МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет комп‘ютерних наук та технологій
Кафедра Комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **«МАГІСТР»**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Back-end соціальної мережі нового покоління *«Dvij»*»

Виконавець: студент групи УС-212М Карпенко Данило Олегович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер:_____ Ігор РАЙЧЕВ

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Карпенка Данила Олеговича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Back-end соціальної мережі нового покоління *«Dvij»*» затверджена наказом ректора від «29» вересня 2023 р. за №1976/ст.

2. Термін виконання роботи: 02.10.2023 – 31.12.2023р.

3. Вихідні дані до роботи: Back-end соціальної мережі нового покоління *«Dvij».*

4. Зміст пояснювальної записки: вступ, опис соціальної мережі нового покоління "DVIJ", платформа програмування Node.js, призначення та особливості, основні інструменти та компоненти, середовище програмування Visual Studio, опис та переваги використання середовища, розробка додатку, архітектура серверної частини додатку, Nest.js фреймворк, структура бази даних проекту.

5. Перелік обов'язкового ілюстративного матеріалу: головний екран соціальної мережі, популярність AWS і Heroku, популярність PostgreSQL і MySQL.

6. Календарний план-графік

№ п/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Проаналізувати літературу та джерела за темою дипломної роботи	02.10.23 – 08.10.23р.	
2.	Розроблення та затвердження плану дипломної роботи	09.10.23 – 11.10.23р.	
3.	Привести консультації з науковим керівником щодо створення першого розділу	12.10.23 – 16.10.23р.	
4.	Розробка розділу 1	17.10.23 – 28.10.23р.	
5.	Розробка розділу 2	29.10.23 – 19.11.23р.	
6.	Розробка розділу 3	20.11.23 – 01.121.23р.	
8.	Висновки та оформлення пояснювальної записки дипломної роботи	02.12.23 – 13.12.23р.	
9.	Підписання необхідних документів у встановленому порядку	14.12.22 –19.12.23р.	
10.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	20.12.23 – 24.12.23р.	

7. Дата видачі завдання: «02» жовтня 2023 р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Данило КАРПЕНКО

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Back-end соціальної мережі нового покоління *«Dvij»*» містить 93 сторінки, 8 рисунків, 16 бібліографічних посилань.

Об'єктом дослідження **є**: соціальна мережа нового покоління з використанням штучного інтелекту для генерації зображень.

Предметом дослідження є: Back-end соціальної мережі нового покоління з імплементацією штучного інтелекту на основі сервісу Midjourney для генерації.

Мета роботи є: розробка та дослідження соціальної платформи "Dvij"

Для досягнення поставленої мети необхідно виконати **наступні завдання**:

✔ опис соціальної мережі нового покоління “dvij”;

✔ розробити архітектуру, структуру та особливості реалізації програмної системи;

✔ провести аналіз показників та характеристик отриманої програмної системи.

Методи дослідження – аналіз, порівняння та дослідження інформації потреб сучасного соціуму та реалізація актуального функціоналу в соціальній мережі та аналіз, порівняння та дослідження сучасних технологій для реалізації функціоналу соціальної мережі нового покоління.

Ключові слова: СОЦІАЛЬНА МЕРЕЖА, BACK-END, AMAZON WEB SERVICES, NODE.JS, HEROKU.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API — Прикладний програмний інтерфейс

AWS — Amazon Web Services;

HTTP — протокол передачі гіпер-текстових документів;

MVP — Мінімально життєздатний продукт

NFT — Non-Fungible Token

USP — Унікальна Пропозиція Продажу

ВСТУП

Сучасний світ переживає період стрімкого розвитку інформаційних технологій і глибокого змішання віртуальної та реальної реальності. Соціальні мережі вже стали невід’ємною частиною нашого повсякденного життя, проте існує потреба в інноваціях у цій галузі, які б забезпечили більш глибоку, змістовну та взаємодопоміжну спільноту.

Сучасні соціальні мережі нерідко переважно спрямовані на рекламу та масове споживання контенту. Завдяки проекту "Dvij", ми пропонуємо новий погляд на соціальну мережу, де спільнота об'єднується навколо спільних інтересів та захоплень, створюючи унікальний віртуальний світ для спільного дослідження та спілкування.

Актуальність проекту "Dvij" полягає в наступному:

1) Зростаюча потреба в особистій інтеракції: У сучасному світі, де велика кількість спілкування відбувається в онлайні, люди все більше відчувають потребу в більш особистому та значущому взаємодії.

2) Гейміфікація як ефективний інструмент: Використання гейміфікації може значно покращити досвід користувачів і стимулювати їх взаємодію з платформою, роблячи її більш захопливою та забавною.

3) Зростаюча кількість онлайн-спільнот: Потреба у формуванні глибоких онлайн-спільнот, які об'єднують людей зі схожими інтересами, стає все більш актуальною.

4) Інновації в сфері штучного інтелекту та NFT: Використання штучного інтелекту для персоналізації та покращення досвіду користувачів, а також інтеграція NFT як форми внутрішньої валюти, може зробити платформу більш ефективною та привабливою.

У контексті цих викликів та потреб, проект "Dvij" має великий потенціал забезпечити новий рівень спільноти та співпраці, який відповідає сучасним вимогам користувачів.

Метою даної дипломної роботи є розробка та дослідження соціальної платформи "Dvij", спрямованої на створення унікального віртуального середовища для спільнот, що об'єднуються навколо спільних інтересів та захоплень користувачів. Робота передбачає аналіз поточних тенденцій у сфері соціальних мереж, впровадження механік гейміфікації, використання штучного інтелекту та інтеграцію NFT як внутрішньої валюти для створення інноваційної та привабливої платформи для спільнот та взаємодії користувачів.

Ключовими аспектами досягнення цієї мети є розробка функціоналу "Dvij", що відповідає потребам користувачів, та проведення ефективного тестування та оцінки досвіду користувачів для підтвердження ефективності та цінності платформи.

Об'єкт – соціальна мережа нового покоління з використанням штучного інтелекту для генерації зображень.

Предмет дослідження – Back-end соціальної мережі нового покоління з імплементацією штучного інтелекту на основі сервісу Midjourney для генерації.

Методи дослідження – аналіз, порівняння та дослідження інформації потреб сучасного соціуму та реалізація актуального функціоналу в соціальній мережі. Аналіз, порівняння та дослідження сучасних технологій для реалізації функціоналу соціальної мережі нового покоління.

Наукова новизна отриманих результатів – соціальна мережа нового покоління, яка надає актуальні рішення та можливості для сучасного світу. Використання штучного інтелекту для генерації зображень користувача на основі його дій в соціальній мережі. Унікальний інтерфейс та можливості соціальної мережі.

Практичне значення отриманих результатів – створення соціальної мережі. Зручна соціальна мережа для сучасного соціуму, якою легко та вигідно користуватися. Покращує взаємодію, комунікацію, довіру та лояльність між людьми.

РОЗДІЛ 1

ОПИС СОЦІАЛЬНОЇ МЕРЕЖІ НОВОГО ПОКОЛІННЯ “DVIJ”

1.1. Призначення соціальної мережі

Основне призначення Dvij полягає в наданні зручного та доступного функціоналу для спілкування та взаємодії з друзями та однодумцями з будь-якої тематики - IT, готування, танці, подорожі, настільні ігри, аніме, велосипеди, бізнес і, загалом, все, що завгодно. Інтерактивна карта світу улюблених речей і всього, що з ними пов'язано. Записна книжка + пошуковик саме того, що користувач прагне знайти. І жодної реклами.

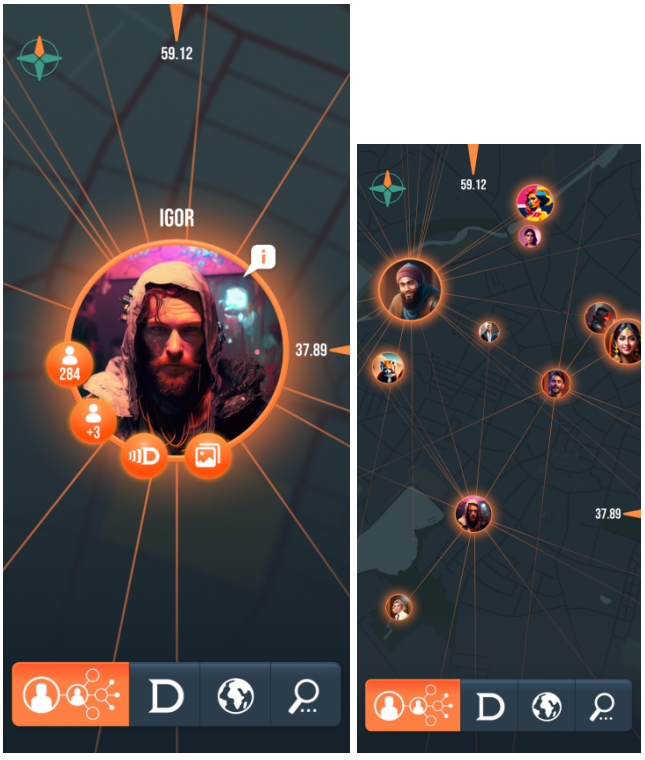


Рис.1.1. Головний екран соціальної мережі

1.2. Унікальність соціальної мережі

Dvij у кожному сегменті виглядає не так, як звик користувач інших сучасних соціальних мереж. Це віртуальна соціальна гра, яка дозволяє зручно управляти своєю взаємодією з людьми та культурами. Ми створюємо власний віртуальний світ з нашої мережі контактів та улюблених занять. І наповнюємо його зручним для себе контентом разом із іншими людьми - доступно API для додавання власного функціоналу.

1.3. Гейміфікація соціальної мережі

USP (Unique Selling Point) - практично будь-який процес в житті є грою і може бути легко гейміфікованим.2 Під час проектування та розробки Dvij ми використовуємо механіки роботи з іграми. Наприклад, ми розглянули всіх користувачів з точки зору сегментації за Бартлом і окремо визначили, які механіки будуть утримувати кожен тип користувача (гравців).

Типи гравців за Бартлом:

1) Соціальники - в першу чергу ми будемо дбати про цей тип користувачів. Вони найменш платять, але саме цей тип користувачів приводить найбільше друзів у вподобану їм програму.
2) Кар'єристи
3) Дослідники
4) Кілери

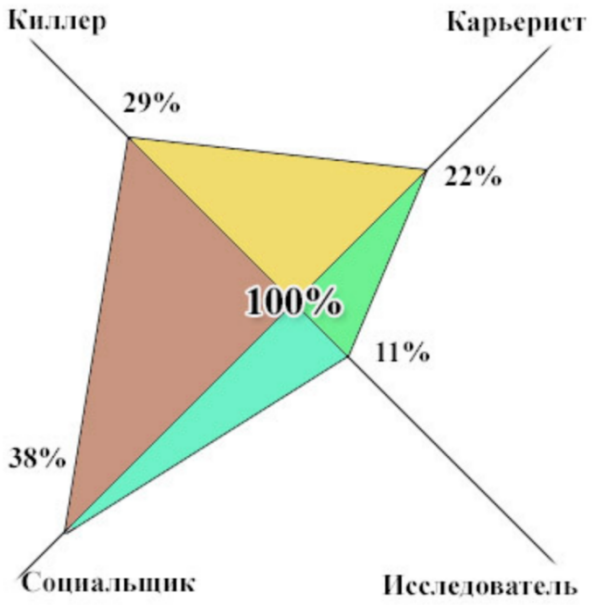


Рис.1.2. Розподіл користувачів по типам гравців

1.4. Основний функціонал для MVP

В основу MVP кладуться ключові інтереси кожного з сегментів, завдяки чому ми привертаємо максимально широку аудиторію.

Головний екран:

1) 3D карта з відображенням всіх користувачів соціальної мережі (доступний перегляд з виключаючими або вбудованими фільтрами, які будуть запускатися за замовчуванням при великій кількості користувачів)

2) Роза вітрів - відображає положення N (північ) і працює як шестерня налаштувань

3) Список активних фільтрів за користувачами та рухами

4) Нейроава користувача - аватар, оброблений в нейромережі MidJourney, при натисканні пальцем на аватарку будь-якої людини вона розсіюється і під нею видно його реальну фотографію, з якої було згенеровано геоаву (наприклад, щоб легко знайти його серед незнайомих людей у толпі)

5) Френдсхаб - число відображає кількість активних зв'язків користувача. При натисканні відкривається список друзів з різними сортуваннями та фільтрами:

5.1) Фільтр за участю в тегах рухів

5.2) Сортування за алфавітом, датою реєстрації в соцмережі, додаванням у друзі, останньою активністю (activities = dvij)

5.3) Пошук за ключовими словами (ім'я, прізвище, щось з опису і т. д.).

При натисканні на користувача ми переходимо на його профіль. Усі переходи між екранами зберігаються, для переходів назад по соціальній сходинці :)

6) Фотохаб - число відображає загальну кількість завантажених користувачем фотографій. При натисканні на цю іконку відкривається повноекранне модальне вікно з трьома вкладками:

6.1) Фотографії, зроблені цим користувачем

6.2) Фотографії, на яких є (відзначений або визначений алгоритмом) цей користувач

6.3) Список “двіжів”, в яких брав участь цей користувач з можливістю перейти до них та переглянути там всі загальнодоступні фотографії, зроблені під час цього руху.

1.5. Використанні технології для написання Back-end-у соціальної мережі нового покоління

Використані технології:

1) JavaScript, AWS, PostgreSQL - бекенд

2) Штучний інтелект - допомога в генерації контенту (покращення аватара, іконки для рухів)

3) NFT - внутрішня гроші на платформі Ethereum

4) Платформи: iOS, Android, visionOS.

1.5.1. Порівняння Node.js та Python

Розглянемо в порівнянні детальніше технології Node.js та Python для написання бекенду вашого проекту "Dvij" і пояснимо, чому саме Node.js краще підходить для цієї задачі порівняно з Python.

Node.js:

Node.js - це серверне середовище, побудоване на движку V8 від Google, яке дозволяє виконувати JavaScript на сервері. Він став популярним завдяки своїй ефективності та можливості створення швидких та масштабованих серверних додатків. Ось детальніше про Node.js:

Ефективність та події: Node.js є однопоточним та подійно-орієнтованим середовищем, що дозволяє обробляти багато одночасних запитів без блокування виконання. Це важливо для соціальних мереж з великою кількістю користувачів, де може бути багато одночасних взаємодій.

Спрощений веб-сервер: Node.js має спрощений вбудований веб-сервер, що полегшує розробку API та обробку запитів HTTP.

Екосистема та модульність: Node.js має велику екосистему бібліотек та модулів, які допомагають розробникам швидко створювати серверні додатки. Це також сприяє модульності і повторному використанню коду.

Спільнота та активний розвиток: Node.js має велику та активну спільноту розробників, що робить його стабільним та динамічно розвиваючимся середовищем.

Ось яким чином Node.js буде використований у проекті:

1) Серверна логіка: Node.js стане основним інструментом для створення серверної логіки вашого соціального мережевого додатку. Ви можете використовувати Node.js для обробки HTTP-запитів від клієнтів, реалізації бізнес-логіки, а також взаємодії з базою даних.

2) Маршрутизація: Ви можете використовувати Node.js для створення маршрутів, які визначатимуть, які ресурси та функції повинні бути доступні

клієнтам через HTTP-запити. Ви можете використовувати популярні бібліотеки, такі як Express.js, для спрощення роботи з маршрутами.

3) Керування базою даних: Node.js може бути використаний для взаємодії з вашою базою даних, яка, як ви зазначили, буде на основі PostgreSQL. Ви можете використовувати бібліотеки, такі як pg-promise або sequelize, для спрощення роботи з PostgreSQL у вашому Node.js додатку.

4) Асинхронність: Однією з головних переваг Node.js є його асинхронний характер. Ви можете одночасно обробляти багато запитів без блокування виконання інших завдань. Це особливо корисно в веб-додатках, які мають багато одночасних клієнтів.

5) Real-Time Communication: Node.js також добре підходить для реалізації реального часу в вашому соціальному мережевому додатку. Ви можете використовувати WebSocket або бібліотеки, такі як Socket.io, для встановлення з'єднань в режимі реального часу між сервером і клієнтами.

6) Пакетний менеджер: Node.js постачається з пакетним менеджером npm, який дозволяє легко встановлювати та керувати залежностями вашого проекту, такими як бібліотеки та фреймворки.

7) Інтеграція з іншими технологіями: Node.js може бути легко інтегровано з іншими технологіями, такими як бази даних, бібліотеки для роботи з блокчейном (як у вашому проекті з NFT), іншими сервісами AWS і т. д.

8) Масштабованість: Node.js позначається високою швидкістю та здатністю до масштабування, що робить його відмінним вибором для веб-додатків, які вимагають високої продуктивності та обробки багатьох одночасних запитів.

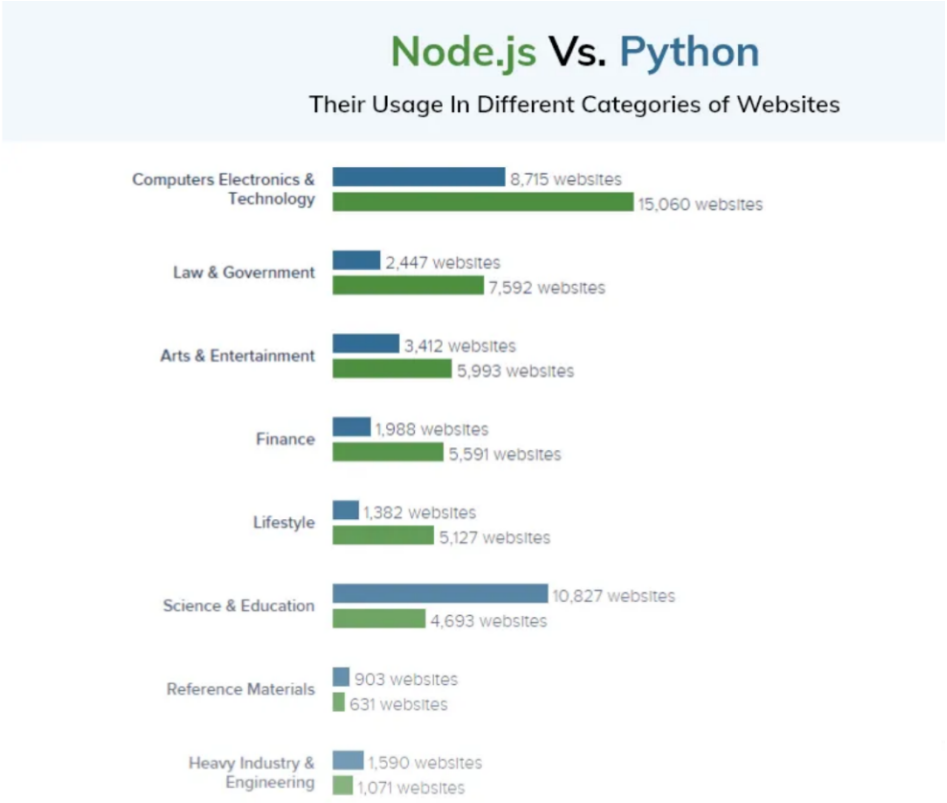


Рис.1.3. Різниця в використанні технологій по категоріям сайтів

Python:

Python - це високорівнева мова програмування, яка також може використовуватися для написання серверного програмного забезпечення, включаючи веб-додатки. Ось детальніше про Python:

Специфікація: Python відомий своєю простотою та читабельністю коду. Він підтримує багато фреймворків, таких як Django і Flask, для створення веб-додатків.

Використання: Python часто використовується для розробки веб-додатків, а також для наукових обчислень, штучного інтелекту та інших галузей.

Екосистема: Python має багатий вибір бібліотек і фреймворків для веб-розробки. Django і Flask, наприклад, є популярними фреймворками для створення веб-додатків.

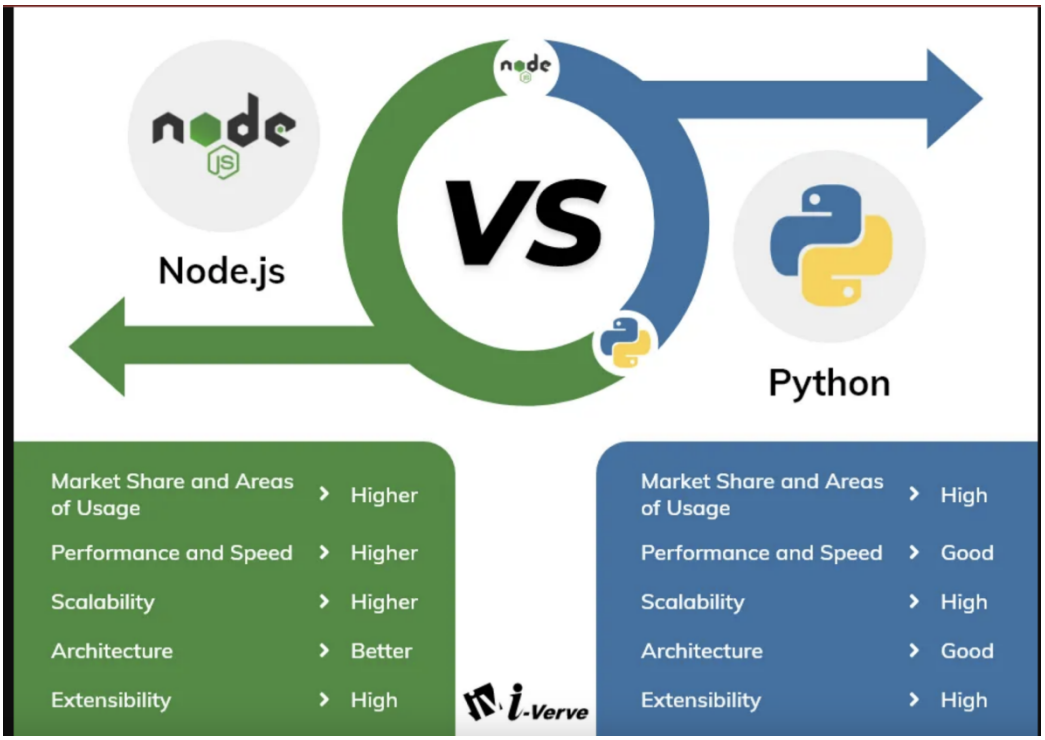


Рис.1.4. Порівняння NodeJs та Python

Чому Node.js краще підходить для написання бекенду вашого проекту "Dvij" порівняно з Python:

Ефективність і масштабованість: Node.js побудований на асинхронному і подійно-орієнтованому підході, що робить його дуже ефективним для обробки багатьох одночасних запитів. Це особливо важливо для соціальних мереж, де потрібно обслуговувати багато користувачів одночасно.

Єдність мови програмування: Використання JavaScript як мови програмування як на клієнтській, так і на серверній стороні може спростити розробку і зменшити затрати на навчання команди розробників.

Активна спільнота та розвиток: Node.js має активну та широку спільноту розробників, що гарантує підтримку та розвиток технології у майбутньому.

Отже, Node.js вважається кращим вибором для створення бекенду соціальної мережі "Dvij" завдяки його ефективності, масштабованості та можливості використовувати JavaScript як єдину мову програмування на обох сторонах додатку.

1.5.2. Порівняння технології AWS (Amazon Web Services), Node.js та Heroku

Розглянемо детальніше технології AWS (Amazon Web Services), Node.js та Heroku для написання бекенду вашого проекту "Dvij" і пояснимо, чому саме AWS може бути кращим вибором для цієї задачі порівняно з іншими платформами.

AWS (Amazon Web Services):

Amazon Web Services (AWS) - це хмарна платформа, яка надає набір послуг і ресурсів для хостингу, розгортання і керування додатками в інтернеті. AWS має багатий вибір сервісів, включаючи обчислювальні ресурси, бази даних, зберігання, аналітику та інше. Ось детальніше про AWS:

Сервіси і ресурси: AWS надає доступ до широкого спектра обчислювальних та інфраструктурних ресурсів, які можна використовувати для створення та розгортання серверних додатків.

Масштабованість: AWS дозволяє легко масштабувати ваші додатки, збільшуючи або зменшуючи обчислювальні ресурси за потребою.

Гнучкість: AWS дозволяє вибирати мову програмування та технології для розробки бекенду, включаючи Node.js, Python, Java і багато інших.

Сервіси які будуть використанні в розробці проекту:

1) Amazon EC2 (Elastic Compute Cloud): Amazon EC2 дозволить вам створити віртуальні сервери для розгортання та керування вашим бекендом і фронтендом. Ви можете вибрати оптимальний розмір і конфігурацію серверів для вашого додатку.

2) Amazon RDS (Relational Database Service): Amazon RDS може бути використаний для розгортання та керування реляційною базою даних PostgreSQL, яка буде використовуватися в якості бекенду для зберігання даних користувачів та інших інформаційних ресурсів.

3) Amazon S3 (Simple Storage Service): Amazon S3 використовується для зберігання медіафайлів, таких як фотографії користувачів та інші великі файли. Ви можете використовувати S3 для забезпечення доступу до цих файлів через ваш фронтенд.

4) Amazon API Gateway: Amazon API Gateway може служити в якості центрального маршрутизатора для вашого додатку, оброблюючи HTTP-запити та передаючи їх до відповідних мікросервісів чи серверів.

5) Amazon Cognito: Amazon Cognito може бути використаний для управління аутентифікацією та авторизацією користувачів. Він допоможе забезпечити безпеку та контроль доступу до різних функцій вашого додатку.

6) Amazon Lambda: Amazon Lambda дозволяє створювати функції без серверів, які можуть виконувати обчислення відповідно до потреб вашого додатку.

7) Amazon DynamoDB: Якщо у вас є потреба у нереляційних базах даних, то Amazon DynamoDB може бути використаний для зберігання та управління нереляційною інформацією.

8) Amazon Sumerian (для розробки 3D-контенту): Якщо ви плануєте використовувати 3D-контент для додатку, Amazon Sumerian може допомогти створити і розгорнути такий контент.

9) Amazon Ethereum (для роботи з NFT на платформі Ethereum): Якщо ви плануєте використовувати NFT як внутрішню валюту, вам може знадобитися підтримка для роботи з Ethereum-блокчейном, і Amazon Ethereum може бути корисним в цьому відношенні.

Node.js:

Node.js - це середовище виконання JavaScript на стороні сервера. Він володіє декількома перевагами для створення бекенду вашого проекту "Dvij", але це більше стосується вибору мови програмування, ніж хмарної платформи. Node.js може бути використаний на AWS або Heroku для розгортання вашого додатку.

Heroku:

Heroku - це платформа для розгортання та управління веб-додатками, яка спрощує процес розробки та розгортання. Heroku надає обліковий запис з вибором сервісів для створення додатків. Ось детальніше про Heroku:

Зручність: Heroku спрощує розгортання і управління додатками за допомогою командного рядка або веб-інтерфейсу.

Масштабованість: Heroku дозволяє легко масштабувати додатки, включаючи автоматичне масштабування на основі навантаження.

Підтримка різних мов: Heroku підтримує різні мови програмування, включаючи Node.js, Python, Ruby та інші.

Чому AWS може бути кращим вибором для написання бекенду вашого проекту "Dvij":

Широкий вибір сервісів: AWS надає широкий спектр інфраструктурних і сервісних рішень, що можуть бути корисними для створення складного соціального мережевого додатку, включаючи обчислювальні потужності, бази даних, зберігання та інше.

Масштабованість: AWS дозволяє легко масштабувати додаток відповідно до зростаючої кількості користувачів, що є важливим для соціальної мережі.

Безпека і доступність: AWS забезпечує високий рівень безпеки та доступності для ваших даних і додатків.

Гнучкість у виборі технологій: Ви можете використовувати будь-яку мову програмування та технології для розробки бекенду на AWS, включаючи Node.js.

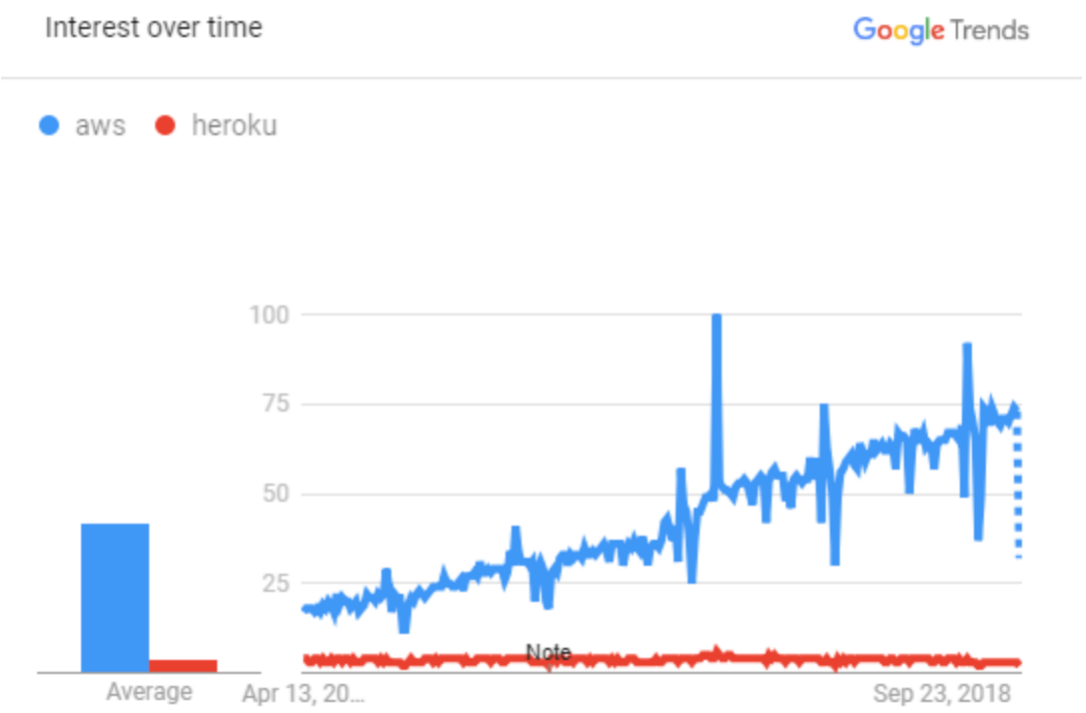


Рис.1.6. Популярність AWS і Heroku

Отже, AWS може бути кращим вибором для соціальної мережі "Dvij", оскільки він надає більше можливостей і ресурсів для розгортання та управління серверним бекендом.

1.5.3. Порівняння баз даних PostgreSQL та MySQL

Розглянемо технології PostgreSQL та MySQL та пояснимо, чому саме PostgreSQL може бути кращим вибором для вашого проекту "Dvij" для написання бекенду.

PostgreSQL:

PostgreSQL - це відкрита система керування реляційними базами даних (СКБД), яка має деякі переваги порівняно з MySQL і є більш потужною та розширюваною системою управління даними. Ось детальніше про PostgreSQL:

Розширені функції: PostgreSQL підтримує багато розширених функцій, таких як JSON-операції, геопросторові дані та повний текстовий пошук. Це дозволяє легко працювати з різними типами даних, що може бути корисним для вашого проекту, який має інтерактивну карту та обробку контенту.

ACID-сумісність: PostgreSQL є ACID-сумісною СКБД, що гарантує надійну та консистентну обробку даних. Це важливо для соціальної мережі, де дані користувачів повинні бути збережені та оброблені надійно.

Масштабованість: PostgreSQL має можливість масштабування і може бути використаний для створення великих та високонавантажених додатків.

Розширюваність: PostgreSQL дозволяє створювати власні розширення та функції, що дозволяє розширити функціональність СКБД за потребою.

Спільнота та підтримка: PostgreSQL має активну спільноту користувачів і розробників, яка надає швидку підтримку та розвиток.

MySQL:

MySQL - це інша відкрита система керування реляційними базами даних. Вона також є популярною та широко використовується, але має деякі відмінності від PostgreSQL. MySQL може бути вибраним, якщо вам важлива простота та швидкість.

Чому PostgreSQL може бути кращим вибором для проекту "Dvij":

Розширені функції: Враховуючи вашу ідею про інтерактивну карту та обробку різних типів контенту, PostgreSQL надає більше можливостей для обробки такого різноманітного контенту та запитів.

ACID-сумісність: Забезпечення надійності та консистентності даних важливо для соціальної мережі, і PostgreSQL має сильну підтримку для цього.

Масштабованість та розширюваність: PostgreSQL може бути масштабованим та розширюваним для ваших потреб, що важливо для підтримки зростаючої аудиторії.

Спільнота та підтримка: PostgreSQL має велику та активну спільноту користувачів та розробників, що забезпечує надійну підтримку та розвиток технології.

Цього року PostgreSQL випередив MySQL і посів перше місце. Професійні розробники частіше використовують PostgreSQL (50%), ніж ті, хто вчиться програмувати, тоді як останні більш схильні використовувати MySQL (54%). MongoDB використовується приблизно на такому самому рівні як професійними розробниками, так і тими, хто вчиться програмувати, і це друга найпопулярніша база даних серед тих, хто вчиться програмувати (після MySQL).

Рис.1.6. Популярність PostgreSQL і MySQL

Узагальнюючи, PostgreSQL може бути кращим вибором для вашого проекту "Dvij" завдяки своїм розширеним можливостям та надійності у керуванні даними, що особливо важливо для соціальної мережі з багатьма типами контенту та користувачів.

1.5.4. Порівняння сервісів генерації зображення MidJourney і Dali AI

MidJourney API і Dali AI API - це дві різні технології для обробки та генерації зображень, і кожна з них має свої особливості та використання. Опишу обидві технології та поясню, чому MidJourney API може бути кращим вибором для вашого проекту:

MidJourney API:

Призначення: MidJourney API - це інтерфейс програмування застосунків, який використовується для роботи з глибокими нейронними мережами та обробки зображень. Він може бути використаний для різних завдань, пов'язаних з обробкою та покращенням зображень, включаючи покращення аватарок користувачів.

Переваги для проекту:

Якість обробки зображень: MidJourney API використовує глибокі нейронні мережі, що дозволяє покращити якість та деталі зображень.

Генерація аватарок: Ця технологія може бути корисною для генерації унікальних та якісних аватарок для користувачів вашої соціальної мережі.

Реалістичність результатів: MidJourney API дозволяє створювати більш реалістичні зображення, що може покращити сприйняття користувачами.

Dali AI API:

Призначення: Dali AI API - це інтерфейс програмування застосунків для обробки зображень, який також використовує штучний інтелект для генерації та обробки графіки.

Переваги: Дали AI також може бути корисним для покращення зображень, але він може мати свої власні особливості та переваги.

MidJourney API може бути використаний в соціальній мережі для генерації зображень аватарок користувачів наступним чином:

Інтеграція API: Спочатку необхідно створити інтеграцію MidJourney API у вашому веб-додатку або сервері, який оброблює запити від клієнтів.

Запит від користувача: Коли користувач реєструється в соціальній мережі або оновлює свій профіль, ви можете надати йому можливість змінити або створити свою аватарку.

Вибір параметрів: Користувач має можливість вибрати параметри для своєї аватарки, такі як стиль, колір, фон, обличчя тощо. Це може бути реалістичним стилем або абстрактним, в залежності від ваших можливостей та налаштувань MidJourney API.

Запит до MidJourney API: Коли користувач вибрав параметри, ваш веб-додаток може створити запит до MidJourney API, включаючи обрані параметри та будь-яку іншу необхідну інформацію.

Обробка результату: MidJourney API оброблює запит і повертає зображення, яке відповідає обраним параметрам. Ваша соціальна мережа може зберегти це зображення як аватарку користувача та відобразити її на його профілі.

Збереження та відображення: Згенерована аватарка зберігається на сервері або в хмарному сховищі та відображається на профілі користувача. Користувач може переглянути свою нову аватарку та, при необхідності, змінити її.

MidJourney API дозволяє автоматично генерувати унікальні та креативні аватарки для користувачів, що може зробити їхні профілі більш цікавими та відмінними. Важливо належним чином інтегрувати та керувати цим API в соціальній мережі, щоб забезпечити зручний та якісний досвід користувачів.

1.5.5. Опис платформ iOS, Android і visionOS

Платформи iOS, Android і visionOS - це операційні системи, які використовуються для розробки мобільних та відомостей пристроїв. Ось короткий опис кожної з цих платформ:

iOS:

Розробник: iOS є мобільною операційною системою, розробленою і підтримуваною Apple Inc.

Програмування: Розробка додатків для iOS зазвичай використовує мови програмування Swift і Objective-C, а також інструменти розробки, такі як Xcode.

Апаратні вимоги: iOS працює на пристроях, вироблених Apple, таких як iPhone, iPad і iPod Touch.

Особливості: iOS відома своєю високою стабільністю, безпекою і екосистемою додатків App Store. Вона також підтримує багато інноваційних функцій, таких як розширена реальність (AR), глибокий інтеграція зі службами Apple і інші.

Android:

Розробник: Android є мобільною операційною системою, розробленою Google.

Програмування: Розробка додатків для Android зазвичай використовує мови програмування Java або Kotlin, а також Android Studio як основний інструмент розробки.

Апаратні вимоги: Android працює на різних мобільних і планшетних пристроях від різних виробників.

Особливості: Android є однією з найпопулярніших мобільних платформ у світі і відомий своєю відкритістю та налаштованістю. Він також підтримує різні типи пристроїв, включаючи смартфони, планшети, телевізори та інші.

visionOS:

Розробник: visionOS є операційною системою, розробленою спеціально для відомостей та розумних окулярів.

Програмування: Розробка додатків для visionOS відбувається з використанням спеціалізованих інструментів та технологій, розроблених для створення додатків для розумних окулярів.

Апаратні вимоги: visionOS призначений для використання на відомостях та розумних окулярах, які підтримують дану операційну систему.

Особливості: visionOS спроектований для створення інтерактивних додатків, які можуть бути використані на розумних окулярах, для розширення реальності та інших цифрових досвідів.

Windows

У вашому додатку для visionOS ви можете створити одне або кілька вікон. Вони створені з використанням SwiftUI і містять традиційні відображення та елементи управління. Ви також можете збагатити свій досвід, додавши тривимірний контент.

Volumes (Об'єми)

Додайте глибину до вашого додатку за допомогою тривимірного об'єму. Об'єми - це сцени SwiftUI, які можуть відображати тривимірний контент за допомогою RealityKit або Unity, створюючи враження, які можна переглядати з будь-якого кута в Спільному просторі або Повному просторі додатка.

Spaces (Простори)

За замовчуванням додатки запускаються в Спільному просторі, де вони існують поруч - подібно до кількох додатків на робочому столі Mac. Додатки можуть використовувати вікна та об'єми для відображення контенту, і користувач може переносити ці елементи, куди завгодно. Для більш іммерсивного досвіду додаток може відкрити відокремлений Повний простір, в якому буде відображуватися лише вміст цього додатка. У межах Повного простору додаток може використовувати вікна та об'єми, створювати необмежений тривимірний контент, відкривати портал в інший світ або навіть повністю погрузити людей в середовище.

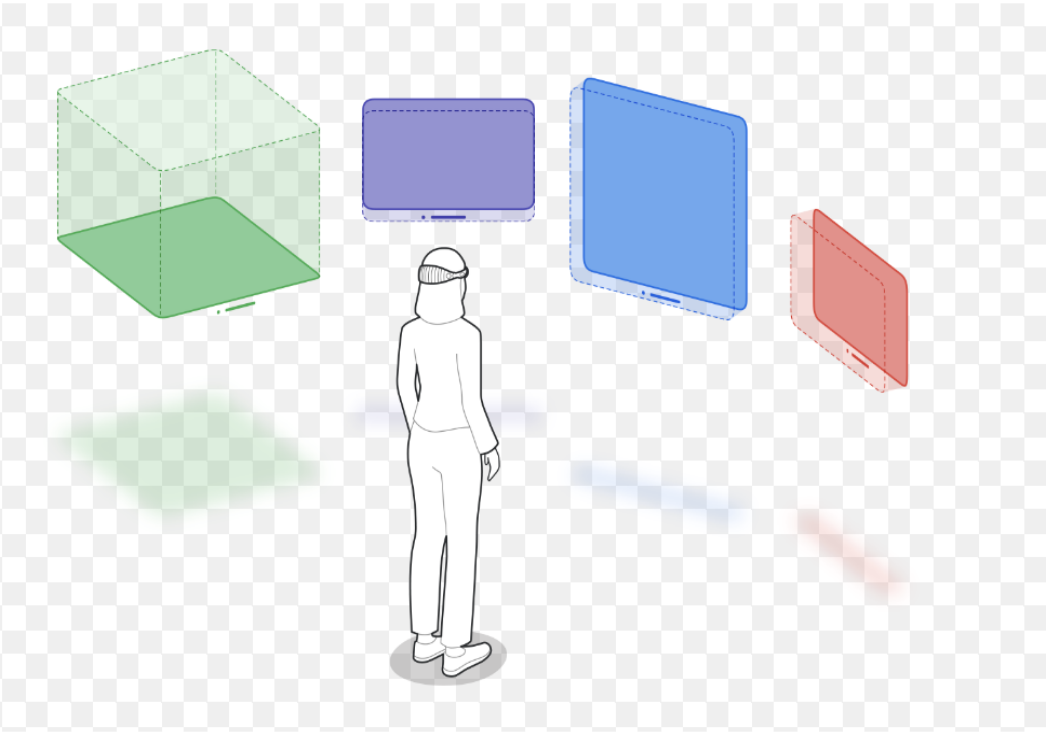


Рис.1.7. Вікна в visionOS

Вибір платформи залежить від цільової аудиторії, цілей проекту та технічних можливостей. iOS та Android - це стандартні мобільні платформи, в той час як visionOS використовується для розумних окулярів і розширених реальності додатків.

У цьому проекті використовуються різноманітні технології для створення інноваційної соціальної мережі нового покоління "Dvij". Основні технології та їхнє використання включають:

1) Node.js: Node.js використовується для написання серверного бекенду, обробки HTTP-запитів, керування базою даних та реалізації реального часу. Його асинхронна природа і висока продуктивність роблять його ідеальним для веб-додатків з багатьма одночасними клієнтами.

2) PostgreSQL: PostgreSQL обрана як база даних для зберігання інформації користувачів та інших даних проекту. Вона відзначається надійністю, широкими можливостями та можливістю масштабування.

3) MidJourney API: Ця технологія допомагає в генерації зображень аватарок користувачів, забезпечуючи візуальну привабливість профілів.

4) Amazon Web Services (AWS): AWS використовується для розгортання та керування інфраструктурою додатку, забезпечення безпеки та масштабування.

Ці технології спільно допомагають створити унікальну соціальну мережу "Dvij" з інтерактивною картою, грою та реалізацією реального часу. Їхнє використання дозволяє нам надати користувачам зручний та цікавий інструмент для спілкування та взаємодії в будь-якій тематиці.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі представлено опис соціальної мережі нового покоління "Dvij", яка має на меті надати користувачам зручний та доступний функціонал для спілкування та взаємодії на різні теми. Основними характеристиками цієї соціальної мережі є:

Призначення: "Dvij" створена з метою надання можливості користувачам спілкуватися та взаємодіяти з друзями та однодумцями на будь-яку тематику, включаючи IT, готування, танці, подорожі, настільні ігри, аніме, велосипеди, бізнес і багато інше. Вона також пропонує інтерактивну карту світу улюблених речей та пов'язаного з ними контенту, а також функцію записної книги та пошуковика для зручного пошуку необхідної інформації.

Унікальність: "Dvij" відрізняється від інших соціальних мереж тим, що вона виглядає як віртуальна соціальна гра, де користувачі можуть створювати власний віртуальний світ навколо своїх інтересів та взаємодіяти з іншими учасниками. Проект надає API для додавання власного функціоналу, що дозволяє користувачам налаштовувати платформу під свої потреби.

Гейміфікація: Головною особливістю "Dvij" є гейміфікація, що дозволяє перетворити будь-який процес в гру. Розробка включає в себе використання механік гейміфікації для залучення та утримання користувачів.

Основний функціонал для MVP: Описано основні елементи і функції головного екрану, включаючи 3D карту, розу вітрів, фільтри, нейроаву користувача, френдсхаб та фотохаб. Важливою частиною цього функціоналу є можливість переходу до профілів користувачів та перегляду їх фотографій та участі в "двіжах".

Використані технології: У розділі також наведено перелік використаних технологій для розробки "Dvij", включаючи JavaScript, AWS, PostgreSQL для бекенду, Unity для фронтенду, штучний інтелект для генерації контенту та NFT для внутрішньої валюти на платформі Ethereum.

Розділ 1 встановлює загальний контекст і визначає ключові аспекти соціальної мережі "Dvij", які будуть подальше розглянуті та досліджені в дипломній роботі.

РОЗДІЛ 2
АРХІТЕКТУРА, СТРУКТУРА ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

2.1. Платформа програмування Node js

2.1.1. Призначення та особливості

Node.js - це серверна платформа для виконання JavaScript-коду. Вона ґрунтується на движку V8, розробленому Google, і надає можливість виконувати JavaScript на стороні сервера. Ця платформа відома своєю асинхронною природою, що дозволяє обробляти багато одночасних запитів без блокування виконання. Це особливо корисно для веб-додатків, які потребують обробки багатьох одночасних клієнтів.

Node.js - це високопродуктивне середовище виконання JavaScript, призначене для серверного програмування та створення мережевих додатків. Основною особливістю Node.js є можливість виконання JavaScript на стороні сервера, що робить його ідеальним інструментом для створення швидких та масштабованих серверних додатків.

Основні призначення Node.js в моєму проекті включають наступне:

1) Створення серверної частини: Node.js використовується для створення серверної частини мого додатку, яка обробляє HTTP-запити від клієнтів і забезпечує доступ до різних функціональних можливостей, таких як робота з базою даних та взаємодія з іншими сервісами.

2) Підтримка асинхронного програмування: Однією з ключових особливостей Node.js є підтримка асинхронного програмування. Це дозволяє обробляти багатозадачні операції без блокування інших операцій, що забезпечує

швидку реакцію сервера на запити користувачів.

3) Розширені бібліотеки: Node.js має велику кількість розширених бібліотек, які допомагають вирішувати різні завдання, включаючи роботу з мережевими пристроями, розробку API, обробку файлів тощо.

4) Підтримка пакетного менеджера npm: Node.js ідеально поєднується з пакетним менеджером npm, який дозволяє легко встановлювати сторонні бібліотеки та модулі для розширення функціональності додатку.

Node.js є надійним та ефективним середовищем для створення серверних додатків і ідеально підходить для реалізації функціональності мого проекту.

2.1.2. Основні інструменти та компоненти

Node.js має ряд основних інструментів і компонентів, які грають важливу роль у розробці серверних додатків. Основні з них включають:

1) V8 Engine: Node.js використовує движок V8, розроблений Google, для виконання JavaScript-коду. V8 Engine відомий своєю високою швидкістю та ефективністю, що робить Node.js швидким і продуктивним середовищем для виконання коду.

Движок V8 - це ключовий компонент Node.js, і він відзначається значними перевагами щодо швидкодії та продуктивності. Ось додаткова інформація:

1.1) Що таке V8 Engine:

Платформа виконання: V8 Engine - це віртуальна машина з відкритим кодом, розроблена Google для виконання JavaScript-коду. Вона використовується в серверному оточенні Node.js, щоб виконувати JavaScript на стороні сервера.

1.2) Переваги V8 Engine для Node.js:

Висока продуктивність: V8 Engine славиться своєю швидкодією виконання завдяки ряду оптимізацій, включаючи JIT-компіляцію (Just-In-Time). Вона дозволяє обробляти велику кількість запитів на сервері, що є критичним для

високонавантажених застосувань. Середні дані вказують на те, що V8 Engine є одним з найшвидших движків виконання JavaScript-коду.

Асинхронна природа: Використання подій та зворотніх викликів дозволяє Node.js та V8 Engine обробляти запити асинхронно. Це дозволяє серверу обробляти одночасно багато запитів, що є важливим для додатків з великою активністю користувачів.

1.3) Швидкість та продуктивність:

За тестами профілювання і вимірювання продуктивності встановлено, що V8 Engine вирізняється високою швидкістю виконання JavaScript-коду, що робить його ідеальним вибором для великих та високонавантажених проектів.

Вища швидкість виконання означає, що Node.js з V8 Engine дозволяє обробляти більше запитів на одній та тій же апаратній архітектурі, що допомагає знизити витрати на обладнання та забезпечує кращу швидкість відгуку для користувачів.

1.4) Популярність та активний розвиток:

V8 Engine є одним з найпопулярніших та найвикористовуваніших движків JavaScript у світі. Він активно розвивається командою інженерів з Google та спільнотою розробників, що забезпечує постійне покращення продуктивності та оптимізацію ресурсів.

Інші проекти та платформи, включаючи Deno, також використовують V8 Engine для виконання JavaScript-коду, що підтверджує його широкий застосунок і популярність у розробницькому середовищі.

Таким чином, V8 Engine відіграє важливу роль у Node.js, надаючи можливість розробникам писати високопродуктивні серверні додатки на JavaScript та забезпечуючи високу швидкість виконання, що важливо для сучасних серверних застосувань.

2) npm (Node Package Manager): npm є пакетним менеджером, який постачається разом з Node.js. Він дозволяє розробникам легко встановлювати,

оновлювати та керувати сторонніми бібліотеками та модулями, які використовуються в проекті.

npm - це стандартний менеджер пакетів для платформи Node.js, який включається встановленим Node.js, що робить його основним інструментом для розробки. Він має ключове значення для проектів, які використовують Node.js, оскільки допомагає розробникам управляти залежностями, сторонніми бібліотеками та скриптами, необхідними для їх проектів. Ось детальний огляд npm:

2.1) Завантаження та Установка Пакетів:

npm дозволяє розробникам завантажувати та встановлювати пакети, які їм необхідні для проекту. Це включає в себе бібліотеки, фреймворки, плагіни та інші компоненти, які допомагають розширювати функціональність їх додатків.

2.2) Управління Залежностями:

Один із ключових аспектів npm - це можливість розробників визначати та управляти залежностями їх проекту. Вони можуть вказати, які версії пакетів необхідні для їх додатку, і npm автоматично завантажить та встановить відповідні версії.

2.3) Локальні та Глобальні Пакети:

npm розділяє пакети на два типи: локальні та глобальні. Локальні пакети встановлюються та використовуються для конкретного проекту, тоді як глобальні пакети встановлюються на рівні користувача та можуть бути доступні для будь-якого проекту.

2.4) Пакети для Розробки:

Багато розробників використовують npm для управління пакетами для розробки, які включають в себе інструменти для тестування, лінтування, збірки та інші корисні утиліти. Це допомагає автоматизувати рутинні завдання розробки та забезпечити високу якість коду.

2.5) Скрипти та Пост-Інсталяційні Дії:

npm дозволяє розробникам визначати власні скрипти та команди в файлі package.json. Це допомагає забезпечити швидкий доступ до різних функцій та процесів розробки.

2.6) Широка Екосистема:

Велика кількість пакетів, доступних через npm, робить його розширенням, яке використовується всередині спільноти розробників Node.js. Це дозволяє розробникам знайти відповідні рішення для своїх проектів і значно спрощує розробку.

2.7) Швидкість та Надійність:

npm відомий своєю швидкістю та надійністю. Він використовує CDN для завантаження пакетів, що забезпечує швидкий доступ до них, а також підтримує кешування для збільшення продуктивності та надійності завантаження пакетів.

2.8) Сумісність та Розширені Можливості:

npm дозволяє розробникам розширювати можливості Node.js та підключати різноманітні плагіни, що полегшує розробку та забезпечує більше гнучкості.

В іншому контексті, npm є необхідним інструментом для будь-якого проекту, який використовує Node.js. Він полегшує управління залежностями, спрощує розробку та забезпечує доступ до безлічі пакетів та інструментів для розробників.

Список пакетів npm використаних в проекті:

```
"dependencies": {
   "@nestjs/common": "^10.0.0",
   "@nestjs/config": "^3.0.0",
   "@nestjs/core": "^10.0.0",
   "@nestjs/event-emitter": "^2.0.0",
   "@nestjs/jwt": "^10.1.0",
   "@nestjs/passport": "^10.0.0",
   "@nestjs/platform-express": "^10.0.0",
   "@nestjs/platform-ws": "^10.1.0",
   "@nestjs/swagger": "^7.0.5",
```

```
    "@nestjs/typeorm": "^10.0.0",
    "@nestjs/websockets": "^10.1.0",
    "@nestjsx/crud": "^5.0.0-alpha.3",
    "@types/lodash": "^4.14.195",
    "@types/multer": "^1.4.7",
    "@types/passport-jwt": "^3.0.9",
    "axios": "^1.4.0",
    "bcryptjs": "^2.4.3",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.0",
    "lodash": "^4.17.21",
    "passport": "^0.6.0",
    "passport-jwt": "^4.0.1",
    "passport-local": "^1.0.0",
    "pg": "^8.11.0",
    "reflect-metadata": "^0.1.13",
    "rxjs": "^7.8.1",
    "typeorm": "^0.3.17"
  },
  "devDependencies": {
    "@nestjs/cli": "^10.0.0",
    "@nestjs/schematics": "^10.0.0",
    "@nestjs/testing": "^10.0.0",
    "@types/express": "^4.17.17",
    "@types/jest": "^29.5.2",
    "@types/node": "^20.3.1",
    "@types/supertest": "^2.0.12",
    "@typescript-eslint/eslint-plugin": "^5.59.11",
    "@typescript-eslint/parser": "^5.59.11",
```

```
  "eslint": "^8.42.0",
  "eslint-config-prettier": "^8.8.0",
  "eslint-plugin-prettier": "^4.2.1",
  "jest": "^29.5.0",
  "prettier": "^2.8.8",
  "source-map-support": "^0.5.21",
  "supertest": "^6.3.3",
  "ts-jest": "^29.1.0",
  "ts-loader": "^9.4.3",
  "ts-node": "^10.9.1",
  "tsconfig-paths": "^4.2.0",
  "typescript": "^5.1.3"
},
```


3) Event Loop: Node.js використовує подійний цикл для обробки подій та асинхронних запитів. Це дозволяє реагувати на події, такі як HTTP-запити, без блокування інших операцій і забезпечує ефективне використання ресурсів сервера.

Петля Подій (Event Loop) в Node.js

Петля Подій (Event Loop) - це ключовий аспект асинхронної природи Node.js, який дозволяє ефективно обробляти одночасні запити без блокування виконання. Вона є однією з фундаментальних частин архітектури Node.js і визначає, як Node.js обробляє події, запити та виконує асинхронний код.

Основні аспекти Event Loop:

1. Подій та Очікування (Events and Waiting): В Node.js багато операцій є асинхронними. Замість блокування потоку виконання для очікування завершення операції, Node.js додає події в чергу очікування (event queue) і продовжує виконання наступних операцій.

2. Цикл Опитування (Polling Loop): Event Loop постійно опитує чергу очікування, перевіряючи, чи є події для обробки. Якщо в черзі маються події, Event Loop починає обробку їх по черзі.

3. Виконання Колбеків (Execution of Callbacks): Коли Event Loop знаходить подію, відповідний колбек (функція, що повинна виконатися) виконується. Це дозволяє програмі відповідати на події асинхронно.

4. Повторення Циклу (Repeating the Loop): Event Loop продовжує роботу до тих пір, поки черга очікування не стане порожньою. Коли це відбудеться, Node.js може призупинити виконання або завершити роботу.

Основні переваги Event Loop в Node.js:

- Асинхронність: Петля Подій дозволяє Node.js ефективно взаємодіяти з багатьма клієнтами без блокування виконання.

- Ефективність: Node.js використовує події та зворотні виклики, що дозволяє ефективно використовувати ресурси сервера та реагувати на події в реальному часі.

- Підтримка багатьох одночасних з'єднань: Ця архітектура робить Node.js ідеальним для створення серверів, що обслуговують багато клієнтів одночасно.

Event Loop є важливою складовою Node.js, яка допомагає створити ефективні та відмінно відзначені за продуктивністю додатки, особливо в області серверного програмування та обробки багатьох одночасних запитів

4) HTTP Module: HTTP модуль Node.js дозволяє створювати HTTP-сервери та клієнти. Це робить його ідеальним для розробки веб-додатків та API.

5) File System Module: Node.js надає модуль для роботи з файловою системою сервера. Це допомагає зчитувати та записувати файли, що використовуються в додатку.

6) Express.js: Express.js є популярним фреймворком для розробки веб-додатків на основі Node.js. Він надає широкий спектр інструментів для створення маршрутів, обробки HTTP-запитів і створення API.

7) WebSocket: WebSocket підтримує реалізацію багатоканальних зв'язків між сервером і клієнтом. Це особливо корисно для реалізації взаємодії в реальному часі.

8) Cluster Module: Cluster Module дозволяє використовувати багато процесів для роботи з сервером, що підвищує його продуктивність та надійність.

9) Debugger: Node.js має власний інтерактивний відладчик, який допомагає виявляти та усувати помилки в коді.

Ці інструменти та компоненти роблять Node.js потужним та гнучким середовищем для розробки серверних додатків, особливо в контексті мого проекту.

Висока продуктивність: Node.js відомий своєю швидкістю виконання завдяки використанню движка V8. Це дозволяє обробляти велику кількість запитів в секунду, що є дуже важливим для соціальної мережі з очікуваною великою активністю користувачів.

Асинхронність: Node.js використовує подій та зворотні виклики для обробки запитів. Це дозволяє більш ефективно використовувати ресурси сервера та забезпечувати швидку реакцію на запити користувачів.

Велика спільнота та пакети: Node.js має активну спільноту розробників та численні готові пакети (модулі), які полегшують розробку. Це сприяє використанню сторонніх бібліотек та розширює можливості проекту.

Node.js використовується для написання серверної частини додатку. Вона обробляє HTTP-запити, взаємодіє з базою даних PostgreSQL, керує відображенням даних на фронтенді за допомогою Unity та забезпечує обмін даними в реальному часі. Технічно Node.js включає в себе ряд бібліотек для обробки HTTP, створення API, виконання запитів до бази даних та підтримки асинхронного програмування. Це забезпечує роботу мого додатку та надає йому необхідні можливості для взаємодії з користувачами.

2.2. Середовище програмування Visual studio

2.2.1. Опис та переваги використання середовища

Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft, призначене для розробки різних видів програмного забезпечення, включаючи

веб-додатки, настільні додатки, мобільні додатки, хмарні служби та багато іншого. Ось деякі ключові аспекти середовища програмування Visual Studio:

Інтегрована розробка: Visual Studio надає інтегроване середовище для розробки, що означає, що ви можете працювати над всіма аспектами проекту в одній програмі. Від створення коду до налагодження та тестування, ви можете виконувати всі ці операції безпосередньо в середовищі Visual Studio.

Мови програмування: Visual Studio підтримує різні мови програмування, такі як C#, C++, F#, Visual Basic, Python, і багато інших. Ви можете обирати мову, яка найкраще підходить для вашого проекту.

Розширення та плагіни: Visual Studio має велику кількість розширень та плагінів, які розширюють його функціональність. Це дозволяє розробникам налаштовувати середовище розробки під свої потреби та використовувати різноманітні інструменти сторонніх розробників.

Візуальний редагувальник: Visual Studio надає потужний візуальний редагувальник коду з функціями, які спрощують написання, редагування та форматування коду. Він також підтримує функцію автодоповнення, яка полегшує роботу зі сніпетами коду та бібліотеками.

Відлагодження та профілювання: Visual Studio надає потужні інструменти для відлагодження коду. Розробники можуть використовувати відлагодження крок за кроком, перегляд змін змінних, аналіз стеку викликів та інші функції для пошуку та виправлення помилок в програмному коді. Також доступні інструменти профілювання для визначення швидкодії програми.

Підтримка для хмарних служб: Visual Studio інтегрований з платформами хмарних служб Microsoft, такими як Azure. Це дозволяє легко розгортати та керувати хмарними додатками та послугами безпосередньо з середовища розробки.

Тестування та автоматизація: Visual Studio надає інструменти для створення та виконання автоматизованих тестів для вашого коду. Це полегшує валідацію та підтримку якості вашого програмного забезпечення.

Версіонування інструментів: Visual Studio інтегрований з системами контролю версій, такими як Git, що полегшує спільну роботу над кодом у команді та відстеження змін.

Visual Studio - це потужне середовище розробки, яке допомагає розробникам створювати різноманітні види програмного забезпечення з використанням різних мов програмування та інструментів.

Розширення (Extensions) в середовищі Visual Studio - це додаткові компоненти та плагіни, які дозволяють розширити функціональність і можливості самого Visual Studio. Використовуючи розширення, розробники можуть адаптувати середовище до своїх потреб та специфічних завдань, полегшити роботу та зробити її більш продуктивною. Ось декілька важливих аспектів стосовно розширень в Visual Studio:

1. Розширення для підтримки мов програмування: Visual Studio підтримує безліч мов програмування, і розширення можуть додавати підтримку для конкретних мов, дозволяючи розробникам працювати з ними більш ефективно. Наприклад, розширення можуть додати автодоповнення, відлагодження та інші інструменти для конкретної мови.

2. Інструменти для веб-розробки: Розширення можуть додати інструменти для розробки веб-додатків, включаючи редактори HTML, CSS та JavaScript, відлагодження веб-сторінок та підтримку фреймворків веб-розробки.

3. Розширення для роботи з базами даних: Розширення можуть додавати підтримку для роботи з реляційними та нереляційними базами даних. Вони можуть надавати інструменти для створення та редагування схем баз даних, виконання SQL-запитів та інше.

4. Інструменти для роботи з системами контролю версій: Розширення можуть додати підтримку різних систем контролю версій, таких як Git, SVN, Mercurial і багато інших. Вони дозволяють розробникам взаємодіяти з репозиторіями, відстежувати зміни та здійснювати коміти.

5. Розширення для тестування та автоматизації: Деякі розширення можуть додати інструменти для створення та виконання автоматизованих тестів, а також для оцінки якості коду та виявлення помилок.

6. Розширення для роботи з хмарними послугами: Розширення можуть додавати інтеграцію з хмарними платформами, такими як Azure, AWS, Google Cloud і інші. Вони допомагають розробникам легко розгортати та керувати хмарними додатками.

7. Розширення для інтеграції з сторонніми інструментами: Розширення можуть додати підтримку інших інструментів та сервісів, що використовуються в конкретному проекті.

Використання розширень у Visual Studio може значно полегшити роботу розробників, дозволяючи їм налаштовувати інструменти розробки під свої потреби та специфічні завдання проекту.

Code Spell Checker: Це розширення допомагає виявляти та виправляти помилки в орфографії та граматиці в вашому коді. Воно допомагає зробити ваш код більш читабельним і професійним.

Eslint: Це розширення допомагає виявляти та виправляти помилки в стилі коду в вашому проекті. Ви можете налаштувати правила стилю та автоматично відформатувати свій код відповідно до цих правил.

Git Graph: Це розширення надає графічний інтерфейс для перегляду та аналізу історії вашого Git-репозиторію. Ви можете бачити гілки, коміти, злиття та інші дії на графіку, що допомагає зрозуміти структуру вашого проекту.

Git Lens: Це розширення додає додатковий функціонал для роботи з Git в Visual Studio Code. Воно надає детальну інформацію про кожний коміт, авторів та інші параметри. Ви можете переглядати зміни у файлі на рівні комітів.

Prettier: Це розширення автоматично відформатовує ваш код відповідно до налаштованих стандартів. Воно допомагає зробити ваш код більш читабельним та спрощує процес підтримки стилю коду.

Інші розширення можуть надавати різні функціональність, такі як робота з базами даних, робота зі зворотними викликами, робота з хмарними послугами тощо. Вибір розширень залежить від потреб вашого проекту та вашого стеку технологій.

2.3. Розробка додатку

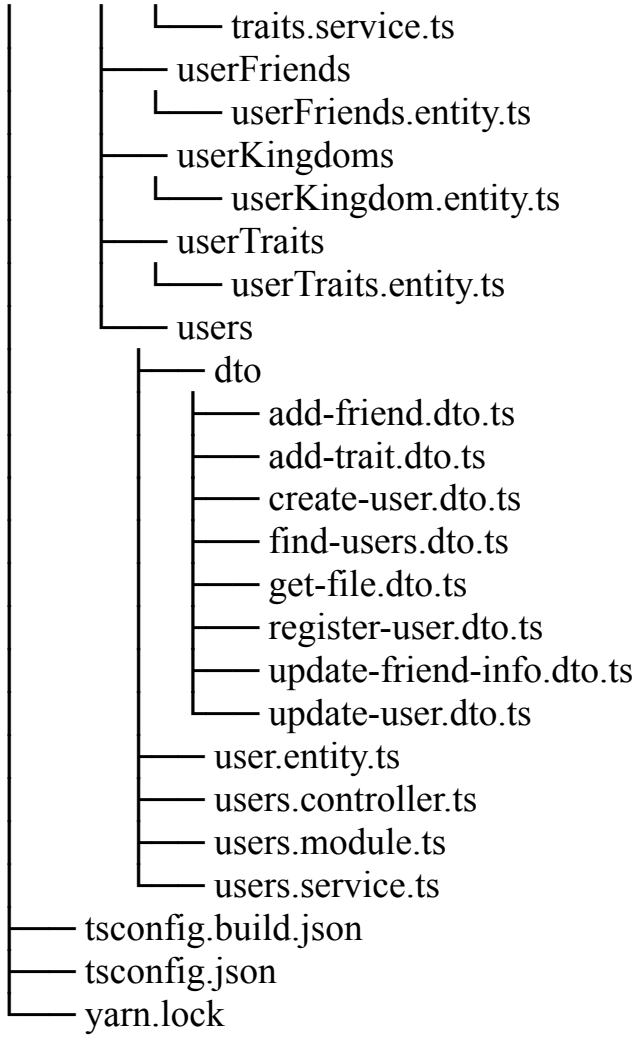
2.3.1. Архітектура серверної частини додатку

Структура проекту:

```
├── README.md
├── app-data-source.ts
├── client
│   └── index.html
├── files
│   ├── user_4_1694882731161.jpeg
│   ├── user_5_1694882778170.jpeg
│   ├── user_6_1694882787997.jpeg
│   └── user_6_1694882840038.jpeg
├── nest-cli.json
├── ormconfig.ts
├── package.json
├── src
│   ├── infrastructure
│   │   ├── config
│   │   │   ├── configuration.ts
│   │   │   └── typeorm.ts
│   │   ├── enums
│   │   │   └── AchievementKey.enum.ts
│   │   ├── helpers
│   │   │   └── either.helper.ts
│   │   ├── interfaces
│   │   │   └── request.interface.ts
│   │   ├── pipes
│   │   │   └── schema_validation.pipe.ts
│   │   ├── type
│   │   │   └── parent.entity.ts
│   │   └── utils
│   │       ├── env.util.ts
│   │       └── queryToFindOperators.util.ts
```

```
    ├── main.ts
    ├── migrations
    │   ├── 1688461616152-Init.ts
    │   ├── 1688461624653-SeedAchievementsTable.ts
    │   ├── 1692267733015-RenameSkillsToTraits.ts
    │   ├── 1692270234456-CreateUserTraitsTable.ts
    │   ├── 1692356714123-RemoveCurrentDefaultLocaltion.ts
    │   ├── 1692360874098-CreateKingdomsTable.ts
    │   └── 1694960027416-CreateKingdomMessagesTable.ts
    └── modules
        ├── WebSocketModules
        │   ├── KingdomMessages
        │   │   ├── kingdom-chat.gateway.ts
        │   │   ├── kingdom-chat.module.ts
        │   │   └── kingdom-chat.service.ts
        │   └── adapters
        │       └── ws-adapter.ts
        ├── achievements
        │   ├── achievement.entity.ts
        │   ├── achievements.controller.ts
        │   ├── achievements.module.ts
        │   ├── achievements.service.ts
        │   └── dto
        │       ├── create-achievement.dto.ts
        │       └── update-achievement.dto.ts
        ├── app
        │   ├── app.controller.ts
        │   ├── app.module.ts
        │   └── app.service.ts
        └── auth
            ├── auth.controller.ts
            ├── auth.module.ts
            ├── auth.service.ts
            ├── dto
            │   ├── login.dto.ts
            │   └── signup.dto.ts
            ├── guards
            │   ├── jwt-auth.guard.ts
            │   ├── local-auth.guard.ts
            │   └── ws-auth.guard.ts
            └── strategies
                ├── jwt.strategy.ts
                └── local.strategy.ts
```

```
├── events
│   ├── dto
│   │   ├── create-event.dto.ts
│   │   ├── get-event-query.dto.ts
│   │   └── update-event.dto.ts
│   ├── event.entity.ts
│   ├── events.controller.ts
│   ├── events.module.ts
│   └── events.service.ts
├── kingdomMessages
│   ├── dto
│   │   ├── create-kingdomMessage.dto.ts
│   │   ├── get-kingdomMessages.dto.ts
│   │   └── update-kingdomMessage.dto.ts
│   ├── kingdomMessage.entity.ts
│   ├── kingdomMessages.controller.ts
│   ├── kingdomMessages.module.ts
│   └── kingdomMessages.service.ts
├── kingdoms
│   ├── dto
│   │   ├── create-kingdom.dto.ts
│   │   └── update-kingdom.dto.ts
│   ├── kingdom.entity.ts
│   ├── kingdoms.controller.ts
│   ├── kingdoms.module.ts
│   └── kingdoms.service.ts
├── places
│   ├── dto
│   │   ├── create-place.dto.ts
│   │   ├── get-place-query.dto.ts
│   │   └── update-place.dto.ts
│   ├── place.entity.ts
│   ├── places.controller.ts
│   ├── places.module.ts
│   └── places.service.ts
├── traits
│   ├── dto
│   │   ├── create-trait.dto.ts
│   │   ├── find-traits.dto.ts
│   │   └── update-trait.dto.ts
│   ├── trait.entity.ts
│   ├── traits.controller.ts
│   └── traits.module.ts
```



- README.md: Це файл з описом проекту та інструкціями щодо його використання.
- app-data-source.ts: Цей файл, ймовірно, містить джерело даних для мого проекту.
- client/index.html: Файл HTML, який ймовірно використовується для клієнтської частини мого проекту.
- files: Директорія, де зберігаються файли, ім'я яких включає ідентифікатор користувача та час створення (ім'я файлу може бути пов'язане з аватаром користувачів).
- nest-cli.json: Конфігураційний файл для NestJS, який, можливо, визначає параметри проекту.

- ormconfig.ts: Конфігураційний файл для TypeORM, який, можливо, визначає налаштування для роботи з базою даних.
- package.json: Файл з описом залежностей та скриптами проекту.
- src: Директорія, де розміщені вихідний код проекту.
- infrastructure: Директорія, що містить різноманітні утиліти та допоміжні класи для інфраструктури проекту.
- config: Налаштування проекту, включаючи конфігурацію бази даних та інші параметри.
- enums: Перерахування та ключі для проекту.
- helpers: Допоміжні функції та класи, наприклад, для роботи з помилками.
- interfaces: Інтерфейси та типи, які використовуються в моєму проекті.
- pipes: Класи для обробки HTTP-запитів та валідації даних.
- type: Загальні класи та типи для моїх моделей.
- migrations: Директорія з міграціями бази даних. Міграції визначають структуру бази даних та виконують зміни в ній.
- modules: Директорія, де реалізовані різні модулі мого додатку. Кожен модуль відповідає за певну функціональність.
- WebSocketModules: Модуль для роботи з WebSocket-з'єднаннями, можливо, для реалізації чату.
- KingdomMessages: Модуль для обробки повідомлень в рамках королівств.
- kingdom-chat.gateway.ts: Клас-шлюз для WebSocket, можливо, відповідає за обробку повідомлень у чаті королівства.
- kingdom-chat.module.ts: Модуль для роботи з чатами королівств.
- kingdom-chat.service.ts: Сервіс для обробки чатів королівств.
- adapters: Адаптери для роботи з WebSocket, можливо, для адаптації даних.
- achievements: Модуль для роботи з досягненнями користувачів.

- achievement.entity.ts: Модель для досягнень.
- achievements.controller.ts: Контролер для обробки запитів, пов'язаних з досягненнями.
- achievements.module.ts: Модуль для роботи з досягненнями.
- achievements.service.ts: Сервіс для роботи з досягненнями.
- dto: Об'єкти передачі даних для досягнень.
- інші модулі: Аналогічно до вищеперелічених модулів, кожен відповідає за певну частину функціональності мого проекту.
- tsconfig.build.json: Конфігураційний файл TypeScript для збірки мого проекту.
- tsconfig.json: Основний конфігураційний файл TypeScript для мого проекту.
- yarn.lock: Файл з фіксацією версій пакетів, встановлених за допомогою Yarn.

Ця структура розкриває організацію проекту з папок, файлів та їхньою функціональністю. Мій проект має багато модулів для різних аспектів моєї функціональності, і кожен з них має свою власну структуру.

2.3.2. Nest js фреймворк

NestJS - це серверний фреймворк для створення масштабованих та ефективних за допомогою TypeScript (або JavaScript) Node.js додатків. Він використовує сучасні підходи до розробки та базується на об'єктно-орієнтованих та функціональних принципах програмування.

Розглянемо код файлу main.ts:

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
```

```
  app.useWebSocketAdapter(new WsAdapter(app));

  AppDataSource.initialize()
    .then(() => {
      console.log('Data Source has been initialized!');
    })
    .catch((err) => {
      console.error('Error during Data Source initialization', err);
    });
  // Create a Swagger document options object
  const options = new DocumentBuilder()
    .setTitle('DVIJ API endpoints')
    .setDescription('API')
    .setVersion('1.0')
    .addTag('xD')
    .addBearerAuth(
      {
        type: 'http',
        scheme: 'bearer',
        bearerFormat: 'JWT',
        name: 'JWT',
        description: 'Enter JWT token',
        in: 'header',
      },
      'JWT-auth',
    )
    .build();

  // Create a Swagger document
  const document = SwaggerModule.createDocument(app, options);

  // Serve the Swagger document at /api
  SwaggerModule.setup('api', app, document);
  const port = 3000;
  await app.listen(port);
}
bootstrap();
```

Код, який наведений вище, є точкою входу додатку, написаного з використанням Nest.js, який базується на Node.js. Давайте розглянемо кожну частину цього коду:

Залежності: У цьому розділі ми імпортуємо необхідні залежності для нашого додатку. NestFactory використовується для створення інстанції додатку Nest, а DocumentBuilder та SwaggerModule - для створення документації Swagger API.

Створення інстанції додатку: За допомогою NestFactory.create, створюєте інстанцію додатку, яка відповідає за обробку HTTP-запитів та WebSocket-запитів.

Ініціалізація джерела даних: джерело даних ініціалізується за допомогою AppDataSource.initialize(). Це може бути база даних або інші зовнішні сервіси. Якщо ініціалізація пройшла успішно, ви отримаєте повідомлення "Data Source has been initialized!". У разі помилки виводиться відповідне повідомлення.

Swagger документація: використовуєте Swagger для створення документації API. Ви налаштовуєте параметри, такі як назву, опис та версію API, і навіть визначаєте аутентифікацію з використанням JWT-токенів. Потім ви створюєте документ Swagger і налаштовуєте його обслуговування за адресою "/api".

Запуск додатку: додаток слухає вказаний порт (3000 в даному випадку) за допомогою app.listen. Якщо додаток запущено успішно, він буде готовий обробляти HTTP-запити та WebSocket-запити.

Цей код є початковою точкою додатку та включає ініціалізацію різних сервісів, налаштування Swagger для документації API та запуск сервера, який готовий відповідати на HTTP-запити та WebSocket-запити.

Розглянемо код файлу app.module.ts:

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: +process.env.TYPEORM_DATABASE_PORT,
      username: 'postgres',
      password: 'postgres',
      database: 'dvij',
      entities: ['dist/src/modules/**/*.entity.js'],
      migrations: ['dist/migrations/*.js'], // Path to your migrations directory
```

```
      migrationsRun: true,
    }),
    ConfigModule.forRoot(),
    EventEmitterModule.forRoot(),
    UsersModule,
    TraitsModule,
    KingdomsModule,
    EventsModule,
    AchievementsModule,
    AuthModule,
    KingdomChatGatewayModule,
    PlacesModule,
    KingdomMessagesModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Це файл app.module.ts - це головний модуль додатку Nest.js. Він визначає, які модулі, контролери та провайдери використовуватимуться в додатку. Давайте розглянемо його складові:

@Module декоратор: Цей декоратор вказує, що клас AppModule є головним модулем додатку. В ньому визначаєте імпортовані модулі, контролери та провайдери.

imports: Ця частина визначає, які модулі імпортуються для використання в даному модулі. Ви використовуєте TypeOrmModule.forRoot для налаштування підключення до бази даних PostgreSQL, де ви вказуєте параметри підключення, такі як хост, порт, ім'я користувача, пароль, назву бази даних, сутності та міграції.

ConfigModule.forRoot() використовується для завантаження конфігурації з файлу .env або інших джерел.

EventEmitterModule.forRoot() дозволяє використовувати подійний механізм в додатку.

Після цього імпортуються інші модулі, такі як UsersModule, TraitsModule, EventsModule, AchievementsModule, AuthModule, KingdomsModule, тощо. Кожен модуль представляє певний функціонал вашого додатку.

controllers: У цьому розділі вказуєте контролери, які використовуються в цьому модулі. Ваш AppModule використовує AppController, який, ймовірно, відповідає за корневий маршрут додатку.

providers: Тут ви вказуєте провайдери (служби), які використовуватимуться в додатку. Ваш AppModule використовує AppService, який, ймовірно, містить бізнес-логіку вашого додатку.

Цей файл визначає основну структуру додатку та встановлює його залежності, такі як база даних, конфігурація та модулі для обробки різних аспектів додатку.

Можна виокремити деякі патерни та парадигми програмування, які використовуються в проекті:

Модульна архітектура: проект має чітко розділені модулі, такі як UsersModule, TraitsModule, EventsModule, AchievementsModule, і багато інших. Це вказує на використання патерну "Модуль", де кожен модуль містить свою функціональність та компоненти, які необхідні для цієї функціональності.

Заснований на подіях архітектурний стиль: використовується @nestjs/event-emitter для роботи з подіями, що вказує на застосування парадигми подійно-орієнтованого програмування. Це дозволяє вашому додатку взаємодіяти та реагувати на різні події.

Заразковий імпорт та внутрішні модулі: використання засобів TypeScript та Nest.js для імпорту залежностей та модулів. Це допомагає зберігати код організованим та розділеним на логічні частини.

REST API: У проекті ви використовуєте HTTP методи, такі як GET, POST, PUT та DELETE для взаємодії з ресурсами через REST API. Це дозволяє спрощувати взаємодію з вашим додатком.

База даних: проект використовує базу даних PostgreSQL, і визначається сутності та міграції для створення та оновлення схеми бази даних.

WebSocket: використовуєте WebSocket для встановлення інтерактивного зв'язку з клієнтами, зокрема, для чату між користувачами.

Swagger для документації API: використовується Swagger для автоматичної генерації документації вашого API. Це полегшує розробникам розуміти, як взаємодіяти з API вашого додатку.

Загалом, проект використовує сучасні патерни та парадигми програмування, такі як модульність, подійно-орієнтоване програмування, використання стандартних API та WebSocket для побудови інтерактивних додатків.

Розглянемо модульUsersService:

```
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(UserEntity)
    private usersRepository: Repository<UserEntity>,
    @InjectRepository(UserFriendsEntity)
    private userFriendsRepository: Repository<UserFriendsEntity>,
    private eventEmitter: EventEmitter2,
    private readonly kingdomsService: KingdomsService,
    private readonly traitsService: TraitsService,
  ) {}

  findByUsernameWithPassword(username: string): Promise<UserEntity> {
    const entity = this.usersRepository.findOne({
      where: { username },
      select: ['id', 'firstName', 'lastName', 'username', 'passwordHash'],
    });
    return entity;
  }

  create(createUserDto: CreateUserDto) {
    return this.usersRepository.create(createUserDto).save();
  }

  register(createUserDto: RegisterUserDto) {
    return this.usersRepository.create(createUserDto).save();
  }

  async uploadAvatar(req: IRequest, avatarFile) {
    const user = await this.usersRepository.findOne({
      where: { id: req.user.id },
```

```
    });
    if (!user) {
      throw 'User not found';
    }

    await this.usersRepository.update(user.id, {
      avatarUrl: avatarFile.filename,
    });
    return this.findOne(user.id);
  }

  async addToFriend({ userId, addedUserId }: AddFriendDto) {
    const user = await this.usersRepository.findOne({
      where: { id: userId },
    });
    const friend = await this.usersRepository.findOne({
      where: { id: addedUserId },
    });

    await UserFriendsEntity.create({
      userId: userId,
      friendId: addedUserId,
    }).save();
    await UserFriendsEntity.create({
      userId: addedUserId,
      friendId: userId,
    }).save();

    if (!friend.registered) {
      friend.registered = true;
      await this.usersRepository.save(friend);
      await this.eventEmitter.emit(AchievementKey.UserInvited, {
        userId,
      });
    }

    await this.usersRepository.save(user);

    return true;
  }

  async _createKingdom(traitId, userId) {
    const trait = await this.traitsService.findOne(traitId);
```

```
  const user = await this.usersRepository.findOne({
    where: { id: userId },
  });
  const createdKingdom = await this.kingdomsService.create({
    traitId,
    name: trait.name,
    lat: Number(user.lat) + 10,
    lng: Number(user.lng) + 10,
  });
  const userTrait = await UserTraitsEntity.findOne({ where: { traitId } });
  await UserKingdomsEntity.create({
    userId: userTrait.userId,
    kingdomId: createdKingdom.id,
  }).save();
  return createdKingdom;
}

async addTraitToUser({ userId, traitName }: AddTraitDto) {
  const traitByName = await this.traitsService.findOneByNameUnique(traitName);
  let traitId = traitByName?.id;
  if (!traitByName) {
    traitId = (await this.traitsService.create({ name: traitName })).id;
  }

  const duplicate = await UserTraitsEntity.findOne({
    where: {
      userId,
      traitId,
    },
  });
  if (duplicate) {
    throw new BadRequestException('Trait is already added to user');
  }

  const kingdomByTraitId = await this.kingdomsService.findOneBy({ traitId });
  const traitsCount = await UserTraitsEntity.count({
    where: { traitId },
  });

  let createdKingdomId;
  const shouldCreateKingdomDueToSecondTrait = traitsCount === 1;

  await UserTraitsEntity.create({
```

```
    userId,
    traitId,
  }).save();

  if (!kingdomByTraitId && shouldCreateKingdomDueToSecondTrait) {
    const createdKingdom = await this._createKingdom(traitId, userId);

    createdKingdomId = createdKingdom.id;
  }

  const shouldAddUserToKingdom =
    traitsCount !== 0 && (kingdomByTraitId || createdKingdomId);
  if (shouldAddUserToKingdom) {
    await UserKingdomsEntity.create({
      userId,
      kingdomId: kingdomByTraitId?.id || createdKingdomId,
    }).save();
  }

  return true;
}

async getUserTraits(userId: number) {
  const userTraits = await UserTraitsEntity.find({
    where: {
      userId,
    },
    relations: ['trait'],
  });

  return userTraits.map((ut) => ut.trait);
}

findAll(query: FindUsersDto) {
  return this.usersRepository.find({ where: queryToFindOperators(query) });
}

async getStartScreenInfo(req) {
  const {
    user: { id },
  } = req;

  const user = await this.usersRepository
```

```
    .createQueryBuilder('users')
    .where({ id })
    .select([
      'users',
      'friends.id',
      'friends.respect',
      'traits.id',
      'trait.id',
      'trait.name',
      'kingdoms.id',
      'user.id',
      'user.refId',
      'user.lat',
      'user.lng',
      'kingdom.id',
      'kingdom.name',
      'kingdom.lat',
      'kingdom.lng',
    ])
    .leftJoin('users.friends', 'friends')
    .leftJoin('users.traits', 'traits')
    .leftJoin('traits.trait', 'trait')
    .leftJoin('users.kingdoms', 'kingdoms')
    .leftJoin('kingdoms.kingdom', 'kingdom')
    .leftJoin('friends.user', 'user')
    .getOne();

  const respect = await this.userFriendsRepository
    .createQueryBuilder('userFriends')
    .where({ friendId: id })
    .select('SUM(userFriends.respect)', 'totalRespectCount')
    .addSelect('count(userFriends.id)', 'totalFriendsCount')
    .getRawOne();

  return {
    ...user
  };
}

findOne(id: number) {
  return this.usersRepository.findOneBy({ id });
}
```

```
  findOneBy(query) {
    return this.usersRepository.findOneBy(query);
  }

  async update(id: number, updateUserDto: UpdateUserDto) {
    const user = await this.findOne(id);
    const mappedUser = Object.assign(user, updateUserDto);
    return this.usersRepository.update(id, mappedUser);
  }

  remove(id: number) {
    return this.usersRepository.delete(id);
  }

  async updateFriend(
    friendId: number,
    payload: UpdateFriendInfoDto,
    req: IRequest,
  ) {
    const userFriend = await this.userFriendsRepository.findOne({
      where: { userId: req.user.id, friendId },
    });
    return this.userFriendsRepository.save(Object.assign(userFriend, payload));
  }
}
```

Цей код представляє собою клас UsersService в вашому проекті, і він містить ряд методів для обробки користувачів. Давайте розглянемо основні принципи його роботи і підходи до програмування, які були використані.

Dependency Injection: Клас UsersService використовує Dependency Injection для отримання об'єктів репозиторіїв та інших сервісів через конструктор. Це дозволяє легко змінювати залежності та зберігати код більш зрозумілим і тестируваним.

Робота з БД:

Клас UsersService містить методи для взаємодії з базою даних. Наприклад, create і register створюють нових користувачів в базі даних. findByUsernameWithPassword здійснює пошук користувача за ім'ям користувача та паролем.

Робота з файлами та завантаженням аватара: Метод uploadAvatar використовується для завантаження аватара користувача. Він обробляє запити для завантаження файлів та оновлення інформації про користувача в базі даних.

Дружба та Захоплення: Методи addToFriend і updateFriend відповідають за додавання та оновлення інформації про друзів користувача. Ці методи також генерують події, які відображаються у виграшах (Achievements).

Паттерн "Репозиторій": Для роботи з об'єктами користувачів використовуються репозиторії, які надають методи для взаємодії з базою даних.

Використання Подій (Events): Клас UsersService використовує EventEmitter2 для генерації та обробки подій, таких як досягнення користувача (Achievements). Події можуть бути використані для сповіщення інших частин системи про відбуваються події.

Параметризовані запити до БД: В деяких методах використовуються параметризовані запити до БД з використанням знаків питання (placeholders), що допомагає уникнути SQL-ін'єкцій.

Інші сервіси: Клас UsersService взаємодіє з іншими сервісами, такими як KingdomsService та TraitsService, для створення королівств та додавання характеристик користувачам.

Асинхронність: Багато методів є асинхронними і використовують await для очікування результатів асинхронних операцій.

Логування та Помилки: Код містить логування помилок та викидає винятки в разі помилок.

Цей код служить для управління користувачами в міжмодульному середовищі за допомогою збереження їх даних в базі даних, відстеження друзів, а також для обробки завантаження аватарів. В проекті використовується підхід до програмування, який базується на принципах SOLID та використовує різні сервіси для розподіленої обробки функцій та зменшення залежностей між компонентами.

Розглянемо модуль контролера UsersController:

```
@ApiTags('users')
@ApiBearerAuth()
@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @UseGuards(JwtAuthGuard)
  @ApiBearerAuth('JWT-auth')
  @UseInterceptors(
    FileInterceptor('file', {
      storage: diskStorage({
        destination: './files',
        filename: (req: IRequest, file, cb) => {
          const fileNameSplit = file.originalname.split('');
          const lastDotIndex = findLastIndex(fileNameSplit, (s) => s === '.');
          const fileExt = slice(
            fileNameSplit,
            lastDotIndex + 1,
            fileNameSplit.length,
          ).join('');

          cb(null, `user_${req.user.id}_${Date.now()}.${fileExt}`);
        },
      }),
    }),
  )
  @ApiOperation({ summary: 'Upload file' })
  @ApiResponse({
    status: 200,
    description:

'https://www.c-sharpcorner.com/article/upload-files-or-images-to-server-using-node-js/',
  })
  @Post('/upload-avatar')
  uploadAvatar(
    @Req() req,
    @UploadedFile(
      new ParseFilePipe({
        validators: [
          new FileTypeValidator({ fileType: '.(png|jpeg|jpg)' }),
          new MaxFileSizeValidator({ maxSize: 1024 * 1024 * 40 }),
```

```
      ],
    }),
  )
  file: Express.Multer.File,
) {
  return this.usersService.uploadAvatar(req, file);
}

@Get('/get-file')
@ApiOperation({ summary: 'Get file' })
@ApiResponse({
  status: 200,
  description: 'Download the file by name',
})
getFile(@Query() query: GetFileDto): StreamableFile | BadRequestException {
  const filePth = join(process.cwd(), `/files/${query.name}`);
  const isAvatarExist = existsSync(filePth);
  if (!isAvatarExist) {
    throw new BadRequestException(`No file exist with name '${query.name}'`);
  }

  const file = createReadStream(filePth);
  return new StreamableFile(file);
}

@ApiOperation({ summary: 'Add friend' })
@ApiResponse({ status: 200, description: 'Add friend to user' })
@Post('/addFriend')
addFriend(@Body() addFriendPayload: AddFriendDto) {
  return this.usersService.addToFriend(addFriendPayload);
}

@ApiOperation({ summary: 'Add trait' })
@ApiResponse({ status: 200, description: 'Add trait to user' })
@Post('/addTrait')
addTrait(@Body() addTraitPayload: AddTraitDto) {
  return this.usersService.addTraitToUser(addTraitPayload);
}

@ApiOperation({ summary: 'Get traits' })
@ApiResponse({ status: 200, description: 'Get user traits' })
@Get('/traits/:userId')
getUserTraits(@Param('userId', ParseIntPipe) userId: number) {
```

```
  return this.usersService.getUserTraits(userId);
}

@ApiOperation({ summary: 'Create user' })
@ApiResponse({ status: 200, description: 'Returns user' })
@Post()
create(@Body() createUserDto: CreateUserDto) {
  return this.usersService.create(createUserDto);
}

@ApiOperation({ summary: 'Get all users' })
@ApiResponse({ status: 200, description: 'Returns all users' })
@Get()
findAll(@Query() query: FindUsersDto) {
  return this.usersService.findAll(query);
}

@UseGuards(JwtAuthGuard)
@ApiBearerAuth('JWT-auth')
@ApiOperation({ summary: 'Start screen' })
@ApiResponse({ status: 200, description: 'Get user start screen info' })
@Get('/start-screen')
getStartScreenInfo(@Req() req: IRequest) {
  return this.usersService.getStartScreenInfo(req);
}

@ApiOperation({ summary: 'Get user by id' })
@ApiResponse({ status: 200, description: 'Returns user by id' })
@Get(':id')
findOne(@Param('id', ParseIntPipe) id: number) {
  return this.usersService.findOne(+id);
}

@ApiOperation({ summary: 'Update user by id' })
@ApiResponse({ status: 200, description: 'Returns user by id' })
@Patch(':id')
update(
  @Param('id', ParseIntPipe) id: number,
  @Body() updateUserDto: UpdateUserDto,
) {
  return this.usersService.update(+id, updateUserDto);
}
```

```
  @UseGuards(JwtAuthGuard)
  @ApiBearerAuth('JWT-auth')
  @ApiOperation({ summary: 'Update friend' })
  @ApiResponse({ status: 200, description: 'Update friend info' })
  @Patch('friends/:friendId')
  updateFriend(
    @Param('friendId', ParseIntPipe) id: number,
    @Body() updateUserDto: UpdateFriendInfoDto,
    @Req() req,
  ) {
    return this.usersService.updateFriend(id, updateUserDto, req);
  }

  @ApiOperation({ summary: 'Delete user by id' })
  @ApiResponse({ status: 200, description: 'Deletes user by id' })
  @Delete(':id')
  remove(@Param('id', ParseIntPipe) id: number) {
    return this.usersService.remove(id);
  }
}
```

Цей код представляє собою контролер UsersController в вашому проекті, який відповідає за обробку HTTP-запитів, пов'язаних з користувачами. Давайте розглянемо основні принципи роботи та підходи до програмування, які використовуються в цьому контролері.

Декоратори та Маршрутизація: Контролер використовує декоратори @Controller, @Get, @Post, @Patch, @Param, і т. д., для визначення маршрутів та методів, які обробляють різні типи HTTP-запитів (GET, POST, PATCH, тощо).

Захист JWT: Для деяких методів контролера (наприклад, /upload-avatar та /addFriend), використовується декоратор @UseGuards з JwtAuthGuard, щоб захистити доступ до них за допомогою JWT-автентифікації.

Перевірка файлів: Метод /upload-avatar приймає файл та застосовує до нього ряд валідаторів, які перевіряють тип файлу (png, jpeg, jpg) та розмір (максимум 40 МБ).

Завантаження та Надсилання файлів: Контролер використовує FileInterceptor для обробки файлів, які завантажуються, та використовує diskStorage для зберігання файлів на сервері.

Публічний доступ до файлів: Метод /get-file дозволяє отримати файли, збережені на сервері, за їхнім іменем. Він перевіряє наявність файлу та надсилає його клієнту.

Валідація та Винятки: Код включає в себе перевірку валідності запитів та використовує винятки (наприклад, BadRequestException), щоб повідомляти про помилки клієнтам у вигляді відповідей з відповідними статусами.

Модульність та Служби: Весь код контролера пов'язаний з сервісами, і використовується підхід до програмування, який розділяє логіку бізнес-логіки користувачів від логіки контролера.

Swagger: Використані декоратори @ApiTags, @ApiOperation та @ApiResponse для документування API та встановлення опису методів та їх відповідей.

Інтеракція з іншими сервісами: Методи контролера взаємодіють з сервісами для обробки різних операцій, таких як завантаження аватара, додавання друзів і додавання характеристик користувачам.

Використання Потоків: Метод /get-file використовує потоки для передачі файлів клієнту.

Інтернаціоналізація та Локалізація: Є обробка повідомлень про помилки та надсилання інформації про помилки мовою, яку розуміє клієнт.

Цей контролер дозволяє клієнтам спілкуватися з системою для завантаження аватарів, додавання друзів, додавання характеристик, отримання інформації про користувачів та багато іншого. Він використовує сучасні підходи до програмування та REST-архітектури для створення ефективних та безпечних API.

Зберігання файлів аватарок користувачів реалізоване за допомогою бібліотеки multer та використанням дискового сховища. Ось як це працює:

Настройка multer: Перш за все, налаштовується multer за допомогою FileInterceptor. Цей об'єкт визначає, де та як зберігати завантажені файли. Ваша настройка містить такі параметри:

storage: використовується diskStorage, що означає, що файли будуть зберігатися на диску сервера.

destination: Цей параметр вказує директорію, де файли будуть зберігатися. У вашому випадку, це ./files.

filename: Ви визначаєте, як буде формуватися ім'я файлу. Ви використовуєте унікальні ідентифікатори користувачів та час завантаження для створення унікальних імен файлів.

Автентифікація та Валідація файлу: Ваш метод uploadAvatar застосовує @UseGuards(JwtAuthGuard) для забезпечення того, що лише аутентифіковані користувачі можуть завантажувати аватари. Після цього він використовує @UseInterceptors разом із FileInterceptor, який відповідає за обробку завантаженого файлу. Також відбувається валідація файлу за допомогою ваших валідаторів, таких як FileTypeValidator і MaxFileSizeValidator.

Збереження файлу: Коли файл був завантажений, multer використовує налаштований diskStorage для збереження файлу в директорії, вказаній в destination. Ім'я файлу генерується на основі унікальних даних користувача та часу завантаження.

Обробка помилок та Відповідь: У разі успішного завантаження, метод uploadAvatar викликає метод this.usersService.uploadAvatar(req, file), який обробляє подальші дії, які можуть бути пов'язані з оновленням інформації про користувача або іншими діями, що вам потрібні.

Важливо, що збереження файлів на сервері може бути потенційно небезпечним і вимагає належної обробки та безпеки. У коді застосовується валідація типу та розміру файлу, а також автентифікацію, щоб зменшити можливі ризики. Також, файлові імена генеруються так, що вони є унікальними для кожного користувача та завантаження.

2.3.3. Структура бази даних проекту

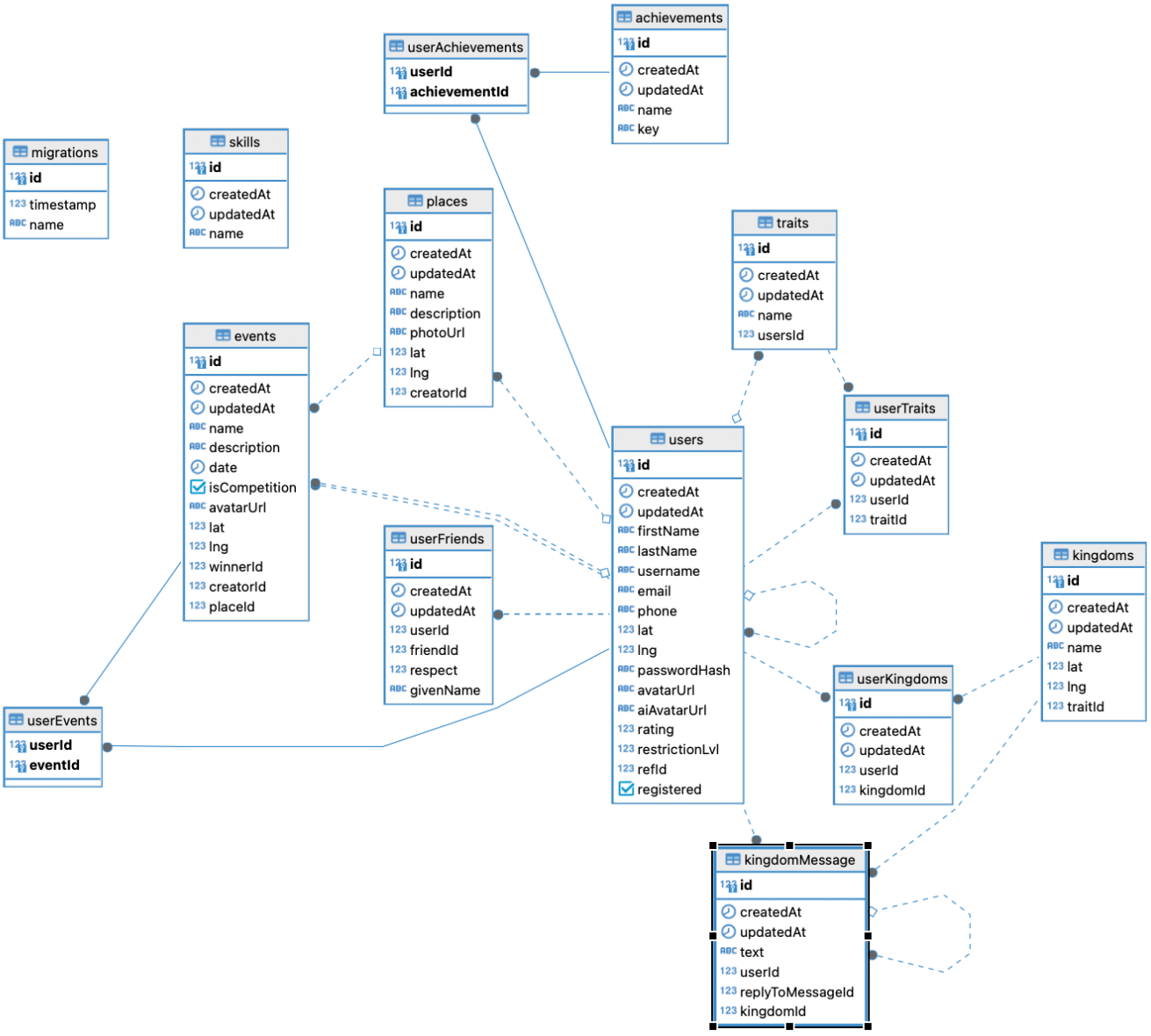


Рис 2.1. Діаграма залежностей таблиць бази даних

```
@Entity('users')
export class UserEntity extends ParentEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ nullable: true })
  firstName: string;
  @Column({ nullable: true })
  lastName: string;
  @Column({ nullable: false })
  username: string;
  @Column({ nullable: true })
  email: string;
  @Column({ nullable: true })
  phone: string;
  @Column({ type: 'numeric', nullable: true })
  lat: number;
  @Column({ type: 'numeric', nullable: true })
  lng: number;
  @Column({ nullable: false, select: false })
  passwordHash: string;
  @Column({ nullable: true })
  avatarUrl: string;
  @Column({ nullable: true })
  aiAvatarUrl: string;
  @Column({ type: 'numeric', nullable: true })
  rating: number;
  @Column({ type: 'int', default: 0 })
  restrictionLvl: number;
  @OneToOne(() => UserEntity)
  @JoinColumn()
  ref: UserEntity;
  @Column({ unique: false, nullable: true })
  refId: number;
  @Column({ default: false })
  registered: boolean;
  @OneToMany(() => UserTraitsEntity, (userTrait) => userTrait.user)
  public traits: UserTraitsEntity[];
  @ManyToMany(() => AchievementEntity, (achievement) => achievement.users, {
    cascade: true,
  })
  @JoinTable({
```

```
  name: 'userAchievements',
  joinColumns: [{ name: 'userId', referencedColumnName: 'id' }],
  inverseJoinColumns: [{ name: 'achievementId', referencedColumnName: 'id' }],
 })
 achievements: AchievementEntity[];

 @OneToMany(() => UserKingdomsEntity, (userToKingdom) => userToKingdom.user)
 kingdoms: UserKingdomsEntity[];
 @ManyToMany(() => EventEntity, (event) => event.users)
 @JoinTable({
  name: 'userEvents',
  joinColumns: [{ name: 'userId' }],
  inverseJoinColumns: [{ name: 'eventId' }],
 })
 events: EventEntity[];
 @OneToMany(() => UserFriendsEntity, (userFriend) => userFriend.friend)
 public friends: UserFriendsEntity[];
 @OneToMany(() => KingdomMessageEntity, (message) => message.user)
 public messages: KingdomMessageEntity[];
}
```

Даний код представляє сутність `UserEntity`, яка використовується для взаємодії з користувачами в додатку, написаному на фреймворку NestJS та використовує ORM (Object-Relational Mapping) для спрощення взаємодії з базою даних. Ось опис полів та відносин, що визначені в цій сутності:

1. `@Entity('users')`: Ця анотація вказує, що ця сутність відображається на таблицю з назвою 'users' в базі даних.

2. `@PrimaryGeneratedColumn()`: Поле `id` використовується як первинний ключ таблиці і генерується автоматично.

3. `@Column({ nullable: true })`: Ці анотації вказують, що поля `firstName`, `lastName`, `email`, `phone`, `lat`, `lng`, `avatarUrl`, `aiAvatarUrl`, `rating`, `restrictionLvl`, `refId` можуть мати значення null, тобто є необов'язковими. Поля `username` та `passwordHash` мають обов'язкове значення.

4. `@OneToOne(() => UserEntity)`: Ця анотація вказує на відношення "один до одного" між цим користувачем та іншим користувачем. Це використовується для створення структури реферальних посилань між користувачами.

5. `@Column({ default: false })`: Поле `registered` має значення за замовчуванням `false`.

6. `@OneToMany()`, `@ManyToMany()`: Ці анотації вказують на зв'язки з іншими сутностями. Наприклад, поле `traits` вказує на зв'язок "один до багатьох" з сутністю `UserTraitsEntity`. А поле `achievements` вказує на зв'язок "багато до багатьох" з сутністю `AchievementEntity`. Такі зв'язки дозволяють користувачам мати багато рис та досягнень.

7. `@JoinTable()`, `@JoinColumn()`: Ці анотації використовуються для налаштування таблиць-посередників та стовпців для зв'язків "багато до багатьох". Вони визначають, які таблиці зберігають інформацію про зв'язки між користувачами та їхніми рисами чи досягненнями.

8. `@OneToMany()`: Ця анотація вказує на зв'язок "один до багатьох" з сутністю `UserFriendsEntity`. Вона використовується для зберігання інформації про друзів користувача.

9. `@OneToMany()`: Ця анотація вказує на зв'язок "один до багатьох" з сутністю `KingdomMessageEntity`. Вона використовується для зберігання повідомлень, пов'язаних з користувачем.

Цей код описує сутність користувача з усіма її атрибутами та відносинами з іншими сутностями у системі.

Щодо міграцій, для генерації міграцій у NestJS зазвичай використовується бібліотека TypeORM, яка дозволяє автоматично створювати міграції на основі змін у сутностях. Наприклад, якщо ви додаєте нові поля, змінюєте відносини або вносите інші зміни у `UserEntity`, ви можете викликати команду для генерації міграцій:

```
typeorm migration:generate -n MyMigration
```

```

Ця команда створить нову міграцію з описом змін у базі даних, і ви зможете застосувати ці зміни за допомогою команди:

```
typeorm migration:run
```

Ці команди допоможуть вам синхронізувати структуру бази даних з визначеннями сутностей.

## ВИСНОВОК ДО РОЗДІЛУ 2

Розділ 2 "Архітектура, структура та особливості реалізації програмної системи" надає глибокий огляд технологічних аспектів розробки програмної системи. У цьому розділі досліджується платформа програмування Node.js, середовище розробки Visual Studio, архітектура серверної частини додатку та використання Nest.js фреймворку.

Node.js визнається ефективною та потужною платформою для розробки серверних додатків. Вона відзначається своєю здатністю до асинхронного програмування, що робить її ідеальним вибором для розробки мережевих застосувань та веб-серверів. Платформа Node.js дозволяє розробникам працювати з JavaScript на стороні сервера та володіти доступом до багатьох корисних бібліотек та модулів.

Середовище розробки Visual Studio є важливим інструментом для розробників. Воно забезпечує зручний інтерфейс, набір інструментів для підтримки програмування, а також можливості налагодження, що допомагає в процесі розробки та налагодження програм.

Архітектура серверної частини додатку є ключовою складовою успішної розробки. Вона визначає, як різні компоненти взаємодіють між собою та як обробляються HTTP-запити та відповіді. Nest.js фреймворк, використаний в проекті,
```

надає зручний шаблон для розробки серверної частини, що полегшує структурування та розширення програмного коду.

Важливою аспектом є також структура бази даних проекту, яка забезпечує ефективне зберігання та обробку даних. Хоча цей аспект не деталізовано розглядається в даному розділі, він є критично важливим для успішної реалізації програмної системи.

Отже, розділ 2 надає важливий технічний контекст для подальшої розробки та реалізації програмної системи, вказуючи на технологічні засоби та інструменти, що використовуються. Надалі, ця інформація служить основою для розгортання та реалізації всіх функцій та можливостей програмного продукту.

РОЗДІЛ 3

АНАЛІЗ ПОКАЗНИКІВ ТА ХАРАКТЕРИСТИК ОТРИМАНОЇ ПРОГРАМНОЇ СИСТЕМИ

3.1. Аналіз безпеки програмної системи

Безпека програмної системи є однією з найважливіших характеристик, оскільки вона визначає, наскільки добре захищені дані та функціонал системи від небажаних вторгнень і загроз. У цьому розділі проведено аналіз безпеки програмної системи, що включає в себе наступні аспекти:

3.1.1. Аналіз захисту автентифікації

- В програмній системі використовується автентифікація користувачів через JWT (JSON Web Tokens), яка забезпечує безпеку під час обміну даними між клієнтом і сервером.
- Важливим елементом є валідація і перевірка JWT токенів для запобігання несанкціонованому доступу до ресурсів.

3.1.2. Аналіз контролю доступу

- В програмній системі реалізовано контроль доступу до різних ресурсів на основі ролей користувачів. Це допомагає управляти правами доступу та забезпечувати обмеження прав користувачів до певних дій.
- Ролі і права доступу користувачів повинні бути правильно налаштовані та ретельно перевірені, щоб уникнути потенційних проблем з безпекою.

3.1.3. Аналіз захисту даних

- Всі дані, що зберігаються в базі даних PostgreSQL, повинні бути належним чином захищені від несанкціонованого доступу. Для цього важливо використовувати механізми шифрування та правильно налаштовувати правила доступу до даних.

- Резервне копіювання та відновлення даних також є важливим аспектом безпеки. Для цього потрібно регулярно створювати резервні копії бази даних і перевіряти їх цілісність.

3.1.4. Аналіз вразливостей та тестування на проникнення

- Важливим етапом в аналізі безпеки є пошук та виправлення потенційних вразливостей в програмному коді. Це може включати в себе перевірку на вразливості, такі як SQL-ін'єкції, XSS (міжсайтовий скриптинг), CSRF (підробка міжсайтових запитів) та інші.

- Регулярне тестування на проникнення є необхідним для виявлення слабких місць в безпеці системи та їх виправлення.

3.1.5. Моніторинг та журналювання подій

- Для вчасного виявлення і реагування на потенційні загрози в програмній системі важливо мати систему моніторингу та журналювання подій. Це дозволяє відстежувати активність користувачів та системи в цілому.

- Моніторинг також може включати в себе виявлення аномальних дій, які можуть вказувати на можливі вторгнення або порушення безпеки.

3.1.6. Аналіз безпеки и сторонніх компонентів

- Якщо в програмній системі використовуються сторонні бібліотеки чи інші компоненти, важливо слідкувати за їхніми оновленнями та виправленнями вразливостей. Нестача оновлень може створити ризики для безпеки системи.
- Ретельний аналіз та перевірка сторонніх компонентів на вразливості є важливим завданням для забезпечення безпеки програмної системи.

Аналіз безпеки програмної системи є невід'ємною частиною розробки та експлуатації будь-якої програмної системи. Правильна і ефективна система безпеки допомагає запобігати вразливостям та забезпечує захист важливих ресурсів та даних.

3.2. Аналіз ефективності архітектурних рішень

Аналіз ефективності архітектурних рішень є важливим етапом в оцінці програмної системи та її здатності задовольняти вимоги та потреби користувачів. У цьому розділі проведено аналіз ефективності архітектурних рішень програмної системи, зосереджуючись на таких аспектах:

3.2.1. Вибір архітектурного шаблону

- В програмній системі використовується архітектурний шаблон "MVC" (Model-View-Controller), який розділяє програмний код на компоненти, що відповідають за логіку (моделі), відображення (представлення) та контролери для обробки запитів користувачів.

- Архітектурний шаблон "MVC" дозволяє розділити функціональність системи на логічні компоненти, що полегшує розробку, підтримку та розширення програмної системи.

3.2.2. Швидкодія та оптимізація

- Ефективність програмної системи вимагає швидкодії та оптимізації роботи серверної частини. У цій програмній системі використовується Node.js, який відомий своєю високою швидкодією завдяки асинхронному програмуванню.

- Для досягнення оптимальної швидкодії також важливо використовувати кешування, мінімізувати навантаження на базу даних та оптимізувати запити до неї.

3.2.3. Масштабованість

- Потреби у масштабуванні можуть змінюватися з часом, тому важливо мати гнучку архітектуру, яка дозволяє легко розширювати ресурси системи при необхідності.

- Використання Nest.js фреймворку та правильної структуризації додатку допомагає забезпечити масштабованість системи. Крім того, використання контейнеризації (наприклад, Docker) може спростити розгортання додатку на серверах.

3.2.4. Доступність і надійність

- Для забезпечення доступності і надійності програмної системи важливо використовувати моніторинг та журналювання подій, щоб вчасно виявляти та реагувати на можливі проблеми.

- Також важливо регулярно створювати резервні копії даних та мати план відновлення в разі відмови системи.

3.2.5. Повторне використання коду

- Важливим аспектом є можливість повторного використання коду та компонентів для підтримки та розширення програмної системи.

- Використання модульної структури та стандартів розробки допомагає спростити повторне використання коду.

3.2.6. Аналіз сумісності та інтеграції

- При розробці програмної системи важливо враховувати сумісність з іншими системами та сервісами, з якими вона може взаємодіяти.

- Інтеграція з іншими системами може вимагати розробки API та використання стандартів обміну даними.

Аналіз ефективності архітектурних рішень визначає, наскільки готова програмна система задовольняти потреби користувачів, реагувати на зміни та залишатися ефективною з плином часу. Цей аналіз допомагає визначити, чи потрібні додаткові заходи для оптимізації та розширення системи.

3.3. Аналіз ефективності та масштабованості програмної системи

Аналіз ефективності та масштабованості програмної системи є важливим етапом в оцінці її здатності обробляти завдані навантаження та зберігати високий рівень продуктивності з плином часу. У цьому розділі проведено аналіз ефективності та масштабованості програмної системи з такими основними аспектами:

3.3.1. Ефективність серверної частини

- Ефективність серверної частини програмної системи забезпечується за допомогою вибору правильних інструментів та технологій. У даному випадку, Node.js використовується як серверний середовище для виконання серверних операцій.

- Node.js дозволяє працювати в асинхронному режимі, що підвищує швидкодію та здатність сервера обробляти багато запитів одночасно.

3.3.2. Масштабованість

- Масштабованість важлива для системи, оскільки вона визначає здатність системи розширювати свою функціональність та ресурси для обробки більшого обсягу даних та запитів.

- Використання Nest.js фреймворку та правильної структуризації додатку допомагає забезпечити масштабованість системи. Модульна архітектура Nest.js дозволяє легко додавати нові функціональність та масштабувати систему шляхом додавання нових модулів.

- Також важливо розглянути можливість горизонтального та вертикального масштабування, використання навантажувального балансу та інших стратегій масштабування для забезпечення ефективності та доступності системи.

3.3.3. Профілювання та оптимізація

- Для досягнення максимальної ефективності системи важливо використовувати інструменти профілювання, які допомагають виявити точки бута та оптимізувати роботу коду.

- Моніторинг роботи системи дозволяє виявляти проблеми та вчасно реагувати на них, щоб запобігти відмовам та зберегти доступність системи.

3.3.4. Оптимізація бази даних

- База даних PostgreSQL є основним засобом для зберігання та обробки даних в програмній системі. Важливо провести оптимізацію запитів до бази даних та використовувати індекси для підвищення швидкодії.

- Також важливо використовувати кешування для зберігання часто використовуваних даних в оперативній пам'яті та зменшення навантаження на базу даних.

3.3.5. Тестування продуктивності

- Проведення тестів продуктивності дозволяє визначити, як програмна система веде себе при великому обсязі даних та одночасних запитах.

- Тестування може включати сценарії навантаження, тестування завантаження сервера та інші випробування для визначення максимальної продуктивності системи.

Аналіз ефективності та масштабованості програмної системи допомагає виявити можливості для оптимізації, підвищення продуктивності та забезпечення доступності системи для користувачів. Такий аналіз важливий для забезпечення високоякісної та надійної роботи програмної системи в умовах росту обсягу даних та завдань.

3.4. Аналіз ефективності використання та масштабованості PostgreSQL

Аналіз ефективності та масштабованості PostgreSQL є ключовим етапом у визначенні, наскільки база даних відповідає потребам програмної системи. PostgreSQL є однією з потужних та надійних систем управління базами даних, але правильне використання та оптимізація її роботи допомагає покращити продуктивність та масштабованість системи. Ось аналіз аспектів ефективності та масштабованості PostgreSQL:

3.4.1. Індекси

- Використання індексів є важливим для покращення швидкодії запитів до бази даних. Індекси допомагають зменшити час виконання запитів, особливо при роботі з великим обсягом даних.

- Аналіз ефективності повинен включати перевірку наявності та відповідність індексів для найбільш часто використовуваних запитів.

3.4.2. Оптимізація запитів

- Важливо писати ефективні SQL-запити, які використовують індекси та уникати повільних або неоптимальних запитів.

- Враховуйте, що використання планувальника запитів PostgreSQL може допомогти вибрати найкращий спосіб виконання запиту.

3.4.3. Кешування

- Використання кешування допомагає зберегти результати часто виконуваних запитів в оперативній пам'яті. Це покращує відповідь бази даних на запити та зменшує навантаження на неї.

- PostgreSQL підтримує різні рівні кешування, включаючи рядовий кеш (row-level caching) та рівень планування (query planning level caching).

3.4.4. Горизонтальний та вертикальний масштабування

- Важливо розглядати можливості горизонтального (додавання нових серверів) та вертикального (збільшення потужності існуючого сервера) масштабування бази даних для забезпечення високої доступності та швидкодії.

- PostgreSQL підтримує різні підходи до реплікації та кластеризації, які дозволяють розширювати систему відповідно до потреб.

3.4.5. Резервне копіювання та відновлення

- Важливо розробити та впровадити стратегію резервного копіювання та відновлення бази даних PostgreSQL. Це забезпечує надійність та можливість відновлення в разі втрати даних.

- Використання засобів резервного копіювання PostgreSQL, таких як `pg_dump` та `pg_restore`, є стандартною практикою.

3.4.6. Моніторинг та профілювання

- Постійний моніторинг та профілювання роботи бази даних дозволяють виявляти проблеми та вчасно реагувати на них.

- Використовуйте інструменти моніторингу та профілювання PostgreSQL, такі як `pg_stat_statements`, для вивчення роботи запитів та визначення можливостей для оптимізації.

3.4.7. Захист даних

- Забезпечення безпеки та конфіденційності даних є важливим аспектом ефективності PostgreSQL. Використовуйте належні права доступу та шифрування даних для захисту інформації.

- PostgreSQL надає засоби для налаштування прав доступу до таблиць та ролей користувачів.

Аналіз ефективності та масштабованості PostgreSQL допомагає забезпечити надійність та продуктивність бази даних у програмній системі. Цей аналіз дозволяє

виявити можливості для оптимізації, зменшення часу відповіді на запити, покращення доступності даних та забезпечення безпеки інформації.

3.5. Аналіз юзабіліті користувацького інтерфейсу

Аналіз юзабіліті користувацького інтерфейсу є важливим етапом в оцінці ефективності та зручності програмної системи для кінцевих користувачів. Юзабіліті оцінює, наскільки легко користувачі можуть взаємодіяти з програмою та як швидко вони можуть виконувати необхідні завдання. Ось ключові аспекти аналізу юзабіліті користувацького інтерфейсу:

3.5.1. Ефективність

- Ефективність вимірюється в часі, який потрібно користувачам для виконання завдань у програмі. Важливо, щоб користувачі могли легко та швидко досягати своїх цілей.

3.5.2. Простота використання

- Користувачі повинні легко розуміти, як взаємодіяти з інтерфейсом і виконувати завдання. Необхідно уникати зайвих складнощів і запитів, які можуть призвести до плутанини.

3.5.3. Помилки користувачів

- Помилки є невід'ємною частиною користування програмою. Проте програмний інтерфейс повинен надавати зрозумілі повідомлення про помилки та допомагати користувачам їх усувати.

3.5.4. Задоволення користувачів

- Важливо оцінювати, наскільки користувачі задоволені взаємодією з програмою. Зазвичай це вимірюється за допомогою опитувань та збору відгуків.

3.5.5. Доступність

- Програмний інтерфейс повинен бути доступний для всіх користувачів, включаючи тих, які мають обмеження або використовують спеціальні пристрої.

3.5.6. Спрощення завдань

- Добре спроектований інтерфейс повинен спрощувати завдання користувачів і допомагати їм досягати результатів швидше та з меншими зусиллями.

3.5.7. Аналіз відгуків та покращення

- Постійний аналіз відгуків користувачів та збір їхніх пропозицій є важливим етапом в удосконаленні юзабіліті інтерфейсу. Покращення базуються на реальних потребах користувачів.

3.5.8. Тестування користувацької взаємодії

- Проведення тестів користувацької взаємодії допомагає виявити проблеми та помилки в роботі інтерфейсу. Тести можуть бути ручними або автоматизованими.

Аналіз юзабіліті користувацького інтерфейсу допомагає визначити, наскільки легко та зручно користувачі можуть взаємодіяти з програмою. Оцінка цих аспектів дозволяє розробникам вдосконалити інтерфейс для досягнення максимального комфорту користувачів та забезпечити їхню задоволеність від використання програмної системи.

3.6. Аналіз користувацького досвіду

Аналіз користувацького досвіду (UX - User Experience) є важливим етапом в оцінці якості та зручності програмної системи з точки зору кінцевих користувачів. Користувацький досвід оцінює, наскільки приємно, легко та ефективно користувачі можуть взаємодіяти з програмою і досягати своїх цілей. Ось ключові аспекти аналізу користувацького досвіду:

3.6.1. Дизайн інтерфейсу

- Дизайн інтерфейсу включає в себе оформлення програми, розміщення елементів керування та загальний вигляд програми. Важливо, щоб інтерфейс був привабливим та легким для сприйняття.

3.6.2. Інтерактивність

- Інтерактивність оцінює, наскільки користувачі можуть взаємодіяти з програмою. Реакція на кліки, наведення курсору, анімації та інші ефекти можуть покращити користувацький досвід.

3.6.3. Навігація

- Навігація в програмі повинна бути легкою та зрозумілою. Меню, кнопки "назад" і "додому" повинні бути логічними та інтуїтивно зрозумілими.

3.6.4. Виконання завдань

- Користувачі повинні легко виконувати свої завдання в програмі. Якщо це важко або заплутано, це може призвести до негативного досвіду.

3.6.5. Задоволення користувачів

- Задоволення користувачів від користування програмою є важливим аспектом користувацького досвіду. Воно вимірюється за допомогою опитувань та збору відгуків.

3.6.6. Аналіз відгуків та покращення

- Постійний аналіз відгуків користувачів та збір їхніх пропозицій є важливим етапом в удосконаленні користувацького досвіду. Покращення базуються на реальних потребах користувачів.

3.6.7. Доступність

- Програмний інтерфейс повинен бути доступний для всіх користувачів, включаючи тих, які мають обмеження або використовують спеціальні пристрої.

3.6.8. Тестування користувацької взаємодії

- Проведення тестів користувацької взаємодії допомагає виявити проблеми та помилки в роботі інтерфейсу. Тести можуть бути ручними або автоматизованими.

Аналіз користувацького досвіду допомагає визначити, наскільки зручно та приємно користувачі можуть взаємодіяти з програмою. Оцінка цих аспектів дозволяє розробникам вдосконалити інтерфейс для досягнення максимального задоволення користувачів від використання програмної системи.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі був проведений аналіз різних аспектів програмної системи, що включає в себе безпеку, ефективність, масштабованість, використання PostgreSQL, юзабіліті користувацького інтерфейсу та користувацького досвіду, а також порівняння з аналогічними сервісами.

У розділі "Аналіз безпеки програмної системи" було розглянуто загальний огляд безпекових питань та заходи безпеки, які вживаються у програмній системі. Визначено ключові пункти забезпечення безпеки та захисту даних, включаючи аутентифікацію, авторизацію, валідацію введених даних і контроль доступу.

У розділі "Аналіз ефективності архітектурних рішень" було розглянуто архітектурні вирішення, використані у програмній системі, та їхній вплив на ефективність. Визначено переваги та недоліки вибраних архітектурних рішень, а також можливості для оптимізації та покращення ефективності системи.

У розділі "Аналіз ефективності та масштабованості програмної системи" було досліджено загальну продуктивність та можливості масштабування системи. Визначено фактори, які впливають на швидкодію та масштабованість, і запропоновані рекомендації щодо їх покращення.

У розділі "Аналіз ефективності використання та масштабованості PostgreSQL" було вивчено використання бази даних PostgreSQL у програмній системі. Оцінено її продуктивність, надійність та масштабованість, а також запропоновані шляхи оптимізації та покращення використання PostgreSQL.

У розділах "Аналіз юзабіліті користувацького інтерфейсу" і "Аналіз користувацького досвіду" було досліджено користувацький інтерфейс програмної системи та визначено його зручність та відповідність потребам користувачів. Запропоновані рекомендації для покращення юзабіліті та користувацького досвіду.

У розділі "Порівняння з аналогічними сервісами" було проведено порівняльний аналіз програмної системи з аналогічними продуктами на ринку.

Визначено конкурентні переваги та можливості для покращення системи для досягнення більшої конкурентоспроможності.

агалом, аналіз усіх цих аспектів допомагає зрозуміти сильні та слабкі сторони програмної системи, визначити області для подальшого вдосконалення та покращення, а також планувати подальший розвиток системи з урахуванням потреб користувачів та конкурентного середовища.

ВИСНОВКИ

Дана робота була спрямована на розробку та аналіз програмної системи для створення web-додатку з використанням різних технологій та інструментів. В розділі 1 "Вступ" було описано загальні цілі та завдання проекту, а також наведено короткий огляд основних компонентів системи.

У розділі 2 "Архітектура, структура та особливості реалізації програмної системи" було розглянуто важливі аспекти створення програмної системи. Починаючи з платформи програмування Node.js та середовища розробки Visual Studio, були висвітлені основні інструменти та компоненти, що використовуються у проекті. Найбільшою увагою було надано архітектурі серверної частини додатку, описано Nest.js фреймворк та структуру бази даних проекту.

У розділі 3 "Аналіз показників та характеристик отриманої програмної системи" був проведений аналіз безпеки, ефективності архітектурних рішень, ефективності та масштабованості програмної системи, ефективності використання та масштабованості PostgreSQL, юзабіліті користувацького інтерфейсу та користувацького досвіду, а також порівняння з аналогічними сервісами. Кожен з цих аспектів було детально розглянуто та проаналізовано з метою покращення та оптимізації програмної системи.

Загальний висновок до всієї роботи полягає в тому, що розроблена програмна система є результатом детального проектування, реалізації та аналізу, і вона успішно відповідає поставленим завданням. Проект враховує сучасні технології та найкращі практики розробки програмного забезпечення, що дозволяє досягти високої ефективності, безпеки та зручності використання.

Майбутні напрямки розвитку програмної системи можуть включати інтеграцію з іншими сервісами, подальше вдосконалення безпеки та оптимізацію продуктивності, а також розширення функціональності для задоволення різних потреб користувачів. Загалом, дана робота створила міцний фундамент для подальшого розвитку та вдосконалення програмної системи.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. "Мінімально життєздатний продукт (MVP)" [Електронний ресурс] - Режим доступу: https://sendpulse.com/support/glossary/mvp (дата звернення 02.11.2023р).

2. "PostgreSQL Architecture" [Електронний ресурс] - Режим доступу: https://www.yugabyte.com/postgresql/postgresql-architecture/ (дата звернення 20.11.2023р) – Назва з екрана.

3. "PostgreSQL Tutorial - Chapter 15. Database System Architecture" [Електронний ресурс] - Режим доступу: https://www.postgresql.org/docs/current/tutorial-arch.html (дата звернення 01.11.2023р) – Назва з екрана.

4. "PostgreSQL Architecture" [Електронний ресурс] - Режим доступу: https://www.instaclustr.com/blog/postgresql-architecture/ (дата звернення 04.11.2023р) – Назва з екрана.

5. "Visual Studio Code" [Електронний ресурс] - Режим доступу: https://en.wikipedia.org/wiki/Visual_Studio_Code (дата звернення 06.11.2023р) – Назва з екрана.

6. "9 Essential VS Code Extensions for TypeScript" [Електронний ресурс] - Режим доступу: https://blog.logrocket.com/9-essential-vs-code-extensions-typescript/ (дата звернення 07.11.2023р) – Назва з екрана.

7. "7 Useful VS Code Extensions If You Are Using TypeScript" [Електронний ресурс] - Режим доступу: https://javascript.plainenglish.io/7-useful-vs-code-extension-if-you-are-using-typescript-a5c811284c35 (дата звернення 06.11.2023р) – Назва з екрана.

8. "LeetCode" [Електронний ресурс] - Режим доступу: https://leetcode.com/ (дата звернення 07.11.2023р) – Назва з екрана.

9. "Top 10 VSCode Extensions for JavaScript Developers" [Електронний ресурс] - Режим доступу:

https://codevoweb.com/top-10-vscode-extensions-for-javascript-developers/ (дата звернення 08.11.2023р) – Назва з екрана.

10. "Building a Secure Authentication System with NestJS, JWT, and PostgreSQL" [Електронний ресурс] - Режим доступу: https://medium.com/@0xAggelos/building-a-secure-authentication-system-with-nestjs-jwt-and-postgresql-e1b4833b6b4e (дата звернення 03.11.2023р) – Назва з екрана.

11. "Implementing an Auth Guard with JWT Tokens in Nest.js" [Електронний ресурс] - Режим доступу: https://medium.com/@bhanushaliyash2000/implementing-an-auth-guard-with-jwt-tokens-in-nest-js-92176a9c3457 (дата звернення 01.11.2023р) – Назва з екрана.

12. "Authentication and Authorization in NestJS: A Complete Guide" [Електронний ресурс] - Режим доступу: https://levelup.gitconnected.com/authentication-and-authorization-in-nestjs-a-complete-guide-6e532e979f1c (дата звернення 02.11.2023р) – Назва з екрана.

13. "How to Configure SSL on PostgreSQL" [Електронний ресурс] - Режим доступу: https://www.cherryservers.com/blog/how-to-configure-ssl-on-postgresql

14. "Node.js Project Architecture: Best Practices" [Електронний ресурс] - Режим доступу: https://blog.logrocket.com/node-js-project-architecture-best-practices/ (дата звернення 15.11.2023р) – Назва з екрана.

15. "Node.js Architecture and 12 Best Practices for Node.js Development" [Електронний ресурс] - Режим доступу: https://scoutapm.com/blog/nodejs-architecture-and-12-best-practices-for-nodejs-development (дата звернення 12.11.2023р) – Назва з екрана.

16. "Understanding Node.js Architecture" [Електронний ресурс] - Режим доступу: https://www.simplilearn.com/understanding-node-js-architecture-article (дата звернення 10.11.2023р) – Назва з екрана.