МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

ПРОЕКТУВАННЯ ТА РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ ДОКУМЕНТООБІГУ

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «бакалавр»

Виконав: здобувач
спеціальності: 122 Комп'ютерні науки
Освітньо-професійної (наукової) програми:
Комп'ютерні науки
Морозов Олександр Олексіойвич
Керівник: канд. ф.-м. наук, проф.,
Співаковський О. В.
Рецензент: доц. кафедри алгебри,
геометрії та мат. аналізу
Григор'єва В. Б.

Херсон – Івано-Франківськ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface.

SPA – Single Page Application.

HTTP – HyperText Transfer Protocol.

REST – Representational State Transfer.

URL – Uniform Resource Locator.

JSON – JavaScript Object notation.

ВСТУП

Актуальність теми: У сучасному світі документообіг є невід'ємною частиною бізнесу, де швидкість та якість обміну документами має велике значення. Створення та розробка сервісу документообігу, здатного забезпечити швидку та ефективну обробку документів, може допомогти підвищити ефективність роботи організації та зменшити витрати часу на вирішення питань, пов'язаних з обробкою документів.

***Об'єкт дослідження*:** проектування та розробка серверної частини сервісу документообігу.

***Предмет дослідження*:** розробка та застосування технологій, які забезпечують ефективну та швидку обробку документів в сервісі документообігу.

***Мета дослідження*:** розробка та реалізація серверної частини сервісу документообігу на базі фреймворка NestJS та бази даних PostgreSQL для поліпшення процесів документообігу в організації.

Завдання дослідження:

1) Аналіз теоретичних аспектів документообігу та сучасних підходів до його автоматизації.
2) Проектування та розробка серверної частини сервісу документообігу на базі фреймворка NestJS та бази даних PostgreSQL.
3) Тестування та оптимізація розробленої системи для забезпечення максимальної продуктивності та ефективності обробки документів.
4) Розробка документації для розробленої системи.

***Структура дослідження*:** вступ, два розділи, висновки.

РОЗДІЛ 1
ТЕОРИТИЧНІ ЗАСАДИ ПРОЕКТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ДОКУМЕНТООБІГУ

1.1 Аналіз потреб користувачів в сервісі документообігу

У сучасному бізнесі робота з документами має велике значення. Сервіси документообігу дозволяють забезпечити ефективне управління документами та процесами, пов'язаними з ними. Із зростанням обсягів роботи з документами, відповідно зростає і складність їх обробки, що призводить до потреби в автоматизації процесів документообігу.

Аналіз потреб користувачів є важливою складовою проектування будь-якого сервісу, включаючи і сервіс документообігу. Для досягнення максимальної ефективності та задоволення потреб користувачів, потрібно вивчити їх потреби, навички та проблеми, пов'язані з документообігом.

У процесі аналізу було виявлено, що користувачі сервісів документообігу потребують зручного та швидкого доступу до документів, можливості працювати з документами в колективі та контролювати доступ до них, а також автоматизації рутинних операцій, пов'язаних з обробкою документів.

Крім того, користувачі вимагають від сервісу високої безпеки та надійності зберігання документів, мати функціонал для інтеграції зі сторонніми сервісами, які використовуються користувачами., а також зручного та інтуїтивно зрозумілого інтерфейсу.

На основі результатів аналізу потреб користувачів можна зробити висновок, що сервіс документообігу повинен мати широкий функціонал,

зручний інтерфейс та високий рівень безпеки, щоб задовольняти потреби користувачів та забезпечувати ефективний документообіг.

1.2 Аналіз і порівняння існуючих систем документообігу з розробленою системою

Система документообігу є важливою складовою будь-якої компанії або організації, яка працює з великим обсягом документів. У світі існує багато різноманітних систем документообігу, і кожна з них має свої переваги та недоліки. Для порівняння були обрані такі системи, як "Docsvision", "Google Документи" та "Microsoft SharePoint".

Перш за все, варто зазначити, що розроблена система документообігу має кілька абсолютних переваг порівняно з існуючими системами. Першою перевагою є те, що система розроблена з використанням сучасних технологій та архітектур, таких як NodeJS, TypeScript та мікросервісна архітектура, що дозволяє досягти високої продуктивності та масштабованості. Другою перевагою є те, що система має досить простий та зрозумілий інтерфейс, що дозволяє користувачам швидко зорієнтуватись та працювати з системою.

Розглянемо сервіс документообігу "Google Документи". Ця система дозволяє створювати, редагувати та зберігати документи онлайн. Вона має декілька переваг, серед яких є доступ до документів з будь-якого пристрою, можливість співпраці над документами в режимі реального часу, безкоштовний доступ до сервісу та великий обсяг безкоштовного сховища для документів. Однак, ця система не підходять для роботи з конфіденційною інформацією, так як ці документи можуть бути доступні для перегляду іншим користувачам. Також, ця система не підходить для роботи з документами великої об'ємності, оскільки вона може сповільнюватись при обробці великих файлів.

Далі розглянемо "Microsoft SharePoint". Ця система є більш розширеною в порівнянні з "Google Документами", оскільки вона дозволяє не тільки створювати та редагувати документи онлайн, але і зберігати, організовувати та ділитись документами внутрішньою командою. Крім того, "Microsoft SharePoint" дозволяє налаштовувати права доступу до документів та визначати рівні доступу користувачів до них. Однак, ця система вимагає певних знань та навичок для її використання, а також має платну версію, яка не підходить для всіх користувачів.

Docsvision - це ще один популярний продукт для документообігу в Україні. Ця система надає можливість зберігати, обробляти та розповсюджувати документи в організації. Програмний продукт має розподілену архітектуру, що дозволяє підвищити стабільність та надійність роботи системи. Docsvision також має можливості для створення власних робочих процесів, аналізу статистичних даних та забезпечує високий рівень безпеки за допомогою шифрування даних та контролю доступу до них [1].

Проте, як і в попередніх системах, Docsvision має свої недоліки. Наприклад, платформа має високі вимоги до апаратного забезпечення сервера, що може стати проблемою для менших організацій з обмеженими бюджетами. Крім того, програмний продукт Docsvision не є безкоштовним і його вартість може бути значною залежно від конкретних потреб користувача та умов контракту.

Система документообігу, розроблена в рамках даної роботи, має декілька переваг порівняно з існуючими системами.

По-перше, система має відкритий вихідний код, що дозволяє користувачам адаптувати систему до власних потреб. Крім того, відкритість вихідного коду забезпечує більшу безпеку в разі виявлення

вразливостей, оскільки будь-який розробник може виявити та виправити проблему.

По-друге, система дозволяє використовувати різні бази даних, що дає можливість вибрати найбільш оптимальний варіант залежно від потреб користувача. Крім того, система використовує власні механізми кешування, що дозволяє зменшити навантаження на базу даних та забезпечити швидкий доступ до інформації.

По-третє, система має гнучку архітектуру, що дозволяє розширювати функціональність за допомогою плагінів. Це дає можливість користувачам налаштувати систему під свої потреби та додати необхідні функції.

Отже, підсумовуючи, розроблена система документообігу має переваги порівняно з існуючими системами, що дозволяє забезпечити більш ефективну роботу з документами та зменшити час, необхідний для їх обробки.

1.3. Вибір технологій для проектування серверної частини

Розглянувши вимоги проекту та можливості наявних технологій, було прийнято рішення використовувати для проектування серверної частини NodeJS, NestJS та PostgreSQL. NodeJS є відомим та широко використовуваним середніми та великими підприємствами для створення серверної частини веб-додатків. NestJS, з іншого боку, є фреймворком для NodeJS, який надає можливість розробки високомасштабованих та ефективних серверних додатків, забезпечуючи багатофункціональність та додаткові безпекові функції. PostgreSQL було вибрано для зберігання даних, оскільки він є одним з найбільш потужних та надійних реляційних баз даних, який підтримує багато

складних операцій та має широкий набір інструментів для адміністрування та моніторингу.

Крім того, обрані технології мають відкритий код та активну спільноту розробників, що дає можливість швидко отримувати підтримку та вирішувати можливі проблеми. Для забезпечення безпеки та захисту даних використовуються додаткові бібліотеки та інструменти, такі як PassportJS для автентифікації користувачів та HelmetJS для захисту від атак типу Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Вибір цих технологій дозволяє створити ефективну та масштабовану серверну частину, яка відповідає вимогам проекту та забезпечує високу безпеку даних.

1.4 Архітектура клієнт-серверної взаємодії веб-додатків

Архітектура клієнт-серверної взаємодії веб-додатків є важливою складовою розробки будь-якої системи, яка базується на мережевому взаємодії з користувачем. За допомогою такої архітектури досягається ефективне та швидке взаємодія між клієнтом та сервером, а також забезпечується більш висока безпека даних.

У веб-додатках існують два основні типи архітектур - традиційна та SPA. Традиційна архітектура включає в себе надсилання запитів на сервер за кожною дією користувача, що часто призводить до перезавантаження сторінки та втрати даних. У свою чергу, SPA використовує асинхронну взаємодію з сервером та дозволяє оновлювати тільки ту частину сторінки, яка змінилася. Це забезпечує більш швидку та зручну взаємодію з користувачем [2].

Однією з найпопулярніших технологій SPA є React.js. Вона дозволяє зручно та ефективно розробляти веб-додатки з високою швидкістю роботи та зручним інтерфейсом [3].

Також варто згадати про технологію WebSocket, яка дозволяє створювати двосторонні з'єднання між клієнтом та сервером та передавати дані у режимі реального часу. Вона знаходить застосування в системах миттєвого спілкування, онлайн-іграх та інших сферах [4].

Клієнт - це програма або людина, яка отримує доступ до послуг API. Браузер також може бути клієнтом, якщо він запитує URL, який в свою чергу може викликати один або більше HTTP запитів [5].

Найбільш поширеними методами HTTP є:

- GET: Отримання ресурсу.
- POST: Створення нового ресурсу.
- PUT: редагування або оновлення наявного ресурсу.
- DELETE: Видалення ресурсу.

Ресурс – це конкретна інформація, яку API надає клієнту. Це може бути будь-що: хештеги, профілі, коментарі, веб-адреси, твіти і т. д. Всі ресурси мають унікальне ім'я, зване ідентифікатором ресурсу, і всі вони зберігаються на сервері [6].

Коли клієнт запитує за допомогою RESTful API, сервер передає стандартизоване представлення стану ресурсу в систему клієнта. Це означає, що сервер передає клієнту не сам ресурс, а уявлення його стану, тобто стан ресурсу певну тимчасову мітку.

Відповіді надсилаються у полегшеному форматі, щоб полегшити інтерпретацію. Формат JSON популярний, тому що читається як людьми, так і машинами, і перевершує їх за доступністю для

користувачів. Він також сумісний із багатьма іншими мовами програмування.

REST явно обмежує комунікаційні можливості клієнта та сервера. Клієнт виключно робить запити, а сервер відповідає на них. Таким чином, ініціатором кожної взаємодії є клієнт, який знає URL запитуваного ресурсу. Сервер відповідає, передаючи дані без можливості самостійного внесення змін [7].

В розробленому сервісі документообігу використана дволанкова клієнт-серверна архітектура, яка передбачає взаємодію двох програмних модулів — клієнтського та серверного (рисунок 1.1). Також ця архітектура передбачає використання моделі товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони [8]. В сервісі реалізовано саме модель товстого клієнта.

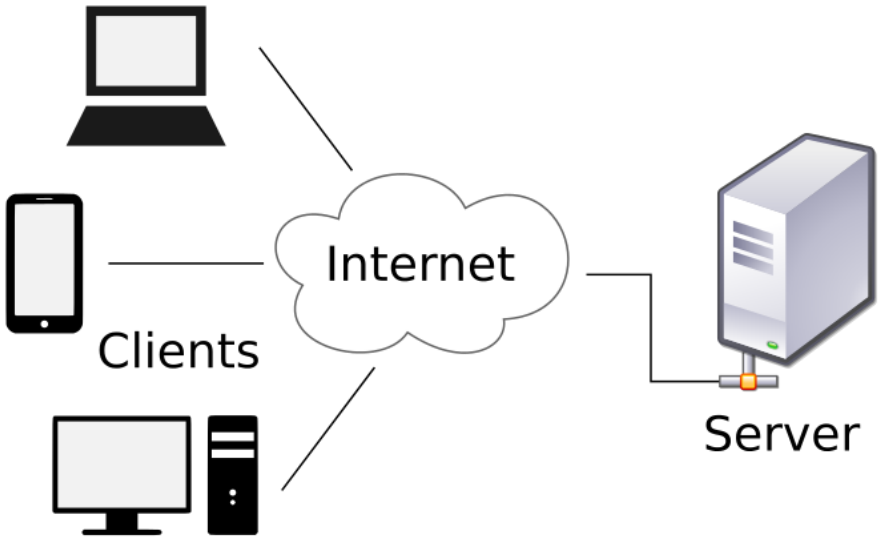

Рисунок 1.1 – Клієнт-серверна архітектура

РОЗДІЛ 2
РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ ДОКУМЕНТООБІГУ

2.1 . Огляд і вибір архітектури серверної частини.

Однією з основних вимог до архітектури є забезпечення масштабовності сервісу для підтримки зростаючого навантаження в майбутньому. Також потрібно забезпечити високу швидкодію та стабільність системи.

Платформою для розробки було обрано Node.js - це платформа для розробки серверних додатків на JavaScript, побудованою на движку V8 від Google. Ця платформа дозволяє використовувати JavaScript для створення високопродуктивних, масштабованих і ефективних додатків на стороні сервера [8].

ECMAScript - це специфікація, на якій базується JavaScript, мультипарадигменна мова програмування, яка підтримує об'єктно-орієнтований, імперативний та функціональний стилі програмування. [10].

Node.js складається з багатьох компонентів, включаючи модульну систему, яка дозволяє розробникам підключати та використовувати зовнішні модулі. Підтримка асинхронного виконання завдань, зокрема операцій вводу/виводу, є однією з ключових особливостей Node.js, що дозволяє досягати високої продуктивності [11] (рисунок 2.1).

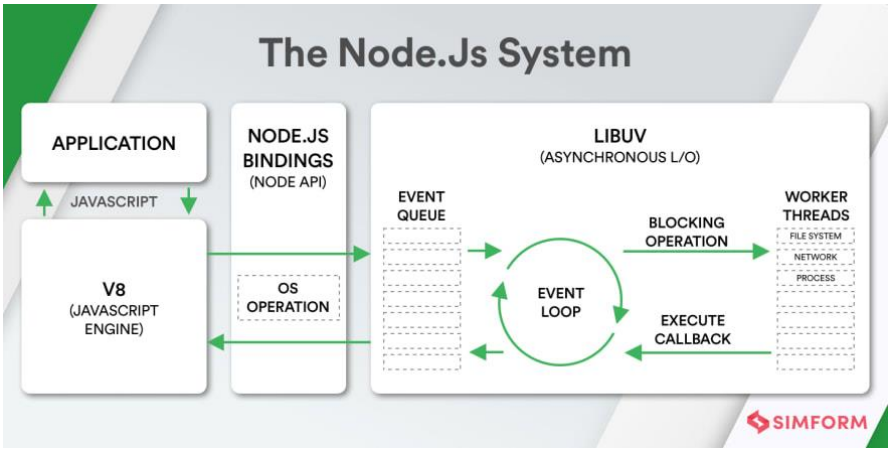

Рисунок 2.1 – Схема роботи платформи NodeJS

Крім того, в проекті використовувався TypeScript - мова програмування, яка є строго типізованою версією JavaScript. TypeScript дозволяє забезпечити більшу стабільність та надійність коду, що допомагає зменшити кількість помилок та покращити підтримку проекту [12]. TypeScript може бути транспільований у JavaScript, що дозволяє використовувати його з існуючим JavaScript-кодом.

Для реалізації серверної архітектури використовується фреймворк NestJS, який забезпечує високу швидкість розробки та реалізації складних серверних додатків. NestJS забезпечує підтримку модульної архітектури, яка забезпечує зручну роботу з різними частинами системи [13].

Крім того, NestJS дозволяє використовувати декоратори для опису різноманітних функцій, таких як роутинг, валідація, авторизація, та інші, що значно спрощує розробку серверного додатку. Також, NestJS має вбудований механізм ін'єкції залежностей, що дозволяє легко керувати залежностями та забезпечує гнучкість у розробці серверної архітектури.

Однією з основних вимог до архітектури є забезпечення масштабовності сервісу для підтримки зростаючого навантаження в майбутньому. Також потрібно забезпечити високу швидкодію та

стабільність системи. У процесі вибору технологій для проекту було враховано кілька важливих факторів. Першим з них була масштабованість системи, оскільки є потреба в розширенні функціональності та збільшенні кількості користувачів.

Було вирішено використовувати модульну архітектуру для розробки серверної частини сервісу документообігу (рисунок 2.2). Кожен модуль має відповідальність за конкретну функціональність системи та складається з набору компонентів, таких як контролер, сервіс та репозиторій даних.

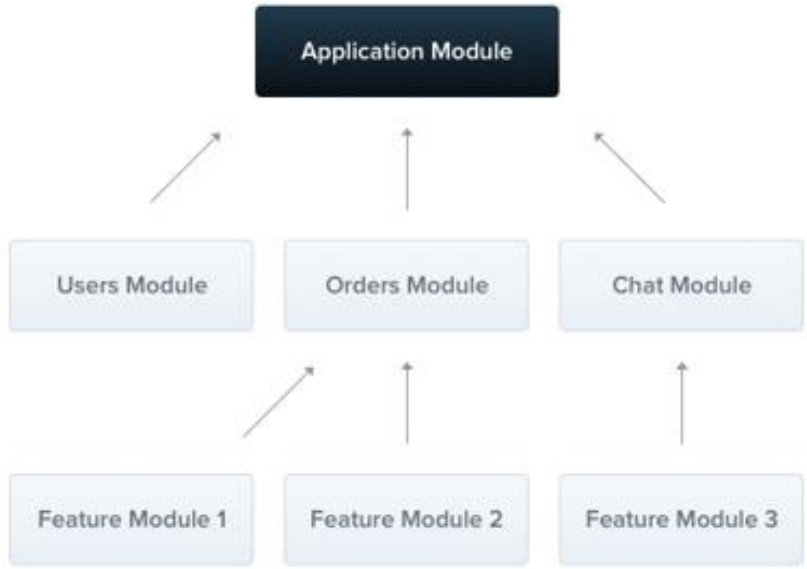

Рисунок 2.2 – Схема модульної архітектури

Оскільки у системі є різноманітні модулі, то для забезпечення їх взаємодії використовується архітектурний паттерн "Сервіс-орієнтована архітектура" (SOA). За допомогою цього паттерну кожен модуль системи може мати власний сервіс [14].

Кожен модуль має відповідальність за конкретну функціональність системи та складається з набору компонентів, таких як контролер, сервіс та репозиторій даних. В модулі, кожен контролер

відповідає за обробку HTTP запитів та взаємодію з сервісом, виконуючи такі завдання, як отримання даних, перевірка прав доступу користувача та відправлення відповіді на запит. Кожен сервіс відповідає за бізнес-логіку та виконання певних завдань, включаючи обробку даних та взаємодію з репозиторієм даних. Репозиторій даних відповідає за доступ до бази даних та збереження/отримання даних з бази.

Для зберігання даних було використано PostgreSQL, що є потужною та стабільною системою управління базами даних з відкритим кодом. PostgreSQL дозволяє зберігати великі обсяги даних та підтримує багато різних типів даних, що дозволяє зберігати в базі даних будь-які дані, не залежно від їх типу [15].

Серверна частина сервісу складається з чотирьох модулів: "користувачі", "документи", "підписи" та "організації". Кожен модуль містить контролер, сервіс та репозиторій даних, які взаємодіють між собою. Наприклад, контролер модуля "користувачі" містить методи для обробки запитів, що стосуються користувачів, сервіс виконує бізнес-логіку, пов'язану з користувачами, а репозиторій даних зберігає та взаємодіє з даними користувачів у базі даних.

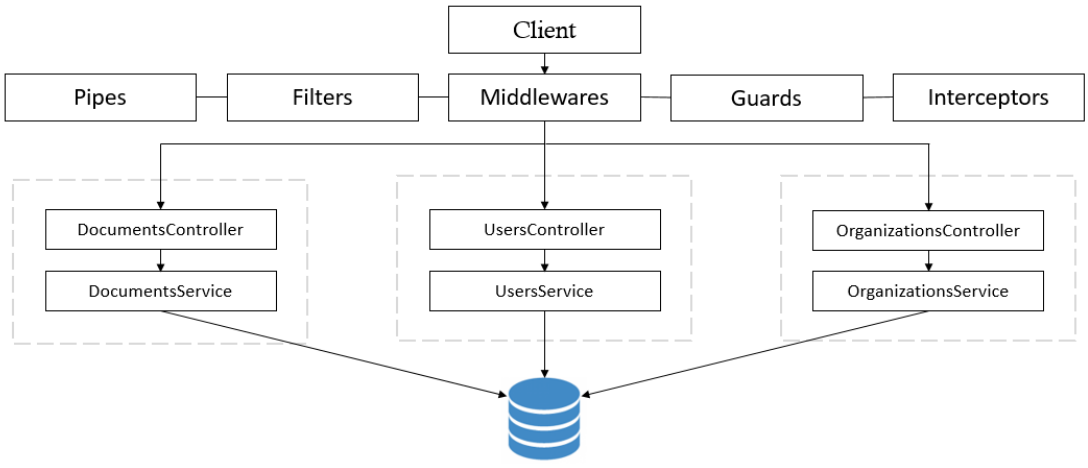

Рисунок 2.3 – Схема модульної архітектури

Описуючи середовище, в якому працює сервер, важливо відзначити використання Docker - це відкритий інструмент для розробки, доставки та експлуатації програмного забезпечення в контейнерах. Він дозволяє розробникам використовувати контейнери для пакування своїх програм та всіх необхідних компонентів, таких як бібліотеки та середовища виконання, що забезпечує їх переносимість між різними середовищами [16].

Dockerfile є текстовим файлом, що містить команди для автоматизованої збірки образу Docker. Це означає, що розробник може створювати образ своєї програми за допомогою команд, що зберігаються в Dockerfile. Цей файл містить інструкції для встановлення та конфігурування всіх необхідних компонентів та середовища виконання, які потрібні для запуску програми.

Docker Compose є інструментом для оркестрування контейнерів Docker та їх взаємодії. Він дозволяє розробникам визначати та запускати багатоконтейнерні додатки за допомогою конфігураційного файлу docker-compose.yml. У цьому файлі вказуються всі необхідні параметри та налаштування контейнерів, що потрібні для запуску додатку.

Для розробки сервісу документообігу було використано Docker, Dockerfile та docker-compose. У Dockerfile були визначені необхідні налаштування для запуску сервера NestJS та бази даних PostgreSQL. Docker-compose був використаний для запуску та керування контейнерів. Це дозволило легко та швидко створити середовище для розробки та разгортанню додатку на серверах. Завдяки цьому, розробники зможуть легко переносити додаток між різними середовищами без складнощів встановлення та налаштування всіх необхідних компонентів та середовищ виконання.

Таким чином, використана архітектура серверної частини дозволяє ефективно взаємодіяти з базою даних, масштабуватись та забезпечувати високу швидкість обробки даних, зберігаючи принцип єдиної відповідальності та дотримуючись сучасних стандартів безпеки.

2.2. Проектування бази даних та її моделювання для сервісу документообігу

За основу представлення даних була взята реляційна модель. Реляційна модель даних забезпечила стандартний спосіб представлення та запиту даних, який може використовуватися будь-якою програмою. Головною перевагою моделі реляційної бази даних було використання таблиць, які є інтуїтивно зрозумілим, ефективним і гнучким способом зберігання та доступу до структурованої інформації.

Для більш ефективнішого використання можливостей реляційної моделі даних, біло вирішено використовувати PostgreSQL - це потужна об'єктно-реляційна база даних з відкритим вихідним кодом. Вона підтримує великий набір функцій та можливостей, таких як транзакції з ізоляцією, індексація тексту, оптимізація запитів та багато іншого. PostgreSQL підтримує мови програмування, такі як PL/Python, PL/Perl та PL/Java, що дозволяє використовувати їх для написання функцій, які можна виконувати прямо в базі даних.

PostgreSQL використовується для різних завдань, включаючи веб-додатки, аналітику даних, геопросторовий аналіз та багато іншого. Вона досить популярна серед розробників через свою доступність та відкритий вихідний код, що дозволяє використовувати її безкоштовно та модифікувати її під власні потреби.

Обрання PostgreSQL було зумовлено декількома факторами. По-перше, PostgreSQL є відкритою і безкоштовною реляційною системою керування базами даних (СКБД), яка дозволяє забезпечувати стійкість, надійність та безпеку даних. Крім того, ця база даних має високий рівень підтримки стандартів SQL, що дозволяє розробникам легше переходити на PostgreSQL з інших реляційних баз даних.

Додатково, PostgreSQL підтримує багато корисних функцій, таких як транзакції, засоби контролю версій, індексацію повнотекстового пошуку, підтримку геоданих та багато іншого. PostgreSQL є надійним і популярним рішенням для бізнес-додатків, особливо тих, що потребують високої швидкості роботи з великими обсягами даних.

Також важливим фактором є наявність TypeORM, який є обгорткою над реляційними СУБД (Система управління базами даних), в тому числі над PostgreSQL. TypeORM дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу, що спрощує розробку та збереження коду.. Також ця система надає можливість підтримує такі функції, як міграції баз даних та збереження моделей сутностей в коді, що є важливим для розробки серверної частини додатку та описувати моделі сутностей та відносин між ними за допомогою TypeScript [17]. Слід зауважити універсальну підтримку різних типів взаємодії з базою даних, такі як SQL-запити, міграції та запити на мові запитів більш високого рівня, що спрощує роботу з базою даних та дозволяє забезпечити її цілісність та консистентність.

Для моделювання бази даних використовувалися методології Entity-Relationship (ER) та Normalization. Було проведено аналіз вимог до бази даних та розроблено ER-діаграму, яка відображає зв'язки між сутностями, атрибути та їх типи (рис. 2.4).

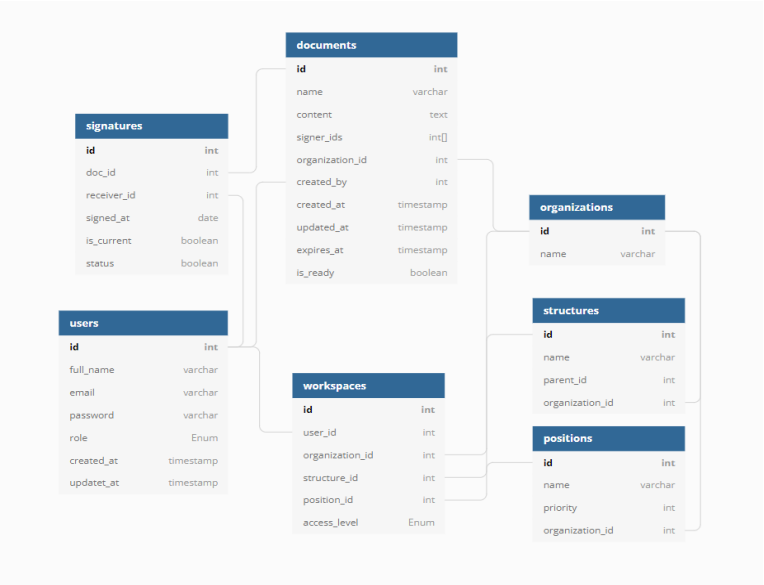

Рисунок 2.4 – Діаграма бази даних

Система документообігу потребує зберігання таких сутностей, як користувачі, документи, папки, ролі, права доступу тощо. З метою оптимізації бази даних та забезпечення її нормалізації було застосовано принципи третьої нормальної форми (3NF).

У результаті проектування та моделювання бази даних було створено оптимізовану та нормалізовану базу даних, яка забезпечує високу продуктивність та масштабованість сервісу документообігу.

2.3. Тестування та оптимізація розробленої системи

Одним з найголовніших етапів розробки та підтримки працездатності системи є тестування. Це може бути тестування цілих модулів системи або окремих її компонентів. Для досягнення найбільш ефективного результату було проведено юніт-тестування – це є однією з ключових складових процесу розробки програмного забезпечення, яка дозволяє перевірити коректність роботи окремих функцій та модулів відокремлено від інших компонентів системи.

Переваги юніт-тестування над ручним тестуванням полягають у тому, що вони дозволяють ефективно виявляти та виправляти помилки на ранній стадії розробки, що забезпечує більш швидке та ефективне виконання проекту. Крім того, вони забезпечують високу якість коду та підвищують стійкість системи до змін, що забезпечує більш ефективне її супроводження. Юніт-тестування також допомагають у плануванні робочих процесів, оскільки вони дозволяють розраховувати час необхідний на розробку окремих модулів та функцій, що дозволяє ефективніше розподіляти робочий час та забезпечує виконання проекту в строк.

Для цього типу тестування було використано Jest - це відкрите та безкоштовне фреймворк для тестування JavaScript коду, який розробляється та підтримується компанією Facebook. Він підтримує тестування коду на Node.js, веб-браузерах та React Native додатках [18].

Jest має вбудовану підтримку для засобів, таких як Babel, TypeScript та Node.js, тому його легко налаштувати для будь-якого проекту. Цей фреймворк має багато функцій, які спрощують процес тестування, таких як автоматичне створення зразків (snapshots), мокування (mocking) та визначення області покриття (code coverage).

Що стосується тестування NestJS додатків, Jest є вибором за замовчуванням для написання юніт-тестів для контролерів та сервісів. Оскільки Jest є частиною стеку технологій NestJS, він дозволяє легко налаштувати тестування і забезпечує зручний та ефективний процес написання та запуску тестів.

В розробленому сервісі всі модулі були покриті юніт-тестуванням таких компонентів, як Controller/Service, а саме:

- Auth.
- Users

- Organizations
- Documents

Однак, не всі помилки можна виявити за допомогою юніт-тестів, особливо ті, які виникають в процесі взаємодії між компонентами системи. Тому було проведено інтеграційні тести для перевірки взаємодії між різними модулями та їх функціональності. Також були проведені автоматизовані тести для тестування API.

Також були проведені дії щодо оптимізації роботи системи. Важливим фактором в цьому є оптимізація запитів до бази даних, яка є важливою задачею для забезпечення ефективної роботи системи. Незалежно від того, яка база даних використовується, оптимізація запитів дозволяє покращити продуктивність додатку та зменшити навантаження на сервер.

Під час тестування було виявлено деякі проблеми та помилки, які були виправлені. Було проведено оптимізацію запитів до бази даних та покращено швидкість роботи системи.

Однією з переваг PostgreSQL є його підтримка для оптимізації запитів, що дозволяє створювати запити, які працюють швидко та ефективно. Для досягнення цього, були застосовані наступні техніки:

1. Індексація - індексація дозволяє швидко знаходити записи в таблиці, що відповідають певним умовам. Для досягнення цього, створено індекси для стовпців, які часто використовуються в умовах WHERE або JOIN. Використання індексів дозволило значно зменшити час виконання запитів до бази даних.
2. Кешування – механізм, який дозволяє зберігати результати запитів до бази даних в пам'яті сервера, що дозволило

значно зменшити час виконання запитів. Наприклад, якщо запит повторюється декілька разів з однаковими параметрами, то результат запиту можна зберегти в кеші, щоб не виконувати його знову.

3. Завдяки поділу таблиць стало можливим розподілити дані на декілька таблиць залежно від їх характеристик та виконувати запити до цих таблиць паралельно.

Отже, під час проведення тестування було виявлено деякі проблеми та помилки, які були виправлені. Було проведено оптимізацію запитів до бази даних та покращено швидкість роботи системи.

2.4. Розробка документації програмного інтерфейсу

Розробка документації API є важливим етапом у створенні будь-якої серверної системи. Документація API - це набір інструкцій та описів, які дозволяють розробникам створювати клієнтські додатки, що взаємодіють зі створеним сервером.

У даній кваліфікаційній роботі, для створення документації API, був використаний фреймворк NestJS разом з Swagger. Swagger - це набір інструментів для розробки та документування API. Він дозволяє автоматично генерувати документацію API, що спрощує розробку клієнтських додатків [19].

У NestJS, інтеграція з Swagger здійснюється за допомогою модуля nestjs/swagger. Цей модуль дозволяє генерувати документацію API на основі маршрутів, які описуються за допомогою декораторів Controller, Get, Post, тощо. Додатково до цього, можна вказати опис параметрів, типів відповідей, та інших характеристик за допомогою декораторів ApiParam, ApiBody, ApiResponse та інших [20].

Створення документації API дозволяє забезпечити прозорість та доступність інформації про структуру та функції серверної системи, що зробить розробку клієнтських додатків більш ефективною та швидкою. Також, документація API може використовуватись для збільшення безпеки серверної системи, оскільки вона дозволяє встановити межі вхідних параметрів, типів відповідей та інші обмеження на доступ до функцій сервера.

У підсумку, використання Swagger у поєднанні з NestJS дозволяє створювати документацію API швидко та ефективно, що робить розробку клієнтської частини більш ефективною.

ВИСНОВКИ

Зважаючи на результати дослідження, можна зробити висновок про ефективність розробленої серверної частини сервісу документообігу. Реалізована система забезпечує ефективну та швидку обробку документів в сервісі документообігу, що позитивно впливає на роботу організації та зменшує витрати часу на вирішення питань, пов'язаних з обробкою документів.

Аналіз теоретичних аспектів документообігу та сучасних підходів до його автоматизації показав, що розробка сервісу документообігу є актуальною на ринку бізнесу. Реалізація даної системи дозволяє підвищити ефективність роботи організації та покращити якість обміну документами.

Проектування та розробка серверної частини сервісу документообігу на базі фреймворка NestJS та бази даних PostgreSQL були проведені з урахуванням сучасних технологій та найкращих практик розробки програмного забезпечення. Результатом цього є створена система з ефективною архітектурою, що забезпечує максимальну продуктивність та ефективність обробки документів.

Тестування та оптимізація розробленої системи показали, що вона працює стабільно та надійно, забезпечуючи максимальну продуктивність та ефективність обробки документів. Розроблено документацію для розробленої системи, що дозволяє користувачам легко та швидко ознайомитися з її функціональними можливостями та інструкціями щодо її використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Система документообігу Docvision. URL: https://www.docsvision.com/uk/
2. Архітектура SPA. URL: https://en.wikipedia.org/wiki/Single-page_application
3. Бібліотека для створення веб-інтерфейсів React.JS. URL: https://react.dev
4. Технологія WebSocket. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
5. Веб-кліент. URL: https://cloudinfrastructureservices.co.uk/what-is-a-web-client-and-how-it-works-with-web-server-applications/
6. Веб-ресурс. URL: https://dbpedia.org/page/Web_resource
7. Rest-архітектура. URL: https://uk.wikipedia.org/wiki/REST
8. Клієнт-серверна архітекутура. URL: https://medium.com/@IvanZmerzlyi/%D0%BA%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0-%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D1%82%D0%B0-%D1%80%D0%BE%D0%BB%D1%96-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D1%96%D0%B2-9893d8048229
9. Середа виконання програмного коду NodeJS. URL: https://nodejs.org/
10. Специфікація ECMAScript. URL: https://ru.wikipedia.org/wiki/JavaScript

11. Потоки в NodeJS. URL: https://nodejs.dev/learn/nodejs-streams
12. Мова програмування TypeScript. URL: https://www.typescriptlang.org/
13. Фреймворк для розробки серверної частини NestJS. URL: - https://docs.nestjs.com/
14. Сервіс-орієнтована архітектура. URL: https://aws.amazon.com/what-is/service-oriented-architecture/
15. Реляційна база даних PostgreSQL. URL: https://www.postgresql.org/about/
16. Інструмент для розгортання додатків Docker. URL: https://www.docker.com/resources/what-container/
17. Бібліотека Об'єктно-реляційного відображення TypeORM. URL: https://typeorm.io/
18. Фреймворк для тестування Jest. URL: https://jestjs.io/
19. Інструмент для документаціїї API Swagger. URL: https://swagger.io/about/
20. Інтеграція NestJS з модулем Swagger. URL: https://docs.nestjs.com/openapi/introduction

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Морозов Олександр Олексійович, учасник освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема СтатутуУніверситету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційномурепозитарії;

- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науководослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів іспівробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних здіяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

12.09.2019 р.		Морозов Олександр
(дата)	(підпис)	(ім'я, прізвище)