізувати веб-сторінки, відображати дані та дозволяти введення користувачами, роблячи взаємодію з сервером безперебійною та інтуїтивно зрозумілою.

- клієнти можуть виконувати різноманітні операції локально перед тим, як надсилати запити на сервер. Це включає перевірку вхідних даних, перетворення даних або рендеринг на стороні клієнта для покращення взаємодії з користувачем і зменшення навантаження на сервер.
- щоб спростити процес розробки, клієнтські бібліотеки та API часто надаються компонентами на стороні сервера. Ці бібліотеки пропонують попередньо визначені функції та методи, які дозволяють розробникам ефективніше взаємодіяти з сервером.
- клієнти можуть підтримувати інформацію про стан, таку як налаштування користувача, дані сеансу або кешовані дані, які можуть зберігатися локально на пристрої або у файлах cookie. Це дозволяє клієнту запам’ятати попередні взаємодії та зберегти персоналізований досвід.
- клієнти відіграють вирішальну роль у забезпеченні безпеки в архітектурі клієнт-сервер. Вони можуть впроваджувати механізми аутентифікації, обробляти шифрування даних і перевіряти введені користувачем дані, щоб запобігти вразливостям безпеки.
- загалом клієнти діють як посередники між користувачами та серверами, дозволяючи користувачам отримувати доступ до серверних ресурсів і взаємодіяти з ними. Вони відповідають за ініціювання запитів, обробку відповідей і представлення даних або послуг у зручній для користувача формі, зрештою сприяючи ефективній комунікації в архітектурі клієнт-сервер.

Сервер - відноситься до комп’ютера або програмного забезпечення, яке надає послуги, ресурси або дані клієнтам. Сервери обробляють запити клієнтів, обробляють їх і надсилають відповідні відповіді:

- існують різні типи серверів, кожен з яких служить певній меті. Деякі типові приклади включають веб-сервери, файлові сервери,

сервери баз даних, поштові сервери та сервери застосунків. Кожен тип спеціалізується на наданні відповідних послуг або ресурсів.

- сервери прослуховують вхідні запити від клієнтів і відповідають відповідним чином. Вони призначені для одночасної обробки кількох клієнтських підключень, використовуючи такі методи, як багатопотокове програмування, програмування, кероване подіями, або асинхронні операції для ефективного керування запитами клієнтів.
- сервери зазвичай розміщують бізнес-логіку та виконують завдання обробки даних. Вони отримують запити клієнтів, обробляють дані, взаємодіють з базами даних або іншими ресурсами та генерують відповідні відповіді, які надсилаються клієнтам.
- сервери мають бути розроблені та налаштовані для обробки різних рівнів трафіку та масштабування в міру зростання кількості клієнтів. Це може включати методи балансування навантаження, кластеризацію, кешування або розподілене обчислення для забезпечення високої продуктивності та доступності.
- сервери відіграють вирішальну роль у забезпеченні заходів безпеки. Вони обробляють автентифікацію та авторизацію запитів клієнтів, перевіряють вхідні дані, захищають від потенційних атак і захищають передачу даних за допомогою протоколів шифрування (наприклад, HTTPS).
- Сервери часто надають інтерфейси прикладного програмування (API), які визначають, як клієнти можуть взаємодіяти з сервером і отримувати доступ до його послуг або ресурсів. Зв’язок між клієнтами та серверами забезпечується за допомогою таких

протоколів, як HTTP, WebSocket або інших спеціалізованих протоколів залежно від конкретних вимог.

- сервери побудовані з використанням різних внутрішніх технологій, таких як мови програмування (наприклад, Java, Python, Node.js), фреймворки, бібліотеки та бази даних. Ці технології дозволяють розробникам створювати надійні та масштабовані серверні програми.
- сервери часто зберігають дані та керують ними за допомогою баз даних або інших систем зберігання. Вони взаємодіють із рівнем даних, щоб виконувати операції читання та запису, підтримувати узгодженість даних і гарантувати цілісність даних.
- для забезпечення безперебійної роботи серверів потрібен моніторинг. Інструменти моніторингу відстежують продуктивність сервера, використання ресурсів і виявляють будь-які проблеми або вузькі місця. Регулярне технічне обслуговування, оновлення та виправлення безпеки необхідні для забезпечення оптимальної роботи серверів.
- Загалом, сервери утворюють основу клієнт-серверної архітектури, надаючи послуги та ресурси клієнтам. Вони обробляють запити клієнтів, обробляють дані, забезпечують дотримання заходів безпеки та сприяють ефективній комунікації та взаємодії в системі клієнт-сервер.

Основи клієнт-серверної архітектури:

1. Зв’язок: Клієнт і сервер спілкуються через мережу, як правило, за допомогою таких протоколів, як HTTP або TCP/IP. Клієнт надсилає запити серверу, а сервер відповідає запитуваними даними або виконує необхідні операції.
2. Відокремлення інтересів: клієнт несе відповідальність за обробку інтерфейсу користувача, взаємодії з користувачем і логіку презентації.

Сервер керує бізнес-логікою, зберіганням і обробкою даних, забезпечуючи чіткий розподіл завдань між зовнішніми та внутрішніми компонентами.

3. Масштабованість: архітектура клієнт-сервер забезпечує масштабованість шляхом розподілу робочого навантаження між декількома серверами. Зі збільшенням кількості клієнтів можна додавати додаткові серверні ресурси, щоб задовольнити підвищений попит.

Плюси клієнт-серверної архітектури:

1. Масштабованість: розподіляючи навантаження обробки між клієнтом і сервером, клієнт-серверна архітектура дозволяє створювати масштабовані системи, які можуть обслуговувати велику кількість одночасних користувачів.
2. Гнучкість. Розподіл проблем дає змогу незалежно розробляти та розвивати клієнтські та серверні компоненти. Оновлення або зміни одного компонента можна вносити, не впливаючи на інший, що забезпечує гнучкість обслуговування та оновлення системи.
3. Безпека: архітектура клієнт-сервер надає можливості для реалізації заходів безпеки на стороні сервера, таких як автентифікація, авторизація та шифрування. Це допомагає захистити конфіденційні дані та забезпечує безпечний зв’язок між клієнтами та сервером.
4. Централізоване зберігання даних: завдяки серверу, який керує зберіганням даних, стає легше підтримувати та контролювати послідовність і цілісність даних. Це забезпечує ефективне керування даними, процеси резервного копіювання та відновлення.

Мінуси клієнт-серверної архітектури:

1. Збільшення мережевого трафіку: зв’язок між клієнтом і сервером залежить від підключення до мережі. Це може призвести до збільшення мережевого трафіку, особливо для програм із інтенсивним об’ємом даних, що вплине на продуктивність системи.

2. Єдина точка відмови: якщо сервер виходить з ладу або перестає працювати, уся система може стати недоступною для клієнтів. Для пом’якшення цього ризику необхідно застосувати відповідні заходи резервування та відмовостійкості.
3. Проблеми масштабованості. Масштабування серверної сторони клієнт-серверної архітектури може бути складнішим порівняно з масштабуванням клієнтської сторони. Балансування навантаження, кластеризація та методи розподіленого обчислення можуть знадобитися для забезпечення ефективного розподілу ресурсів і задоволення підвищеного попиту користувачів.

Ключові особливості клієнт-серверної архітектури:

1. Модель «запит-відповідь»: клієнти надсилають запити на сервер, а сервер відповідає запитуваними даними або виконує необхідні операції.
2. Зв'язок без збереження стану: сервер не зберігає жодної інформації про стан клієнта між запитами. Кожен запит від клієнта обробляється незалежно, що полегшує масштабування та розподіл запитів.
3. Презентація на стороні клієнта: клієнтський компонент обробляє інтерфейс користувача, включаючи логіку презентації та взаємодію з користувачем.
4. Обробка на стороні сервера: серверний компонент керує бізнес-логікою, зберіганням і обробкою даних. Він обробляє запити клієнтів, виконує необхідні обчислення та отримує або оновлює дані з основного сховища даних.

Архітектура клієнт-сервер отримала широке поширення і є основою багатьох веб-систем. Він забезпечує структурований підхід до побудови масштабованих, гнучких і безпечних систем, розділяючи проблеми та використовуючи можливості як клієнтського, так і серверного компонентів.

2.2 Програмні засоби для реалізації клієнтської частини

Існує кілька програмних інструментів і технологій, доступних для створення клієнтської сторони архітектури клієнт-сервер. Ці інструменти полегшують розробку інтерфейсів користувача, обробку логіки на стороні клієнта та взаємодію з сервером.

HTML, CSS і JavaScript (JS) відіграють важливу роль як клієнтські компоненти в моделі сервер-клієнт.

HTML (мова розмітки гіпертексту) є основою веб-сторінок і забезпечує структуру та вміст клієнтського інтерфейсу. Він визначає різні елементи, такі як заголовки, абзаци, форми, зображення та посилання. Теги HTML організовують і представляють вміст у структурований спосіб. CSS (каскадні таблиці стилів): CSS використовується для керування представленням і стилем елементів HTML. Він визначає візуальні аспекти інтерфейсу користувача, включаючи кольори, макети, шрифти та адаптивний дизайн. CSS дозволяє розробникам створювати візуально привабливі та узгоджені дизайни для різних пристроїв і розмірів екранів. JavaScript: JavaScript — це потужна мова сценаріїв, яка працює на стороні клієнта та забезпечує інтерактивність і динамічну поведінку веб-сторінок. Він додає такі функції до клієнтського інтерфейсу, як перевірка форми, анімація, обробка подій і маніпуляції DOM. JavaScript може спілкуватися з сервером через запити AJAX, дозволяючи отримувати та оновлювати дані без перезавантаження всієї сторінки.

Фронтенд фреймворки відіграють вирішальну роль як клієнтський компонент у моделі сервер-клієнт. Вони забезпечують структурований та ефективний спосіб побудови складних інтерфейсів користувача та вдосконалення процесу розробки.

Фреймворки Frontend, такі як React, Angular і Vue.js, пропонують компонентну архітектуру, яка допомагає структурувати та організовувати кодову базу на стороні клієнта. Компоненти інкапсулюють елементи

інтерфейсу користувача, логіку та дані, що полегшує керування та підтримку складних інтерфейсів користувача.

Фреймворки сприяють багаторазовому використанню, дозволяючи розробникам створювати багаторазові компоненти. Ці компоненти можна використовувати в різних частинах програми, що забезпечує ефективність і узгодженість коду. Багаторазові компоненти заощаджують час і зусилля на розробку, підвищуючи продуктивність.

Управління станом: керування та оновлення стану програми є критичним аспектом розробки на стороні клієнта. Фронтенд-фреймворки часто надають вбудовані або рекомендовані рішення для управління станом. Ці рішення допомагають керувати даними програми та забезпечують узгоджену роботу компонентів.

Багато фреймворків використовують віртуальну DOM (модель об’єктів документа) для ефективного оновлення інтерфейсу користувача. Замість того, щоб безпосередньо маніпулювати фактичною DOM, структура порівнює зміни у віртуальній DOM і застосовує лише необхідні оновлення до реальної DOM, зменшуючи непотрібне повторне відтворення та покращуючи продуктивність.

Маршрутизація та навігація. Фронтенд фреймворки часто включають можливості маршрутизації, уможливлюючи навігацію на стороні клієнта та здатність обробляти різні перегляди або сторінки в програмі. Це забезпечує безперебійну роботу користувача шляхом динамічного завантаження вмісту без перезавантаження сторінок.

Інтеграція з API даних: фреймворки полегшують інтеграцію API даних і серверних служб за допомогою запитів AJAX або інших методів зв’язку. Вони забезпечують механізми для надсилання HTTP-запитів до сервера, отримання даних і оновлення інтерфейсу користувача на стороні клієнта на основі отриманих даних.

Фронтенд фреймворки постачаються з багатою екосистемою інструментів, бібліотек і підтримки спільноти. Ця екосистема надає

інструменти розробки, утиліти для налагодження, інфраструктури тестування та розширення, які покращують робочий процес розробки та покращують якість коду.

Вцілому фронтенд фреймворки пропонують структурований підхід до розробки на стороні клієнта в моделі сервер-клієнт. Вони спрощують процес розробки, надають архітектурні шаблони та пропонують функції, які покращують продуктивність, зручність обслуговування та продуктивність. Використовуючи фронтенд фреймворки, розробники можуть створювати інтерактивні та адаптивні користувацькі інтерфейси, бездоганно інтегруючись із компонентами на стороні сервера.

Мною було проаналізовано і порівняно три найпопулярніші фронтенд фреймворки:

Angular, Vue та React

Плюси Angular:

1. Комплексна структура: Angular надає повне рішення з вбудованими інструментами, маршрутизацією, керуванням станом і впровадженням залежностей, що робить його придатним для великомасштабних програм.

2. Підтримка TypeScript: Angular побудовано на TypeScript, що забезпечує такі переваги, як статичний тип, кращі інструменти та покращену підтримку коду.

3. Потужний CLI: Angular CLI пропонує потужний інтерфейс командного рядка, який спрощує налаштування проекту, генерацію коду, тестування та процеси розгортання.

4. Надійність і масштабованість: впевнена архітектура та шаблони дизайну Angular сприяють організації коду, зручності обслуговування та масштабованості для складних програм.

5. Активна спільнота та підтримка: Angular має велике та активне співтовариство, яке надає велику кількість ресурсів, навчальних посібників та варіантів підтримки для розробників.

6. Офіційно підтримується Google: Angular розробляється та підтримується Google і користується підтримкою авторитетної організації, що забезпечує його постійний розвиток і підтримку.

7. Двостороннє прив’язування даних: двостороннє прив’язування даних Angular спрощує синхронізацію даних між моделлю та представленням, зменшуючи шаблонний код.

8. Розширені можливості тестування: Angular надає потужні можливості тестування за допомогою таких інструментів, як Karma та Protractor, що полегшує написання модульних і наскрізних тестів.

9. Доступність: Angular приділяє велике значення функціям доступності, надаючи вбудовану підтримку для створення доступних веб-застосунків.

10. Корпоративне впровадження: Angular часто віддають перевагу великим підприємствам і компаніям зі складними вимогами до програм, оскільки він пропонує чітко визначену структуру та потужні функції.

Мінуси Angular:

1. Складний у вивченні: Angular складний у вивченні через свою комплексну природу та потребу вивчити TypeScript, що може вимагати додаткових зусиль і часу для розробників, які не знайомі з фреймворком.

2. Багатослівність: Angular може бути більш докладним порівняно з іншими фреймворками, що призводить до довшого коду та потенційного збільшення часу розробки.

3. Складність: впевнений підхід Angular і великий набір функцій можуть призвести до підвищення складності, особливо для невеликих проектів, які не вимагають повного спектру його можливостей.

4. Накладні витрати на продуктивність: обширний фреймворк Angular може призвести до деяких накладних витрат на продуктивність порівняно з більш легкими фреймворками, особливо для невеликих застосунками із простішими вимогами.

5. Великий розмір комплекту: за замовчуванням застосунки Angular зазвичай мають більший розмір пакета порівняно з іншими фреймворками, що може вплинути на початковий час завантаження.

6. Крива навчання впровадження залежностей: Хоча впровадження залежностей в Angular приносить переваги, воно вимагає від розробників розуміння концепцій і належного їх використання.

7. Крива навчання для тестування: Інфраструктура тестування та налаштування Angular можуть бути складними для розробників, які тільки починають працювати з фреймворком або тестуванням загалом.

8. Більше думок: впевнений підхід Angular може обмежити гнучкість і можливості налаштування, особливо для розробників, які віддають перевагу більшій свободі у виборі архітектури.

9. Міграція та сумісність версій: у міру розвитку версій Angular міграція між основними версіями або робота з сумісністю версій може бути складною і трудомісткою.

10. Повільніший цикл випуску: Angular має повільніший цикл випуску порівняно з іншими фреймворками, що може призвести до більш тривалого очікування нових функцій і оновлень.

Плюси Vue:

1. Легка у вивченні: Vue легкий у вивченні, що робить його доступним для початківців і розробників, які тільки знайомляться з інтерфейсними фреймворками.

2. Легкість: Vue — це легкий фреймворк, що означає швидший час завантаження та покращену продуктивність порівняно з більшими фреймворками.

3. Гнучка та адаптована: модульна архітектура Vue дозволяє розробникам поступово адаптувати її до існуючих проектів або використовувати певні частини фреймворку за потреби.

4. Універсальність: Vue можна використовувати для створення різних типів програм, від односторінкових програм (SPA) до складних веб-інтерфейсів.

5. Реактивність та ефективність: система реагування Vue ефективно оновлює та відтворює компоненти, забезпечуючи плавну та чутливу роботу користувача.

6. Vue CLI: Vue надає офіційний інтерфейс командного рядка (CLI), який спрощує налаштування проекту, скелети та завдання з обслуговування.

7. Компонентна розробка: Компонентна архітектура Vue сприяє багаторазовому використанню, модульності та полегшенню обслуговування коду.

8. Vue Router: Vue має офіційну бібліотеку маршрутизаторів (Vue Router), яка забезпечує прості та гнучкі можливості маршрутизації для створення SPA.

9. Велика та підтримуюча спільнота: Vue має зростаючу та підтримуючу спільноту, яка надає численні ресурси, навчальні посібники та плагіни для допомоги розробникам.

10. Інтеграція з існуючими проектами: Vue можна легко інтегрувати в існуючі проекти або співіснувати разом з іншими фреймворками, що забезпечує плавну міграцію та поступове впровадження.

Мінуси Vue:

1. Менша екосистема: порівняно з більшими фреймворками, такими як Angular і React, екосистема Vue може мати менше офіційних бібліотек і інструментів. Проте спільнота активно розвивається та розширює наявні ресурси.

2. Розвиненість: хоча Vue набув значної популярності, деякі організації можуть сприймати його як менш розвинений порівняно з усталеними фреймворками.

3. Обмежене впровадження на підприємствах: хоча Vue набирає обертів, він може бути не настільки широко прийнятий в організаціях корпоративного рівня порівняно з Angular або React.

4. Менше можливостей роботи: хоча популярність Vue зростає, кількість можливостей роботи, спеціально націлених на Vue, може бути відносно меншою порівняно з іншими фреймворками.

5. Навчальні ресурси іншими мовами. Незважаючи на те, що доступно багато навчальних ресурсів англійською мовою, доступність ресурсів іншими мовами може відрізнятися.

6. Розмір підтримки спільноти: хоча спільнота Vue зростає, вона може бути не такою великою чи активною, як спільноти навколо Angular або React.

7. Інструменти та плагіни: хоча Vue пропонує широкий вибір інструментів і плагінів, вибір може бути не таким широким, як у більших фреймворків.

8. Строгість настанов: гнучкість Vue може бути палкою з двома кінцями. Без суворих вказівок розробники можуть мати більше свободи, але також ризикувати невідповідностями в структурі проекту та організації коду.

9. Повільніше оновлення: цикл випуску Vue, порівняно з іншими фреймворками, може призвести до повільніших оновлень і впровадження нових функцій.

10. Обмежена офіційна підтримка: хоча основна команда забезпечує підтримку фреймворку Vue, розробники можуть зіткнутися з проблемами, коли шукають офіційну підтримку для певних плагінів або бібліотек сторонніх виробників.

Плюси React:

1. Компонентна архітектура: Компонентна архітектура React сприяє багаторазовому використанню, модульності та легшому супроводу коду.

2. Віртуальний DOM: React використовує віртуальний DOM, який оптимізує рендеринг і покращує продуктивність шляхом мінімізації фактичних маніпуляцій з DOM.

3. Велика та активна спільнота: React має велику та активну спільноту, яка надає великі ресурси, бібліотеки та підтримку, керовану спільнотою.

4. React Native: Екосистема React включає React Native, що дозволяє розробляти мобільні програми за допомогою знайомого синтаксису та принципів React.

5. Односпрямований потік даних: React забезпечує односпрямований потік даних, полегшуючи розуміння та налагодження змін стану програми.

6. JSX: Синтаксис JSX React дозволяє розробникам писати компоненти, використовуючи комбінацію JavaScript і HTML-подібного синтаксису, в результаті чого код стає більш виразним і читабельним.

7. Висока продуктивність: Ефективний алгоритм візуалізації React і віртуальний DOM сприяють його високій продуктивності, що робить його придатним для складних і інтенсивних програм.

8. Інструменти, керовані спільнотою: популярність React призвела до створення надійної екосистеми інструментів, бібліотек і фреймворків, які бездоганно інтегруються з проектами React.

9. Підтримка Facebook: React розроблено та підтримується Facebook, що забезпечує постійну підтримку, оновлення та постійне вдосконалення.

10. Багаторазові компоненти: Компонентна природа React дозволяє створювати повторно використовувані компоненти інтерфейсу користувача, забезпечуючи швидшу розробку та узгоджений дизайн у проектах.

Мінуси React:

1. Крута крива навчання: у React є крива навчання, особливо для розробників, які не знайомі з компонентною архітектурою та синтаксисом JSX.

2. Шаблонний код: React може вимагати написання більшого коду порівняно з більш простими фреймворками, що може призвести до збільшення часу розробки та потенційної багатослівності.

3. Проблеми інтеграції: інтеграція React в існуючі проекти або робота разом з іншими фреймворками може вимагати додаткових налаштувань і налаштувань.

4. Відсутність нав’язливості: React — це бібліотека, а не повноцінний фреймворк, що означає, що розробники мають більше свободи, але їм також потрібно самостійно приймати архітектурні рішення.

5. Складність інструментів: хоча React має багату екосистему, вибір правильної комбінації інструментів і бібліотек може бути надзвичайно складним для новачків.

6. Обмежена офіційна підтримка: Офіційна підтримка від команди React може не охоплювати всі аспекти сторонніх бібліотек або плагінів, які використовуються в проектах React.

7. Відсутність стандартизованих практик: гнучкість React може призвести до варіацій у структурі проекту та методах кодування, що може вимагати додаткової координації в налаштуваннях команди.

8. Велика кількість вибору: широкий спектр доступних бібліотек та інструментів для React може ускладнити вибір найбільш підходящих для конкретних вимог проекту.

9. Часті оновлення. Цикл активної розробки та випуску React може вимагати від розробників бути в курсі останніх змін, що може викликати проблеми сумісності.

10. Залежність JSX: використання React JSX вимагає етапу збірки та транспіляції, що може ускладнити процес розробки та розгортання.

Серед вищезазначених фреймворків мій вибір зупиняється на Angular через те, що він використовує TypeScript, що дозволяє писати код більш безпечно. Також фактор, що сприяв вибору на користь Angular є те, що даний фреймворк має високу стійкість і гарну масштабованість, що у майбутньому дозволить простіше розширювати веб застосунок.

2.3 Програмні засоби для реалізації серверної частини

Існує велика кількість інструментів і технологій, які зазвичай використовуються для створення серверної програми клієнт-серверу.

До переліку цих інструментів відносяться серверні фреймворки. Вони забезпечують структурований і ефективний спосіб створення та розгортання серверних програм. Ці фреймворки зазвичай обслуговують основні функції, необхідні для розробки веб-застосунків, API та інших служб на стороні сервера.

Нижче наведено деякі поширені серверні фреймворки разом із їхніми основними функціями та принципом роботи:

Node.js (Express.js, Koa.js):

Основні функції: введення/виведення, кероване подіями, асинхронне програмування, обробка HTTP-запитів, маршрутизація, підтримка проміжного ПЗ та інтеграція з базами даних.

Як вони працюють: фреймворки Node.js використовують керовану подіями неблокуючу модель введення-виведення, що робить їх високомасштабованими та ефективними для обробки одночасних запитів. Вони використовують JavaScript як мову програмування та працюють на двигуні V8 JavaScript.

Python (Django, FastAPI):

Основні функції: обробка HTTP-запитів, маршрутизація, рендеринг шаблонів, інтеграція бази даних (ORM), керування сеансами, автентифікація та безпека.

Як вони працюють: фреймворки Python, такі як Django та FastAPI, використовують протокол WSGI (інтерфейс шлюзу веб-сервера) для зв’язку між веб-серверами та фреймворком. Вони надають абстракції та утиліти для оптимізації розробки веб-застосунків.

Java (Spring Boot):

Основні функції: впровадження залежностей, архітектура MVC (Model-View-Controller), обробка запитів, маршрутизація, інтеграція бази даних (ORM), безпека та керування транзакціями.

Як вони працюють: фреймворки Java, такі як Spring Boot, використовують API сервлетів Java для обробки запитів HTTP. Вони використовують анотації, конфігурацію XML і принципи конфігурації, щоб спростити розробку та забезпечити надійні функції корпоративного рівня.

Ruby (Ruby on Rails, Sinatra):

Основні функції: Конвенційне налаштування, архітектура MVC, обробка HTTP-запитів, маршрутизація, ORM (об’єктно-реляційне відображення), рендеринг шаблонів та інтеграція бази даних.

Як вони працюють: фреймворки Ruby, такі як Ruby on Rails і Sinatra, дотримуються принципів конвенції над конфігурацією, прагнучи до простоти та швидкого розвитку. Вони використовують інтерфейс Rack для зв’язку між веб-сервером і фреймворком.

C# (.NET, ASP.NET):

Основні функції: обробка запитів HTTP, маршрутизація, архітектура MVC, інтеграція бази даних (ORM), керування сесіями, безпека та автентифікація.

Як вони працюють: фреймворки C#, такі як ASP.NET, використовують середовище виконання .NET і середовище виконання загальної мови (CLR) для створення веб-застосунків. Вони використовують конвеєр ASP.NET для обробки запитів і пропонують різноманітні функції та інструменти для розробки застосунків на рівні підприємства.

Серед розглянутих фреймворків мій вибір зупинився на FastAPI через наступні причини

1. Висока продуктивність: FastAPI побудовано на основі Starlette, яка відома своєю високою продуктивністю та асинхронними можливостями. Він використовує синтаксис async і await Python для ефективної обробки запитів,

що робить його придатним для вимогливих програм, які вимагають високої пропускної здатності та низької затримки.

2. Швидка розробка: FastAPI пропонує чистий та інтуїтивно зрозумілий синтаксис із автоматичною серіалізацією та перевіркою даних. Він підтримує підказки типів Python, що дає змогу розробникам завчасно виявляти помилки та писати код, який зручно підтримувати. Автоматичне створення документації API FastAPI також заощаджує час розробки.

3. Безпека типів: FastAPI використовує функцію підказки типів Python, що забезпечує кращу читабельність коду, покращену перевірку помилок і розширений досвід розробника. Підказки щодо типу також дозволяють автоматично перевіряти дані запиту та відповіді, зменшуючи ймовірність помилок під час виконання.

4. Проста інтеграція: FastAPI може легко інтегруватися з іншими бібліотеками та фреймворками Python, що робить його гнучким і адаптованим до різних вимог проекту. Він може працювати з популярними ORM (Object-Relational Mappers), фреймворками автентифікації, базами даних та іншими інструментами в екосистемі Python.

5. Асинхронна підтримка: FastAPI має вбудовану підтримку асинхронного програмування з використанням синтаксису Python async і await. Він може ефективно обробляти одночасні запити, що робить його придатним для застосунків, які потребують високої паралельності та масштабованості.

6. Аутентифікація та авторизація: FastAPI надає зручні функції для впровадження механізмів автентифікації та авторизації. Він підтримує різні методи автентифікації, такі як OAuth2 і JWT (веб-токени JSON), що полегшує безпеку кінцевих точок API і захист конфіденційних даних.

7. Автоматична документація API: FastAPI автоматично генерує інтерактивну документацію API на основі коду та анотацій. Документація містить детальну інформацію про моделі запитів і відповідей, доступні кінцеві точки, параметри тощо. Ця функція економить час і зусилля на

документуванні API та допомагає розробникам і споживачам зрозуміти, як ним користуватися.

8. Екосистема Python: FastAPI отримує переваги від величезної екосистеми Python, яка включає широкий спектр бібліотек, інструментів і підтримку спільноти. Ця екосистема надає додаткові ресурси, інтеграції та пакети сторонніх розробників, які можуть покращити процес розробки та розширити функціональність API.

9. Готові до виробництва функції: FastAPI включає такі готові до виробництва функції, як перевірка введення, обробка помилок, перевірка запитів тощо. Ці функції допомагають створювати надійні REST API, які можуть працювати зі сценаріями реального світу.

10. Спільнота та підтримка: FastAPI набув значної популярності та має активну спільноту розробників. Ця спільнота надає підтримку, ділиться знаннями та сприяє розвитку фреймворку, полегшуючи пошук ресурсів, навчальних посібників і допомоги, коли це необхідно.

Попри свої переваги, FastAPI має ряд недоліків, які варто зазначити

1. Процес вивчення: розширені функції FastAPI, такі як асинхронне програмування та впровадження залежностей, можуть бути складнішими у вивчення для розробників, які тільки починають працювати з веб-фреймворками Python.

2. Обмежена підтримка мов: FastAPI в основному зосереджено на Python, що означає, що він може не підходити для проектів, які вимагають багатомовної розробки або інтеграції з екосистемами, відмінними від Python.

3. Менша спільнота: хоча FastAPI набув популярності, її спільнота може бути відносно меншою порівняно з більш усталеними фреймворками, такими як Spring Boot або ASP.NET.

4. Обмежене впровадження на підприємствах: FastAPI може бути не настільки широко поширеним в організаціях корпоративного рівня порівняно з більш усталеними фреймворками в екосистемах Java або C#.

5. Документація та ресурси: хоча FastAPI надає автоматичну документацію API, доступність вичерпних ресурсів і навчальних посібників може бути відносно обмеженою порівняно з більш розвиненими фреймворками.

6. Розвиненість: FastAPI є відносно новою структурою порівняно з Spring Boot і ASP.NET, що може викликати занепокоєння щодо стабільності, довгострокової підтримки та загальної розвиненості екосистеми.

7. Обмежений інструментарій: інструментарій та екосистема навколо FastAPI можуть бути не такими обширними, як у більш усталених фреймворків, що призводить до меншого вибору та інтеграції.

8. Менша підтримка спільноти: хоча FastAPI має активну спільноту, рівень підтримки спільноти та доступні бібліотеки сторонніх розробників можуть бути нижчими порівняно з більш розвиненими фреймворками.

9. Складність розгортання: розгортання програм FastAPI може вимагати додаткової конфігурації та налаштування порівняно з більш сталими фреймворками, такими як Spring Boot, які надають вбудовані параметри розгортання.

10. Проблеми інтеграції: інтеграція FastAPI з існуючою інфраструктурою, застарілими системами чи іншими технологіями може вимагати додаткових зусиль і спеціальної інтеграційної роботи порівняно зі структурами з ширшими можливостями інтеграції.

Загалом FastAPI пропонує поєднання високої продуктивності, зручних для розробників функцій і легкої інтеграції з екосистемою Python, що робить його переконливим вибором для створення REST API.

Наступним, до переліку інструментів, які зазвичай використовуються для створення серверної програми клієнт-серверу, можна віднести базу даних адже вона є її важливим компонентом.

База даних забезпечує структурований і організований спосіб зберігання та керування даними, які використовуються веб-системою. Це забезпечує ефективне зберігання та пошук інформації, забезпечуючи

цілісність і послідовність даних. Багато веб-систем вимагають створення динамічного вмісту. База даних дозволяє зберігати та отримувати динамічні дані, такі як профілі користувачів, списки продуктів, публікації в блогах, коментарі тощо. Це дозволяє веб-системі надавати користувачам персоналізований та актуальний вміст. Бази даних часто використовуються для зберігання інформації про користувачів, включаючи імена користувачів, паролі, адреси електронної пошти та інші відповідні дані. Це забезпечує аутентифікацію користувачів, авторизацію та персоналізований досвід у веб-системі. Постійність даних: веб-системам зазвичай потрібно постійно зберігати дані, щоб підтримувати безперервність між сеансами користувача та перезавантаженням системи. Бази даних пропонують довговічне зберігання, яке гарантує, що дані залишаються доступними навіть після збою системи чи сервера. Бази даних забезпечують масштабованість, дозволяючи веб-системі обробляти зростаючі обсяги даних і трафік користувачів. Завдяки правильному дизайну та оптимізації баз даних веб-системи можуть ефективно масштабуватися в міру зростання бази користувачів. Вони також полегшують аналіз даних і звітність у веб-системі. Зберігаючи релевантні дані, система може виконувати запити та генерувати статистичні дані, які допомагають у прийнятті рішень, моніторингу ефективності та бізнес-аналітики. З допомогою баз даних можна інтегруватись з іншими системами та службами. Веб-системам часто потрібно взаємодіяти із зовнішніми API, програмами сторонніх розробників або іншими базами даних. Інтеграція з базою даних забезпечує безперебійний обмін даними та взаємодію.Узгодженість даних: База даних надає такі механізми, як транзакції та обмеження, для підтримки узгодженості та цілісності даних. Це гарантує, що дані залишаються точними та дійсними, навіть якщо кілька користувачів або процесів отримують до них доступ і змінюють їх одночасно.

Загалом база даних є критично важливим компонентом веб-системи, що забезпечує зберігання даних, генерацію динамічного вмісту, керування

користувачами, збереження даних, масштабованість, аналітику, інтеграцію та узгодженість даних. Він відіграє важливу роль у функціональності, продуктивності та надійності веб-систем.

Існує кілька типів баз даних, кожна з яких призначена для певних цілей і задовольняє різні вимоги до зберігання та пошуку даних. Ось деякі типові бази даних, які зазвичай використовуються:

1. Система управління реляційними базами даних (RDBMS): реляційні бази даних зберігають дані в табличній формі, де дані організовані в таблиці з рядками та стовпцями. Вони використовують SQL (структуровану мову запитів) для обробки та пошуку даних. Прикладами популярних реляційних баз даних є MySQL, Oracle Database, Microsoft SQL Server і PostgreSQL.

2. Бази даних NoSQL. Бази даних NoSQL — це нереляційні бази даних, які забезпечують гнучкі моделі даних і горизонтальну масштабованість. Вони призначені для роботи з великими обсягами неструктурованих і напівструктурованих даних. Різні типи баз даних NoSQL включають:

a. Бази даних документів: ці бази даних зберігають і отримують дані у формі документів, як правило, у форматах JSON або XML. Приклади включають MongoDB і CouchDB.

b. Зберігання ключ-значення: бази даних ключ-значення зберігають дані як набір пар ключ-значення, де кожен ключ унікальний і пов’язаний зі значенням. Приклади включають Redis і Riak.

в. Зберігання сімейства стовпців: бази даних сімейства стовпців організовують дані в сімействах стовпців або сімействах стовпців у рядку. Кожне сімейство стовпців може мати різний набір стовпців. Приклади включають Apache Cassandra і HBase.

d. Графові бази даних: Графові бази даних зосереджені на зв’язках між об’єктами та зберігають дані у графоподібній структурі вузлів і ребер. Вони підходять для керування даними з високим ступенем зв’язку. Приклади включають Neo4j і Amazon Neptune.

3. Бази даних NewSQL: Бази даних NewSQL мають на меті поєднати переваги традиційних баз даних SQL із масштабованістю баз даних NoSQL. Вони забезпечують гарантії ACID (атомність, узгодженість, ізоляція, довговічність), одночасно пропонуючи покращену продуктивність і масштабованість. Приклади включають CockroachDB і Google Spanner.

4. Бази даних часових рядів. Бази даних часових рядів оптимізовані для зберігання та отримання даних із мітками часу або часових рядів, таких як дані датчиків, журнали, фінансові дані та дані IoT. Вони забезпечують ефективне зберігання, пошук і аналіз даних часових рядів. Приклади включають InfluxDB і Prometheus.

5. Бази даних у пам’яті: бази даних у пам’яті зберігають дані переважно в системній пам’яті (RAM) замість традиційного дискового сховища. Вони пропонують швидкий доступ до даних і швидкість обробки, але можуть мати обмежену довговічність даних. Приклади включають Redis (який також підпадає під категорію зберігання ключ-значення) і Memcached.

6. Просторові бази даних: просторові бази даних призначені для зберігання та запиту просторових або географічних даних, таких як карти, GPS-координати та просторові зв’язки. Вони забезпечують спеціалізовані функції та структури індексування для ефективних просторових запитів. Приклади включають PostGIS і Oracle Spatial.

Серед перелічених видів баз даних мій вибір зупинився на реляційних базах даних через табличну структуру, де дані організовані в таблиці з попередньо визначеними стовпцями та типами даних. Цей структурований підхід забезпечує цілісність і узгодженість даних, полегшуючи зберігання, запити та аналіз даних. Вони надають властивості ACID (атомність, узгодженість, ізоляція, стійкість) для транзакцій. Відповідність ACID гарантує, що транзакції бази даних є надійними, послідовними та зберігають цілісність даних. Це гарантує, що зміни в базі даних відбуваються контрольованим і передбачуваним чином. Також важливим фактором є SQL (Structured Query Language) — це стандартизована мова, яка

використовується для взаємодії з реляційними базами даних. Вона надає багатий набір операцій і потужних можливостей запитів, що дозволяє здійснювати пошук складних даних, маніпулювати ними, агрегувати та аналізувати. Декларативний характер SQL дозволяє відносно легко писати та розуміти запити. Реляційні бази даних мають добре налагоджену екосистему з широким набором інструментів, фреймворків і бібліотек, які підтримують їх розробку, адміністрування та інтеграцію. Це включає надійні оптимізатори запитів, механізми резервного копіювання та відновлення, а також велику спільноту для підтримки та обміну знаннями.

Серед найпопулярніших SQL баз даних існує такі дві: PostgreSQL і MySQL

MySQL - це відкрита система керування базами даних, що розроблена на основі mSQL. В даний час Oracle володіє цим продуктом. MySQL може бути використана на різних платформах, таких як Mac, Windows, Linux і Unix. Основне використання MySQL - це розробка веб-застосунків.

MySQL може обробляти великі обсяги даних і підтримує різноманітні типи даних, включаючи цілі числа, числа з плаваючою комою, різні формати дат і виконавчі дані. Крім того, MySQL надає можливість використовувати рядкові типи даних змінної і фіксованої довжини.

Основні переваги MySQL:

1. Популярність: MySQL є однією з найбільш популярних систем керування базами даних. Це означає наявність великої спільноти розробників з великим досвідом використання цієї БД. Крім того, існує багато документації та навчальних матеріалів, які полегшують процес вивчення та розробки.

2. Безпечність: MySQL забезпечує повну безпеку вхідних даних користувачів шляхом обмеження доступу до даних. Це дозволяє забезпечити конфіденційність та цілісність інформації.

3. Швидкість: Незважаючи на те, що деякі можливості MySQL ще не повністю реалізовані, вона відома своєю високою продуктивністю. Це

означає, що MySQL здатна швидко обробляти та виконувати запити до бази даних.

Недоліки MySQL:

1. Взаємозалежність: MySQL зосереджена на швидкості та простоті, що призводить до обмежень у використанні складних можливостей SQL.
2. Платні функції: MySQL має дві ліцензії - відкриту та комерційну, і деякі функції доступні лише за плату.
3. Необхідність гарантій щодо розвитку: Оскільки Oracle придбала MySQL, існує неясність щодо швидкості та напрямку подальшого розвитку продукту.
4. Зниження продуктивності при великому обсязі даних: При збільшенні обсягу даних може спостерігатися зниження продуктивності, особливо при індексуванні та архівуванні великих таблиць.

PostgreSQL - це об'єктно-реляційна система управління базами даних, яка була створена як відкритий програмний проект у Каліфорнійському університеті в 1986 році. Спочатку розроблена для платформи UNIX, PostgreSQL була подальшою портована на різні поширені платформи. PostgreSQL відомий як проект з відкритим початковим кодом, доступний під ліцензією PostgreSQL, яка є загальнодоступною.

Особливості PostgreSQL включають можливість розширення стандартних функціональних можливостей за допомогою власних функцій, які можуть бути реалізовані в різних стилях програмування. Крім того, PostgreSQL дозволяє створювати власні типи даних, індекси та інші додаткові структури.

Переваги PostgreSQL:

1. Оптимізована обробка синхронних вимог та можливість блокування елементів.
2. Підтримка різних типів індексів, таких як btree, hash і багато інших.
3. Можливість розподілених таблиць.

4. Гнучкість у виборі методів реплікації, включаючи синхронну, асинхронну, логічну та фізичну реплікацію.
5. Підтримка SSL для забезпечення безпеки передачі даних.
6. Можливість створення власних типів даних, додаткових видів відомостей, що розширюють можливості PostgreSQL.
7. Наявність драйверів для багатьох платформ та стилів програмування.
8. Підтримка повнотекстового пошуку.
9. Проект з відкритим початковим кодом, підтримуваний активною спільнотою, що забезпечує розвиток та документацію.

Недоліки PostgreSQL:

1. Запізніла популярність: PostgreSQL відстає від MySQL у плані популярності, що обумовлено відсутністю активного комерційного просування та недостатньою кількістю сторонніх інструментів та експертів з адміністрування цієї бази даних.
2. Обмежене екосистема: В порівнянні з іншими системами управління базами даних, PostgreSQL має меншу кількість готових рішень, плагінів та розширень, що може обмежити гнучкість та можливості розробки.
3. Вимоги до ресурсів: PostgreSQL може вимагати більше обчислювальних ресурсів та пам'яті порівняно з іншими СУБД, особливо при роботі з великими обсягами даних.

Беручи до уваги перераховані вище недоліки для використання у кваліфікаційній роботі бакалавра було обрано СУБД PostgreSQL через свої функціональні можливості, а також через свою цілковиту відкритість.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ОРГАНАЙЗЕРА НА ОСНОВІ ВЕБ СИСТЕМИ

3.1 Постановка задачі. Технічне завдання

Після вивчення теоретичних основ проектування органайзеру на основі веб системи та виявлення основних принципів, які варто дотримуватися, а також недоліків, які потрібно уникати, основним завданням цієї роботи є:

розробка прототипу органайзеру, що базується на веб системі. Сервіс буде побудовано використовуючи клієнт-серверну архітектуру.

Для виконання поставлених задач потрібно покроково вирішити наступні завдання:

- визначити організацію потоків і зберігання даних у базі даних.
- розробити серверну частину, яка буде записувати, читати, трансформувати та видаляти дані з бази даних і комунікуватиме з клієнтською частиною.
- розробити клієнтську частину, яка буде робити запити до серверу і відображати дані для кінцевого користувача.
- забезпечити принцип захищеності даних користувачів

При побудові системи також потрібно дотриматися основних вимог та рекомендацій, наведених в попередніх розділах роботи, щоб забезпечити максимально ефективний та зручний органайзер на основі веб системи.

3.2 Архітектурне і програмне рішення веб системи

3.2.1 Визначення організації потоків і зберігання даних у базі даних

Для роботи з даними в цьому проекті був вибраний стек технологій, що базується на використанні реляційної бази даних, який був описаний у другому розділі цієї роботи. Відповідно до реляційних принципів роботи з даними, база даних містить заздалегідь визначені зв'язки між її елементами. Елементи представлені у вигляді таблиць зі стовпцями та рядками. Таблиці відображають об'єкти, які використовуються в системі. Стовпці таблиці мають певний тип даних, а кожне поле містить фактичне значення атрибута. Рядки в таблиці представляють набір пов'язаних значень для одного об'єкта

або сутності. Кожен рядок таблиці має унікальний ідентифікатор (первинний ключ, PK), а рядки між різними таблицями пов'язані за допомогою зовнішніх ключів (зовнішній ключ, FK). Доступ до цих даних здійснюється за допомогою запитів на мові SQL (Structured Query Language - Структурована мова запитів), при цьому не потрібно реорганізовувати самі таблиці бази даних.

На рис. 3.1 наведено схему бази даних органайзера на основі веб системи.

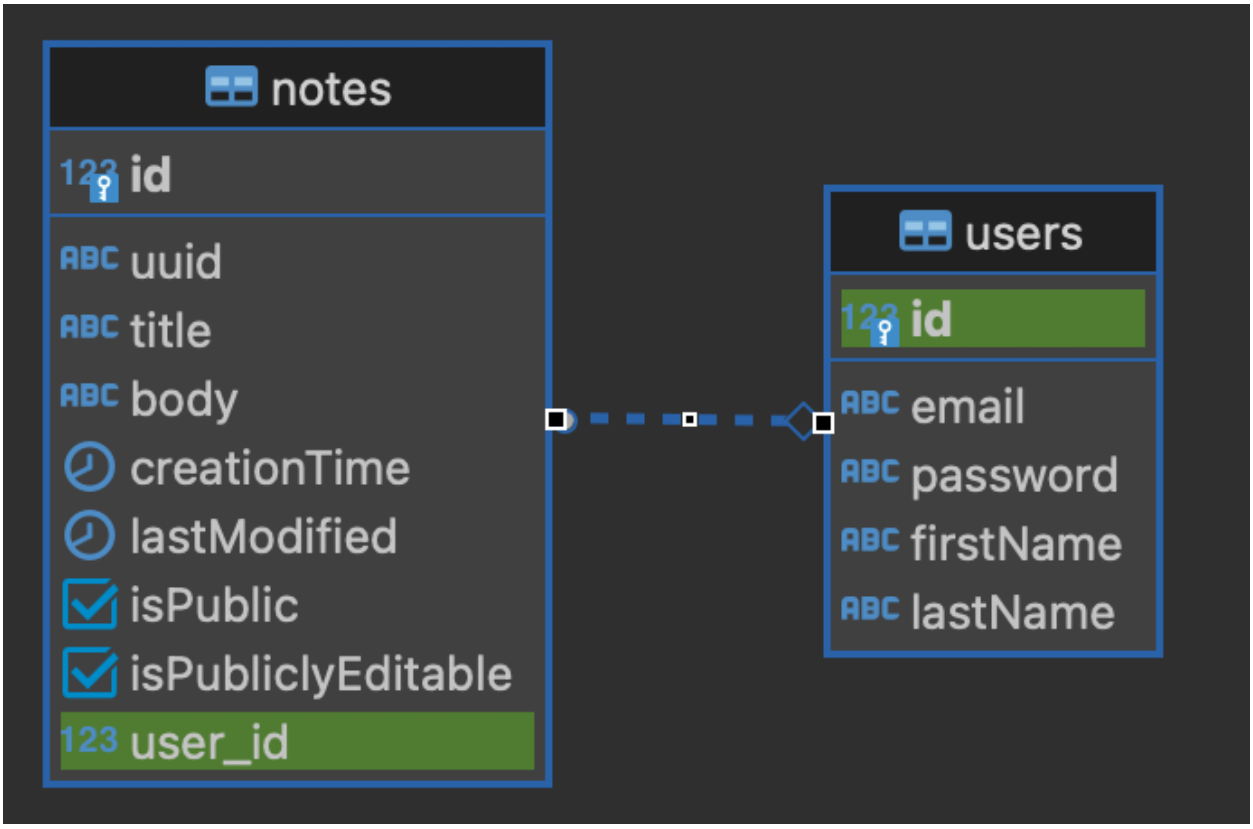

Рисунок 3.1 - Схема бази даних системи

Таблиця users містить у собі інформацію про користувачів системи.

- id - первинний ключ
- email - зберігає емейл користувача, має бути унікальним по всій табличці
- password - поле паролю захешованого із додаванням “солі”
- firstName - поле відображає ім’я користувача
- lastName - поле відображає прізвище користувача

Таблиця notes містить у собі інформацію по всіх нотатках системи.

- id - первинний ключ
- uuid - унікальний ідентифікатор нотатки, використовується для того, щоб при публічному доступі до нотатки не показувався індекс у базі даних адже з нього можна отримати інформацію скільки приблизно було створено нотаток до цього.
- title - поле, що відображає заголовок нотатки
- body - поле, що відображає тіло нотатки
- creationTime - відображає час, коли було створено нотатку
- lastModified - відображає час, коли було останній раз модифіковано нотатку
- isPublic - показує чи є нотатка публічною
- isPubliclyEditable - показує чи є нотатка публічно редагованою
- user_id - використовується для позначення якому користувачу належить дана нотатка

Зі схеми бази даних випливає, що у системі буде дві основні сутності - користувач і нотатка, яка йому належить.

3.2.2 Розробка серверної частини

Як було згадано у другому розділі, мій вибір зупинився на фреймворку FastAPI - це сучасна, високопродуктивний веб-фреймворк для створення API за допомогою Python. Він створений як швидкий, ефективний і простий у використанні. FastAPI використовує потужність підказок типів у Python і забезпечує автоматичну перевірку запитів і відповідей, серіалізацію та створення документації. Він створений на основі Starlette, асинхронної веб-платформи, і використовує асинхронні можливості Python для ефективної обробки високих навантажень і паралельного виконання. FastAPI підтримує різні функції, такі як маршрутизація, впровадження залежностей, безпека та автентифікація, перевірка даних, фонові завдання тощо. Він відомий своєю чудовою продуктивністю та функціями, зручними для розробників, що робить його популярним вибором для створення надійних і масштабованих API.

Для вищезазначених сутностей було розроблено API ендпоінти, які надають можливість створення, читання, оновлення і видалення сутностей.

Рисунок 3.2 - Ендпоїнти розробленого API

Ендпоїнти можна виділити у дві групи: перша - користувачі, друга - нотатки.

До групи користувачі відносяться такі ендпоїнти:

- Create User - відповідає за створення користувача
- Login User - відповідає за вхід вже створеного користувача
- До групи нотатків відносять наступні ендпоїнти:
- Create User Note - створює нову нотатку для користувача
- Get User Note - повертає інформацію про нотатку користувача
- Update Note - оновлює вміст нотатки
- Delete Note - видаляє нотатку
- Get User Notes - повертає всі нотатки користувача
- Get Public Note - повертає інформацію про публічну нотатку
- Change Note Public Status - змінює публічний статус нотатки
- Change Note Publicly Editable Status - змінює можливість редагування публічної нотатки

Для взаємодії сервера із базою даних використовується ORM SQLAlchemy. Вона надає функціонал для створення запитів до бази даних

без прямої побудови SQL запиту. Такий підхід підвищує захищеність програми і запобігає sql ін’єкціям.

Для взаємодії із базою даних SQLAlchemy використовує проміжні моделі (рис. 3.3) для подальшої побудови запиту

```python
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    email = Column(String, nullable=False, unique=True)
    password = Column(String, nullable=False)
    firstName = Column(String, nullable=True)
    lastName = Column(String, nullable=True)

    notes = relationship("Note", backref="user")

    def dict(self):
        return { c.key: getattr(self, c.key) for c in inspect(self).mapper.column_attrs }


class Note(Base):
    __tablename__ = 'notes'

    id = Column(Integer, primary_key=True)
    uuid = Column(String, nullable=False, unique=True)
    user_id = Column(Integer, ForeignKey('users.id', ondelete="CASCADE", onupdate="CASCADE"))
    title = Column(String, nullable=False)
    body = Column(UnicodeText, nullable=True)
    creationTime = Column(DateTime, nullable=False)
    lastModified = Column(DateTime, nullable=False)
    isPublic = Column(Boolean, nullable=False, default=False)
    isPubliclyEditable = Column(Boolean, nullable=False, default=False)

    def dict(self):
        return { c.key: getattr(self, c.key) for c in inspect(self).mapper.column_attrs }
```

Рисунок 3.3 - Моделі SQLAlchemy

3.2.3 Розробка клієнтської частини

Для розробки клієнтської частини було вибрано фронтенд фреймворк - Angular. Він використовується для створення динамічних односторінкових програм (SPA) і надає повний набір інструментів і бібліотек для зовнішньої розробки. Angular дотримується компонентної архітектури, де застосунок розділено на багаторазово використовувані компоненти, які інкапсулюють логіку HTML, CSS і TypeScript.

Під час розробки було виділено такі компоненти (рис. 3.4)

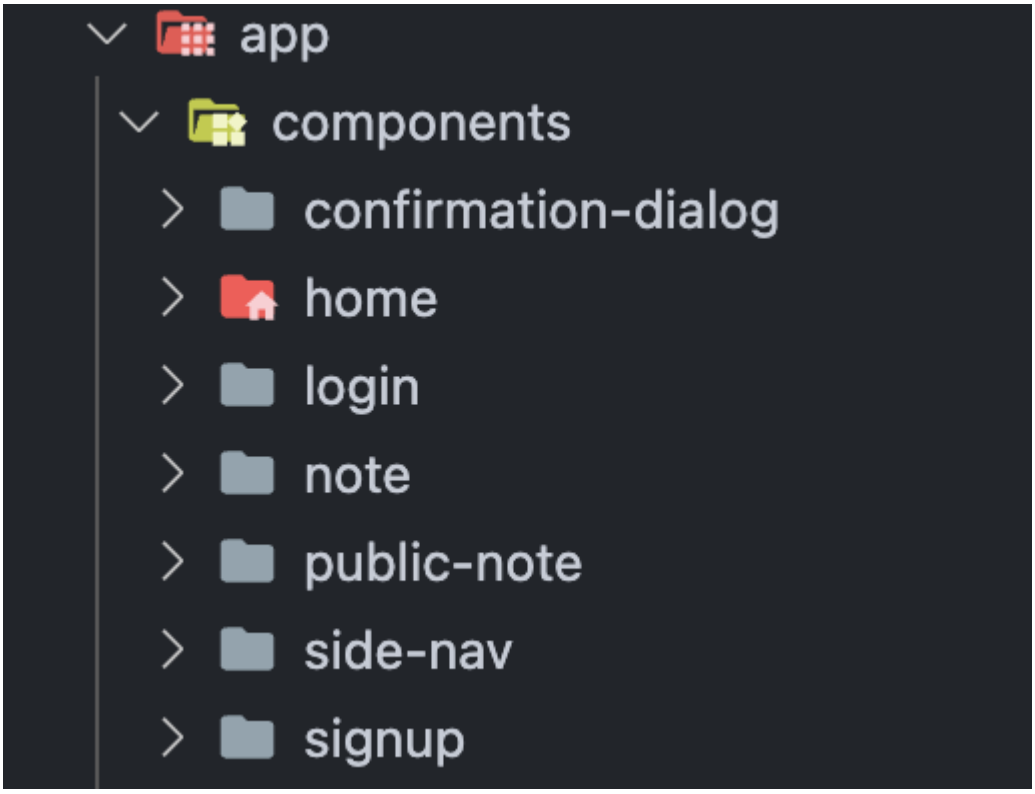

Рисунок 3.4 - Компоненти створені на клієнтській частині

confirmation-dialog - компонент, що відображає діалогове вікно підтвердження

home - домашній (головний) компонент, який бачать незареєстровані користувачі

login - компонент для входу в систему

note - компонент, що відображає нотатку

public-note - компонент, що відображає публічну нотатку

side-nav - компонент, що відображає бокове меню

signup - компонент, що відображає форму реєстрації

Окрім компонентів, Angular має поняття сервісів, які можуть слугувати як збереження стану застосунку або виконання певних функції, таких як звернення до зовнішнього API, яким у даному випадку виступає сервер

Приклад запиту із клієнта, на сервер (рис. 3.5)

```
login(formData: any): Observable<HttpResponse<any>> {
  return this.http.post<any>(this.apiUrl + "user/login/", formData, { observe: 'response' })
    .pipe(
      tap((resp: HttpResponse<any>) => {
        return resp;
      }),
      catchError(this.errorHandler)
    );
}
```

Рисунок 3.5 - Приклад запиту зі сторони клієнту

Було розроблено наступні сторінки

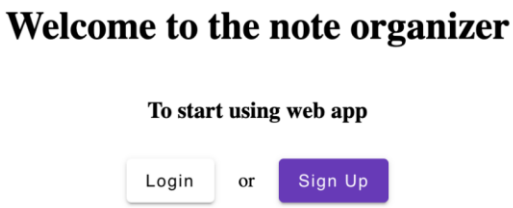

Рисунок 3.6 - Домашня сторінка

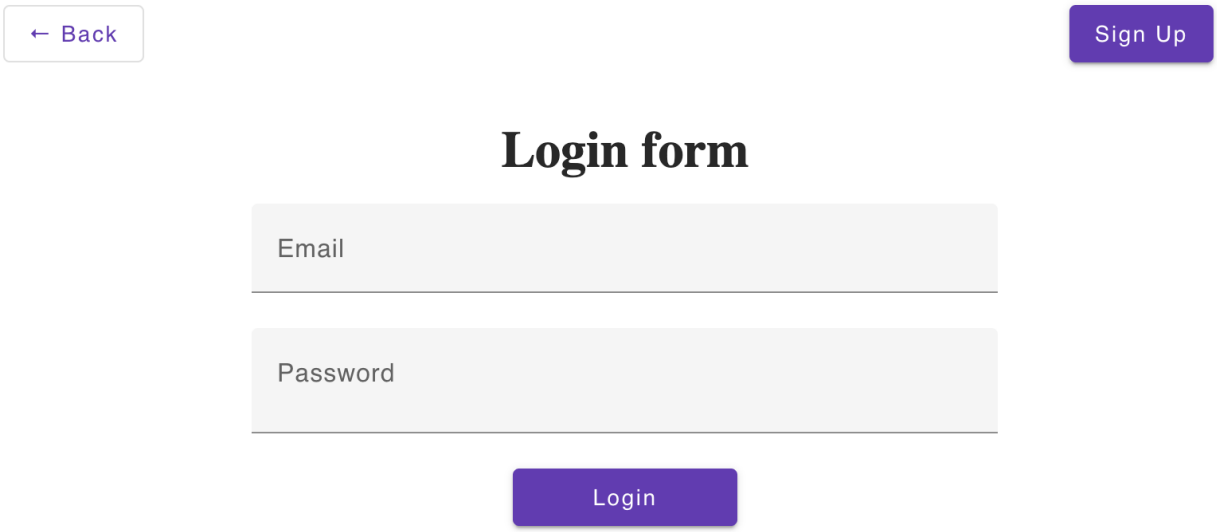

Рисунок 3.7 - Форма входу

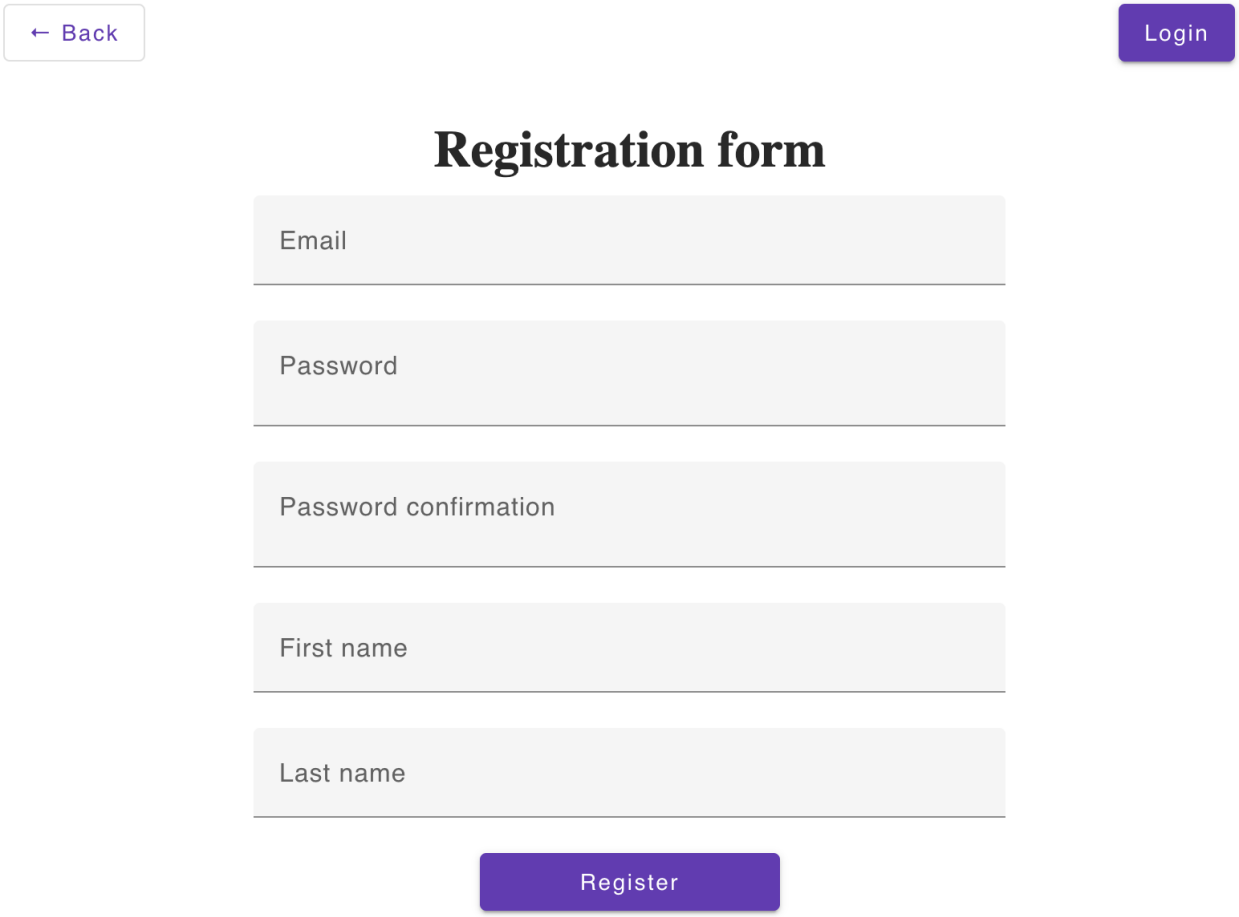

Рисунок 3.8 - форма реєстрації

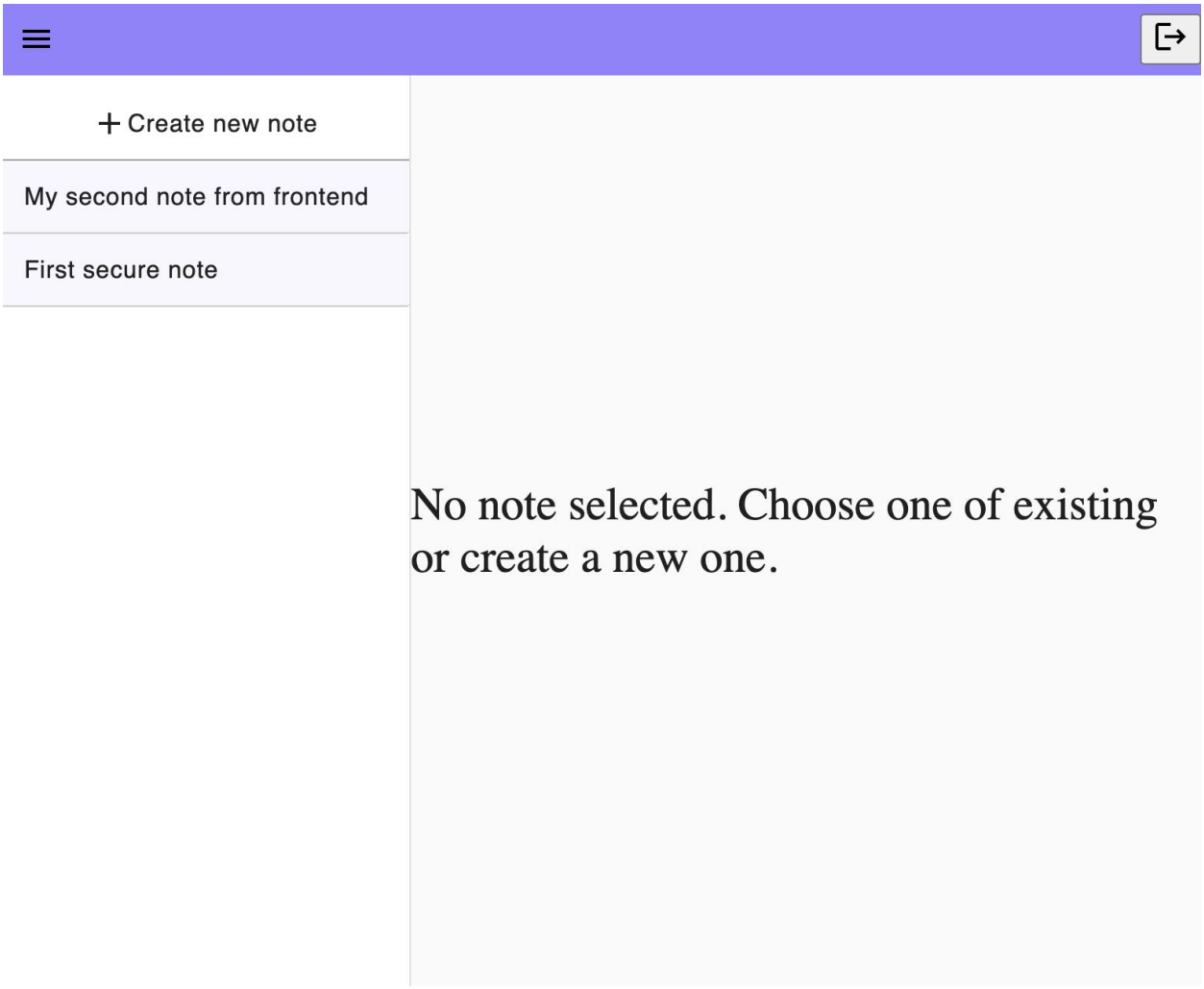

Рисунок 3.9 - Екран зареєстрованого користувача

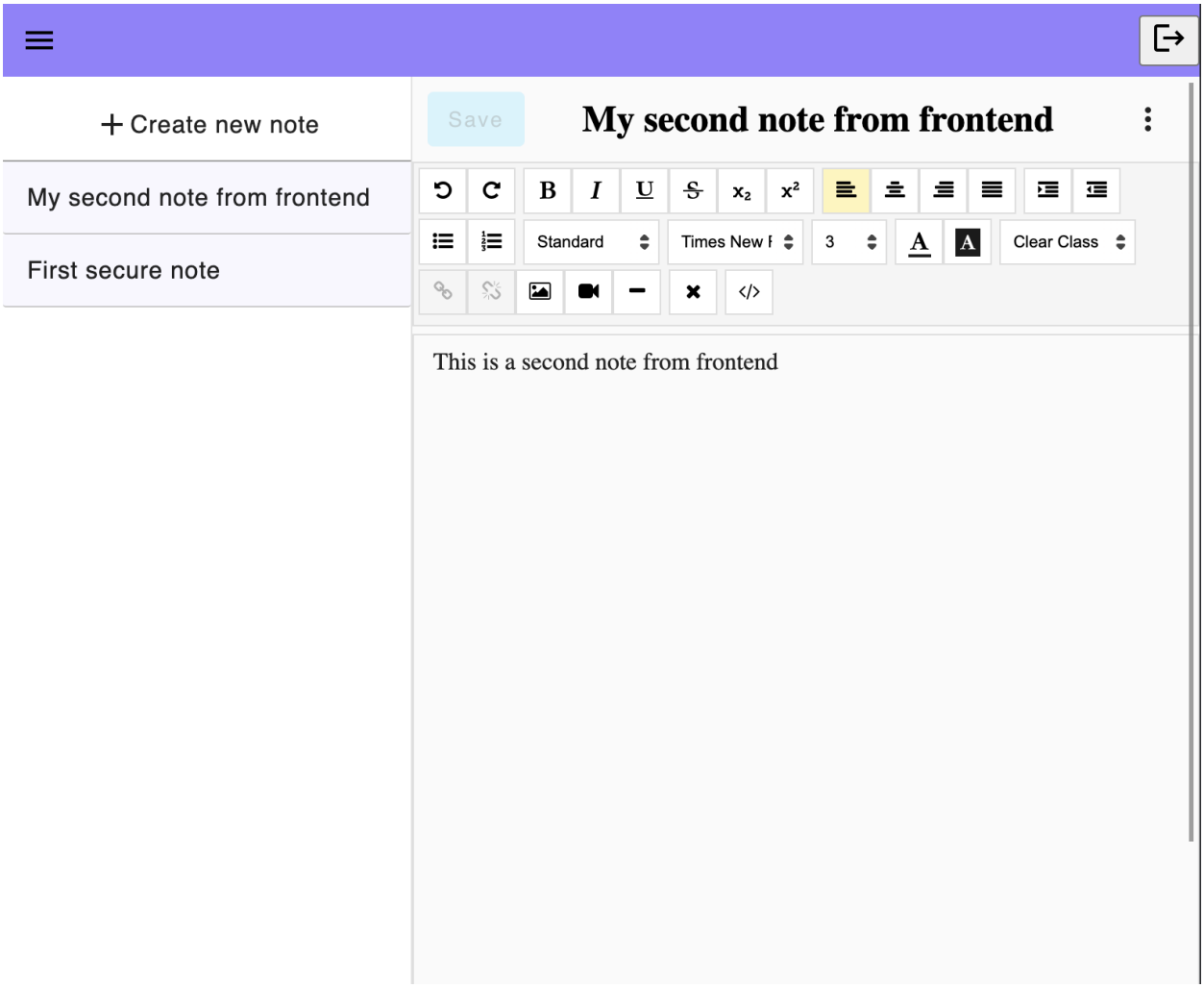

Рисунок 3.10 - Вікно вибраної нотатки

3.2.4 Забезпечення принципів захищеності даних користувачів

Для захисту даних користувачів було впроваджено автентифікацію, що базується на JWT токені - популярний метод безпечної передачі інформації між сторонами як об’єкт JSON. Він складається з трьох частин: заголовка, корисного навантаження та підпису.

Корисне навантаження JWT містить претензії, які є заявами про сутність (як правило, користувача) і додаткові метадані. Претензії можна класифікувати на три типи: зарезервовані претензії (попередньо визначені стандартом JWT), публічні претензії (використовуються для обміну інформацією) і приватні претензії (спеціальні претензії, специфічні для програм).

Щоб забезпечити цілісність і автентичність JWT, він підписується за допомогою секретного ключа або пари закритий/відкритий ключ. Підпис генерується шляхом застосування криптографічного алгоритму (наприклад, HMAC або RSA) до закодованого заголовка, корисного навантаження та секрету/ключа.

Одержувач JWT може перевірити його автентичність, підтвердивши підпис за допомогою відповідного відкритого або секретного ключа. Якщо підпис дійсний, це означає, що JWT не змінено.

JWT зазвичай використовуються для цілей автентифікації та авторизації. Після успішного входу користувача JWT може бути видано та надіслано клієнту. Подальші запити до захищених ресурсів можуть включати JWT, що дозволяє серверу автентифікувати та авторизувати користувача на основі інформації в токені.

Ще одним важливим фактором безпеки застосунку є хешування паролів користувачів. Розроблений органайзер на основі веб системи використовує Python бібліотеку bcrypt — це бібліотека, яка забезпечує безпечний спосіб хешування паролів за допомогою алгоритму хешування bcrypt. Bcrypt — це широко використовувана функція хешування паролів, яка включає в себе сіль і фактор вартості для підвищення безпеки.

Алгоритм bcrypt розроблений як повільний і дорогий за обчислювальними ресурсами, що робить його стійким до атак грубої сили. Він спеціальним чином використовує шифр шифрування Blowfish для створення хешованого представлення пароля. Bcrypt також автоматично генерує унікальну сіль для кожного пароля, додаючи додатковий рівень безпеки.

3.3 Інструкція користувача

Після заходу на сайт, користувачу буде запропоновано увійти або зареєструватися (рис. 3.6)

У формі реєстрації (рис. 3.8) потрібно вказати такі обов’язкові поля як електронну адресу, пароль, підтвердження пароля. Також можна вказати ім’я та прізвище.

Форма входу майже аналогічна, за винятком, що з полів буде доступно тільки ввід електронної адреси і пароля.

Після входу або реєстрації користувача буде автоматично перенаправлено на сторінку з його нотатками (рис. 3.9), де він зліва бачить список власних нотаток, може додати нову нотатку, видалити вибрану нотатку або зробити публічною.

Найбільшою перевагою даного органайзера є те, що він підтримує збереження і редагування нотатків як HTML сторінок.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було здійснено реалізацію органайзера на основі веб системи. Перед початком роботи було розглянуто підходи до розроблення органайзеру на основі веб системи, проаналізовано архітектурні рішення і зроблено вибір програмних засобів для реалізації органайзера на основі веб системи.

Було спроектовано та розроблено серверну частину веб платформи на базі фреймворку FastAPI, бібліотеки SQLAlchemy в якості об’єктно-реляційної проекції, бібліотеки bcrypt для реалізації хешування паролів користувачів. За допомогою фреймворку Angular було спроектовано клієнтську частину платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PostgreSQL: Documentation: веб-сайт. URL: https://www.postgresql.org/docs/ (дата звернення: 05.12.2022)
2. MySQL Documentation: веб-сайт. URL: https://dev.mysql.com/doc/ (дата звернення: 06.12.2022)
3. Angular (фреймворк) - Wikipedia: веб-сайт. URL: https://uk.wikipedia.org/wiki/Angular_(фреймворк) (дата звернення: 01.02.2023)
4. React Wikipedia: веб-сайт. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 04.02.2023)
5. Our documentation | python.org: веб-сайт. URL: https://www.python.org/doc/ (дата звернення: 15.02.2023)
6. What is software development? | IBM: веб-сайт. URL: https://www.ibm.com/topics/software-development (дата звернення: 05.02.2023)
7. Build Your Software Architecture Right. Here's How: веб-сайт. URL: https://builtin.com/software-engineering-perspectives/software-architecture (дата звернення: 15.02.2023)
8. How to Sign and Validate JSON Web Tokens - JWT Tutorial: веб-сайт. URL: https://www.freecodecamp.org/news/how-to-sign-and-validate-json-web-tokens/ (дата звернення: 25.02.2023)
9. FastAPI: веб-сайт. URL: https://fastapi.tiangolo.com/ (дата звернення: 06.01.2023)
10. SQLAlchemy: веб-сайт. URL: https://www.sqlalchemy.org/ (дата звернення: 08.01.2023)
11. bcrypt - Wikipedia: веб-сайт. URL: https://en.wikipedia.org/wiki/Bcrypt (дата звернення: 10.01.2023)