Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

АВТОРЕФЕРАТ

випускної кваліфікаційної роботи бакалавра

«СИСТЕМА ПОШИРЕННЯ СТАТИЧНОГО КОНТЕНТУ»

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проєкту

№	П.І.Б.	Група
1	Дмитрієв М. Ю.	КН-П-181
2	Горелік М. А.	КН-П-181
3	Міхов І. С.	КН-П-181

Автор звіту

_____ Дмитрієв Максим Юрійович

РЕЗУЛЬТАТИ ЕКСПЕРТИЗИ

Експерт	П.І.Б.	Оцінка	Підпис
Науковий керівник	доц. Самойленко Д. М.		
Рецензент	доц. Суліма М. М.		
ДЕК (захист проєкту)	доц. Голова Е. К.		

Одеса – 2022

АНОТАЦІЯ

УДК 004.9

Д-30

Дмитрієв М. Ю. Система поширення статичного контенту: Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп’ютерні науки». - Одеса: ОТУШ, 2022 - 35 с.

Робота присвячена дослідженням, розвиненню, проектуванню та впровадженню інформаційної системи, головним призначенням якої є поширення статичного контенту. Додаткові функції системи включають реєстрацію, авторизацію та автентифікацію, засоби створення контенту, відгуків, пошук користувачів, повідомлення, особистий кабінет користувача та інше.

Актуальність роботи визначається поширеністю у сучасних мережевих інформаційних технологіях використання систем розподіленої обробки даних, сервіс-орієнтованих систем та ефективного розроблення програмного забезпечення. Додатково, актуальність диктується потребою впровадження мобільних інформаційних систем та реалізації у них покращених засобів захисту інформації, зокрема, персональних даних.

Об'єктом дослідження у рамках даної роботи виступає створення програмного забезпечення з використанням сервісно-орієнтованих систем, систем розподіленої обробки та збереження даних.

Предметом дослідження була обрана комплексна система обміну користувацьким контентом.

У якості задач дослідження було встановлене наступне:

- провести огляд та впровадити інформаційні технології для ефективного розроблення програмного забезпечення комп’ютерних мереж і систем розподіленої обробки даних, встановити напрями їх удосконалення
- дослідження, розроблення та впровадження інтернет-технологій для побудови сервіс-орієнтованих систем, а також для організації та реалізації систем розподіленої обробки інформації

- скласти програмне забезпечення, реалізувавши у ньому результати, досягнуті у попередніх задачах, провести випробування за різних умов та для різних пристроїв.

Методи дослідження, що використані для досягнення поставлених задач, утворені суперпозицією методів системного аналізу, моделювання та когнітивної методології.

При складанні звіту використано 15 посилання на друковані та електронні джерела інформації.

ЗМІСТ

ВСТУП

З розвитком і поширенням інформаційних технологій у людства з'являється багато можливостей для комунікацій та поширенню інформації в мережі інтернет. Стратегія розвитку інформаційного суспільства в Україні визначає, що «комунікаційні технології є необхідним інструментом соціально-економічного прогресу, який є одним з основних чинників інноваційного розвитку економіки».

За даними багатьох інформаційних платформ станом на початок 2021 року кількість українських інтернет-користувачів становила майже 30 мільйонів, тобто близько 67% населення країни. З кожним роком цифра користувачів збільшується мінімум на 4%.

У зв’язку з інтенсивним зростанням кількості інтернет-користувачів, збільшується і обсяг даних які вони завантажують. Збільшується також кількість та складність обчислень цих даних за рахунок зростання вимог скорості доступа до них. Це призводить до того, що у певний момент кожна система, при розробці якої не було закладено можливості гнучкого масштабування, стає не здатна справлятися з навантаженням та кількістю даних.

В рамках цієї роботи досліджується розробка соціальної мережі основною функцією якої є поширення статичного контенту з використанням методів гнучкого масштабування. Завдяки цьому користувачі веб-додатку матимуть змогу швидкого доступу до великої кількості даних, можливість у поширенні, створенні та перегляду наявного контенту, який має бути забезпечений засобами перегляду, обговорення та взаємодії без необхідності завантаження.

Детальне технічне завдання до проєкту наводиться далі. У рамках даного звіту розглядається загальна архітектура проекту, організація ефективного командного розроблення програмного забезпечення і систем розподіленої обробки та збереження даних, реалізація розподіленого сервісу збереження, завантаження, обробки та віддачі зображень, реалізація розподіленого сервісу збереження, віддачі та обробки повідомлень у режимі реального часу та розгортання проекту. Висновки відбивають результати, що досягнуті у рамках даної частини проєкту, загальний висновок є сукупністю звітів усіх учасників проєкту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

Загальний опис.

Програмне забезпечення (соціальна мережа), головна особливість якої полягає у поширенні, створенні та перегляду наявного контенту який має бути забезпечений засобами перегляду, обговорення та взаємодії без необхідності завантаження. Функції системи мають забезпечити повний життєвий цикл управління контентом: створення, актуалізація, модифікація та знищення.

Соціальна мережа базується на обміні фотографіями, дозволяє користувачам завантажувати контент, редагувати а також поширювати його між їхніми підписниками та всією платформою в цілому завдяки поширювальним алгоритмам.

Призначення:

ПЗ орієнтується на створення узагальненої платформи для обміну та поширенню статичних даних. Головний акцент робиться на засобах групування, пошуку та розподілу доступу до контенту. Функція відтворення контенту є безпосереднім предметом завдання. ПЗ створюється за прикладом сервісу фотохостингу, що надає послуги розміщення фотоматеріалів, у той же час має складатись за принципом універсальності, мати змогу змінити цільове призначення.

Вимоги до функціоналу. Обов'язкова частина

Головна сторінка

Основна сторінка додатку яка відображає увесь доступний контент. Має забезпечити усі необхідні засоби для

- Перегляд контенту створений користувачами з якими забезпечена підписка
- Пагінація
- Адаптив для різних типів пристроїв
- Можливість взаємодії з контентом

Реєстрація користувачів:

Користувач ПЗ має змогу зареєструватись як за допомогою авторизаційних даних (логіну/паролю), так і за допомогою облікового запису google.

При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - електронна пошта. Засіб оперативного контактування при реєстрації підлягає верифікації через відповідь на надісланий лист (користувач авторизувався за допомогою логіну та паролю). Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.

За умови що користувач забув пароль, ЗОК має бути використаний для його відновлення. Схема відновлення паролю жорстко не регламентується.

Створення публікацій:

Зареєстрований користувач ПЗ має змогу створити публікацію з фотографією та описом у вільній формі. Публікація з’являється у стрічці підписників цього користувача.

Навігаційна панель

В ПЗ реалізовано декілька типів навігаційних панелей. Для користувачів пристроїв з великим розширенням дисплея використовується одна навігаційна панель з наступними реалізаціями

- Кнопка переходу до головної сторінки
- Панель пошуку
- Меню взаємодії з акаунтом

Для користувачів мобільних пристроїв реалізовано два види навігаційних панелей перший з них розташований зверху і має наступні функції

- Навігація по ПЗ
- Кнопка переходу до мессенджеру

Нижня мобільна навігаційна панель являє собою навігацію по основним сторінкам ПЗ

Управління власним контентом:

Основний ресурс зареєстрованого користувача ПЗ. Повинен забезпечити усі необхідні засоби для

- Створення метаданих (розміщення фото та його опису).
- Зміна у власному контенті (редагування)
- Збереження публікацій
- Перегляд поточної активності: історії переглядів, перелік відзначеного контенту (вибране), контактні особи (друзі), коментарі до контенту, тощо
- Пошук контенту інших користувачів за метаданими, групування та фільтрування результатів за різними ознаками, наявними у знайденому контенті. Для об'ємних результатів має бути забезпечена пагінація.
- Контактування з іншими користувачами, що розміщують контент, за допомогою ЗОК або через внутрішні засоби системи (чат)

Також засобами особистого кабінету має бути передбачено можливість зміни персональних даних. У разі зміни ЗОК необхідна його актуалізація за тією ж схемою, що й при реєстрації.

Взаємодія з контентом

Користувач має змогу встановити уподобання та залишити коментар.

Особистий кабінет користувача:

Основне робоче місце користувача ПЗ. Повинен забезпечити усі необхідні засоби для

- Налаштування профілю
- Налаштування додатку
- Перегляд власного контенту
- Перегляд збереженого контенту
- Можливість вибору взаємодії с контентом
- Адаптив для мобільних телефонів та екранів з великим розміром

Повідомлення

Модальная форма або сторінка завдяки якої користувач має можливість переглядати всі події які відбулися з профілем або контентом. Наприклад: повідомлення про додавання коментаря посту користувача, повідомлення про підписку, повідомлення про “вподобання”, та інше.

Пошук

Модальна форма або сторінка (в залежності від типу засобу) яка дозволяє здійснювати пошук інших користувачів. Має забезпечувати такі заходи

- Можливість пошуку по ніку та повному ім'ю користувача
- Пагінація
- Можливість переходу до сторінки знайденого профілю
- Механізм пошуку з повільним інтернет з'єднанням

Контент

Користувальницький інтерфейс який представляє собою пост. Має забезпечувати такі заходи

- Можливість взаємодії з постом
- Відображення контенту
- Відображення інформації
- Відображення відгуків
- Відображення різних типів посту в залежності від типу пристрою та контексту додатку
- Можливість залишення відгуку

Пост може бути у вигляді модального вікна або у вигляді окремої незалежної сторінки. Всі пости можно відкрити як окрему сторінку. Контент посту може бути у вигляді одного або декількох файлів формату: jpeg, webp. Всі пости мають різний вигляд та реалізацію. Кожний має адаптивні властивості.

Оптимізоване завантаження зображень

Бекенд має відправляти у відповіді на запит файл зображення у оптимізованому для конкретного запиту розмірі, таким чином зменшуючи загальну кількість трафику. Повинно бути виконано у вигляді стороннього веб-серверу задовольняючи принципи модульної моделі ПЗ.

Інше

В додатку реалізовано багато користувальницьких інтерфейсів: додаткові сторінки, модальні форми, поп апи, інше. Наприклад форма логіну, форма регістру, поп ап користувача, сторінка та модальна форма підписників, підписок,

сторінка та модальна форма уподобань, сторінка коментарів, сторінка додавання контенту, верхня навігаційна панель для великих екранів, верхня навігаційна панель для мобільних девайсів, нижня навігаційна панель для мобільних засобів. Кожен користувальницький інтерфейс має свій адаптив і здатний мати декілька візуальних тем в залежності від вибору користувача та теми користувацького пристрою.

Вимоги до функціоналу. Опціональна частина

Модерація

Для забезпечення дотримання загальноприйнятих норм спілкування, а також з метою недопущення публікування контенту, який містить заборонені компоненти (ненормативні висловлювання, заклики до протиправних дії, приниження чи дискримінація, тощо), усі інформаційні одиниці, а також створені користувачами метадані (описи, теги), а також коментарі мають можливість бути заблокованими. Право блокування належить кореневому адміністратору системи, а також модераторам. При погодженні інформаційна одиниця має зберігати ідентифікатор користувача, який встановив блокування. Також має бути зворотна можливість розблокування заблокованого контенту.

Мессенджер

Сторінка мессенджеру яка дозволяє користувачам мати співбесіду у режимі реального часу, тобто повідомлення приходять одразу. Сторінка мессенджеру має адаптив під різні типи пристроїв.

Розумна користувацька стрічка

Контент для користувачів має створюватись як найменше з двох джерел: публікацій користувачів на яких підписаний поточний користувач та публікацій які можуть бути цікаві поточному користувачеві. Спосіб визначення цікавого контенту визначає алгоритм який обробляє метрики користування платформою: вподобані публікації, час який був витрачений на перегляд публікації, підписки на користувачів та інше.

Сервіс статистики

ПЗ для збирання, зберігання та обробки статистичних даних користувачів. Предметом для збору статистики є такі події:

- перегляд профілю користувача
- перегляд публікації
- “вподобання” публікації та коментаря
- кількість часу проведене на сторінці користувача
- кількість часу перегляду публікації
- та інші

Сервіс розпізнавання контексту зображень

ПЗ для розпізнавання контексту зображень розробляється за допомогою алгоритму штучного інтелекту та дата-сетів завдяки якому система має можливість визначити контекст зображення. Ці дані можуть бути використані для безлічі технічних функціональностей: генерування розумної стрічки чи визначення статистики захоплень користувачів

Архітектура

ПЗ має бути спроектоване за архітектурою PWA (Progressive Web Application) з чітко відокремленими розподіленими вузлами системи:

- Backend - серверне програмне забезпечення, що забезпечує збереження та оброблення даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.
- Image Hosting - серверне програмне забезпечення, що забезпечує збереження, віддачу та оброблення зображень в оптимізованому форматі з метою зменшення загальної кількості веб трафіку.
- Notifications Service - серверне програмне забезпечення, що забезпечує збереження, віддачу та оброблення повідомлень в режимі реального часу.
- Web App - додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів усіх категорій

Усі модулі ПЗ (окрім візуальної моделі Mockup) повинні обов'язково включати до свого складу засоби

- тестування функціональності та модульні тести,
- розгортання на новому пристрої (інсталяції) із встановленням початкових таблиць у БД та кореневого користувача-адміністратора

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:

- root
    - Backend
    - Web App

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

Backend - NodeJS, Typescript, NestJS, PostgresSQL або MongoDB, TypeORM або Mongoose, PassportJS

Web App - JavaScript, TypeScript, ReactJS, Redux, Axios, Formik

Команда проєкту

Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проєкту.

Мінімальна команда проєкту - два учасники, у такому складі один учасник виконує модуль Backend та програмну частину Web, другий - модель Mockup та розміткову частину Web.

Рекомендована команда проєкту - три учасники, у такій команді кожен з учасників відповідає за одну з частин: Mockup, Backend, Web.

Дозволяється включення до команди більшої кількості учасників, у такому разі модуль Mobile App набуває обов'язкової реалізації. Також модуль Mockup може бути поділений на окремі моделі для Web та Mobile частин.

Звітність

За результатами виконання завдань із створення ПЗ кожен з учасників команди проєкту складає власний (індивідуальний) звіт, що складається з практичної частини та автореферату. Практична частина є прямим результатом роботи і подається у вигляді посилання на репозиторій (чи інший URL) або у вигляді архіву на довільному носієві даних. Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.

Автореферат містить описову частину, у якій розкриваються основні концепти проєкту, деталізуються завдання до виконавця, встановлюється актуальність роботи, її об'єкт та предмет, обґрунтовується вибір технологій та відзначається шлях виконання та ступінь досягнення поставлених задач, посилання на джерела інформації.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

У відповідності до технічного завдання (ТЗ) система була спроектована за архітектурою «Progressive Web Application» (PWA). У складі системи було умовно відокремлено наступні архітектурні частини:

- «Backend»,
- «Frontend»,
- «Image Serving API»,
- «Notifications»

Частина «Backend» реалізована як централізований вузол збереження та оброблення даних, що створюються та модифікуються протягом життєвого циклу інформаційної системи. Дана частина включає у себе базу даних проєкту та програмний інтерфейс (API) доступу до основних її функцій, зокрема, керування контентом, авторизації, узгодження профілів користувачів різного типу та/або різних пристроїв.

Частина «Backend» виконана на базі програмного забезпечення NodeJS та СУБД «PostgreSQL». Додатково використані програмні пакети «NestJS», «TypeORM», «PG», «Bcrypt», «Cors», «DotEnv», «ExpressJS», «JsonWebToken», «PassportJS», «Morgan», «Multer», «SQLite3», «Jest», «Prettier», «Supertest», «Ts-Jest».

Клієнтська частина системи «Frontend», призначена для надання користувачеві доступу до функцій системи через засоби візуального інтерфейсу.

Частина «Frontend» виконана на базі програмного забезпечення NodeJS та програмного пакету для створення користувацьких інтерфейсів «React». Додатково використані програмні пакети «ReduxJS», «Axios», «Formik», «Notistack», «MUI», «YUP», «Prettier».

З метою покращення безпеки, звертаючи увагу на той факт, що мобільні пристрої досить часто підключаються до незахищених мереж передавання даних, інформаційний обмін між частинами «Backend» та «Frontend», «Image Serving API» та «Frontend», «Notifications API» та «Frontend» захищений попереднім шифруванням сучасним криптоалгоритмом SHA-256 за рахунок використання протоколу HTTPS який передається через TLS, який є покращеною версію його попередника SSL. Це гарантує помірний захист від

прослуховування та від нападу «людина посередині»[1] у відкритих мережах, а також спрощує задачі автентифікації та доступу до контенту з обмеженим доступом без наявності ключів розшифрування даних, навіть за умови дискредитації інших модулів системи.

Частина «Image Serving API» реалізована як централізований вузол що забезпечує збереження, віддачу та оброблення статичних зображень в оптимізованому форматі з метою зменшення загальної кількості веб трафіку. Дана частина включає у себе базу даних завантажених зображень та програмний інтерфейс (API) доступу до основних її функцій, зокрема, віддачу, оброблення та завантаження контенту.

Частина «Image Serving API» виконана на базі програмного забезпечення NodeJS та СУБД «MongoDB». Додатково використані програмні пакети «HTTP», «DotEnv», «Mongoose», «Prettier», «Formidable», «Express», «Nodemon», «Sharp».

Частина «Notifications API» реалізована як централізований вузол що забезпечує збереження, віддачу та оброблення повідомлень в режимі реального часу. Дана частина включає у себе базу даних проєкту та програмний інтерфейс (API) доступу до основних її функцій, зокрема, віддачу, оброблення та завантаження контенту.

Частина «Notifications API» виконана на базі програмного забезпечення NodeJS та СУБД «MongoDB». Додатково використані програмні пакети «ExpressJS», «DotEnv», «Prettier», «PG», «amqplib».

РОЗДІЛ 2. ПРОЕКТУВАННЯ

2.1. Методологія розробки проекту

Так як під час розробки проекту дуже важливими була швидка комунікація між всією командою, гнучкість і швидкість у прийнятті рішень, самоорганізованість, можливість отримувати уроки з набутого досвіду, аналізувати свої успіхи та провали, щоб постійно вдосконалюватись. У зв'язку з цим, нами було прийнято рішення використовувати гнучку методологію Scrum яка сприяє цьому.

Методика Scrum - це ітеративний підхід до управління проектами та розробки ПЗ, що дозволяє командам прискорити доставку цінності клієнтам. Замість випуску всього продукту цілком, ми виконуємо роботу в рамках невеликих, але зручних інкрементів. Вимоги, плани та результати постійно проходять перевірку на актуальність, завдяки чому ми можемо швидко реагувати на зміни та тренди.

Для організації роботи в команді, а також як систему відстеження помилок, призначену для організації спілкування між користувачами, і для управління проєктом було використано онлайн сервіс Atlassian JIRA, розроблений компанією Atlassian.

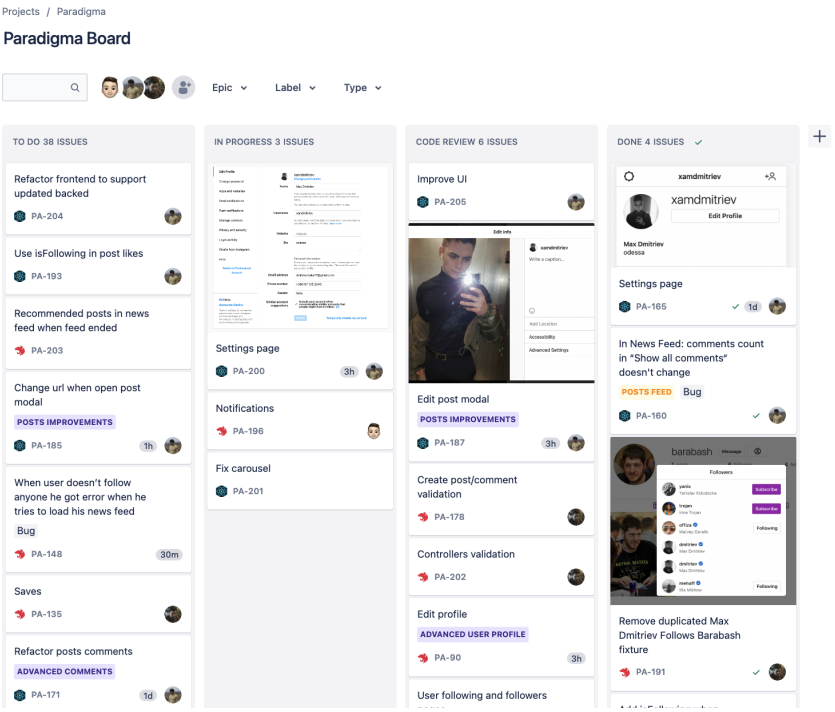

Рис 1. Загальний вигляд Kanban доски проекту.

Спершу, всі задачі створюються в беклог проекту, звідки потім, на етапі планування кожного спринта переносятся на основну доску проекту у розділ “Зробити” (див рис. 1). Для зручності, задачі у розділі “Зробити” відсортовані у порядку виконання, де найвище розташовані задачі – це ті, котрі треба зробити ближчим часом. У кожному завданні вказана область відповідальності та необхідна компетентність виконувача. В рамках цього проекту – це Backend або Frontend задачі.

Через те що обрана методика розробки передбачає обговорення продуктивності у режимі реального часу та повну прозорість робочих процесів. Робочі завдання візуально представлені на дошці Kanban, що дозволяє учасникам команди бачити стан кожного завдання у будь-який час.

Основними етапами життєвого циклу завдання є наступні колонки: “В роботі”, “На перевірці”, “Зроблено”. Кожне завдання проходить через ці етапи і тільки після цього вважається виконаним.

2.2. Управління проектом

Через те що в нашій команді є як окремо Backend та Frontend розробники, так і Full Stack розробник це призвело до проблеми конфліктів файлів. А саме конфлікти, які виникають внаслідок редагування одних і тих же файлів різними учасниками команди в рамках роботи над різними задачами і в тимчасових рамках, що перетинаються. Для вирішення цієї проблеми, нами було прийнято рішення використання Git, як розподіленої системи керування версіями файлів та спільної роботи. Це дозволило нам організувати паралельну розробку і роботу над одними і тими самими частинами проекту в одних часових рамках.

Для виявлення помилок та для забезпечення покращення загальної якості програмного коду. Нами було прийняте рішення зробити перевірку коду одним із обов’язкових етапів розробки програмного забезпечення для покращення загальної якості програмного коду та усунення помилок.

Для реалізації і спрощення перевірки кода було використано розділ “Pull Requests” в онлайн системі контроля версій GitHub. (див. рис. 2)

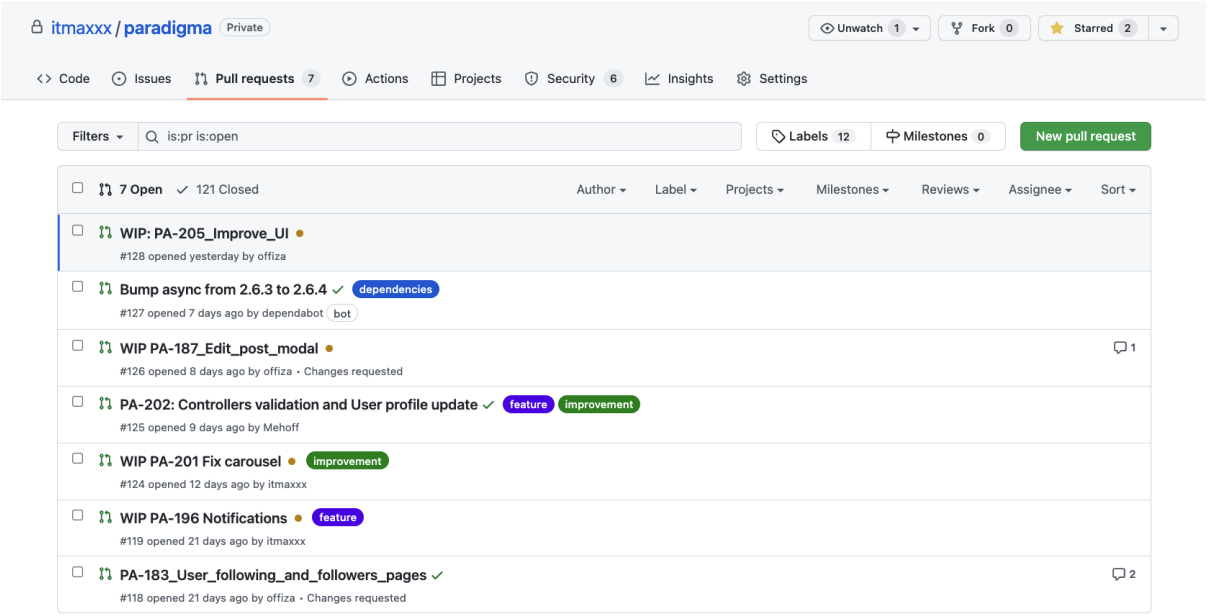

Рис 2. Розділ “Pull Requests” проекту, в системі контроля версій GitHub.

Для зручності використання, кожний Pull Request, тобто запит на злияння нового функціонала, має бути названий за зразком: “PA-[Номер завдання з доски задач] [Назва завдання]”. Для позначення задач які знаходяться “в роботі” використовується додаткове програмне забезпечення “WIP”. Для цього необхідно на початку заголовку Pull Request написати “WIP”, що з англійської розшифровується як “work in progress” і перекладается як “знаходиться в роботі”.

Додатково, для кожного Pull Request діє правило необхідності вирішення всіх конфліктів перед проведенням злияння, а також необхідно отримати хоч одну позитивну перевірку кода іншим учасником проекту.

2.3. Архітектура Backend та Frontend

За останні декілька років серед інженерних компаній, таких як Google[2], Twitter[3], Facebook[4], набуло популярності використання єдиного сховища коду або монорепозиторію. Після прийняття його як основної частини своєї інфраструктури в світових корпораціях – це викликало шквал впроваджень, навіть серед проектів з меншими командами розробників.

Бажаючи випробувати та на практиці оцінити корисність цього інженерного тренду нами було прийнято рішення організувати основний бекенд та фронтенд у вигляді монорепозіторія. Для полегшення організації єдиного

сховища коду було використано додаткове програмне забезпечення «NX Framework». Це дозволило швидко створити та налаштувати проект, а також зручно керувати програмними залежностями в майбутньому.

Основною метою використання моно репозиторія було бажання отримати можливість повторно використовувати подібні функціональні можливості та програмні інтерфейси між різним програмним забезпеченням. В нашому випадку між Backend та Frontend.

Завдяки впровадженню єдиного сховища коду ми бажали полегшити масштабний рефакторинг коду та впровадження нового функціоналу. Оскільки розробники мають доступ до всього проекту це може гарантувати, що кожна частина проекту продовжує функціонувати після рефакторингу або впровадження нового функціоналу.

Також ми очікували покращити атомарність комітів – коли проекти, які працюють разом, містяться в окремих репозиторіях, версії їх залежностей повинно синхронізувати. А в досить великих проектах керування сумісними версіями між залежностями може стати пеклом залежностей.[5] У монорепо цю проблему можна зняти, оскільки розробники можуть змінювати декілька проектів атомарно.[6]

Проект має наступну структуру:

- root
    - apps
        - backend
            - helpers
            - src
            - tests
        - paradigma
            - src
    - libs
        - common
            - src
                - lib
                - models

“apps” – директорія, де знаходиться ПЗ, в нашому випадку Backend та Frontend.

“libs” – директорія, яка містить у собі програмні пакети, код яких можно використовувати в інших проектах в рамках цього репозиторію. В нашому випадку тут містяться функції та моделі даних, які ми повторно використовуємо в Backend та Frontend.

2.4. Розподілена система завантаження, обробки та віддачі статичних зображень

Однією з основних проблем при створенні сховища статичних зображень є обмеження дискової пам'яті та обмеження швидкості читання диска, обсягу трафіку та пропускної спроможності інтернет обладнання на одному фізичному сервері.

Через те що це все проблеми вертикального масштабування, для вирішення цих проблем нами було прийнято рішення відокремлення сервісу завантаження, обробки та віддачі статичних зображень у окремий мікросервіс. Для взаємодії з мікросервісом використовується протокол HTTP, це дозволяє легко збільшувати кількість серверів при збільшенні обсягу даних контенту користувачів. Також це дозволяє балансувати навантаження між кількома серверами використовуючи сервер для балансування навантаження. Запровадження такої системи відкриває можливість горизонтального масштабування.

Детальніше про розподілену систему зображень розповідається в розділі 3.1 цієї роботи.

2.5. Розподілена система віддачі повідомлень в режимі реального часу

Перед створенням розподіленої системи збереження та віддачі повідомлень в режимі реального часу потрібно було обрати спосіб для реалізації зв'язку між клієнтом і сервером в режимі реального часу. Провівши огляд основних способів для реалізації комунікації “клієнт-сервер” в режимі реального часу, було виявлено, що основною технологією для створення

двонапрямленого повнодуплексного каналу зв'язку між сервером та клієнтом, є протокол WebSocket за стандартом RFC 6455.[7]

Під час дослідження WebSocket протоколу було виявлено, що при зростанні кількості користувачів одночасно під’єднаних до серверу використовуючи протокол WebSocket, немає ніяких проблем коли достатньо збільшити об’ем оперативної пам’яті або встановленивити кращий процессор для фізичного серверу. Але коли ми стикаємося з проблемами продуктивності сервера, через занадто велику кількість одночасно підключених користувачів виникає проблема яку можно вирішити лише за допомогою горизонтального масштабування. Тут і виникають перші проблеми з WebSocket через використання механізму сессій які прив'язані до одного серверу. Отже, щоб мати змогу запустити кілька екземплярів ПЗ працюючих з використанням протокола WebSocket, нам потрібно запровадити балансувальник навантаження та зв’язок між усіма WebSocket серверами.

Для вирішення цієї проблеми, було прийнято рішення використовувати Redis. Це розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання на бажання користувача. Це ПЗ з відкритим сирцевим кодом написане на ANSI C. Redis реалізує паттерн проектування публікація-підписка[8], при якому створюється канал, повідомлення з якого поширюються клієнтам, що підписані на канал[9]. В нашому випадку клієнти – це сервери які використовують WebSocket з’єднання, що дозволяє налаштувати комунікацію між серверами.

Детальніше про реалізацію розподіленої системи повідомлень розповідається в розділі 3.4 цієї роботи.

2.6. Авторизація користувачів при використанні сервіс-орієнтованої архітектури

Так як інфраструктура проекту допускає використання великої кількості серверів, які найчастіше не є однією фізичною машиною і можуть бути розташовані в різних частинах світу, не мати спільних жорстких дисків і знаходитися в різних мережах. Це робить неможливим використання механізму сесій, як фізичних файлів, для аутентифікації клієнта на різних серверах.

Для реалізації можливості системи авторизації яка не буде залежати від одного фізичного сервісу нами було прийнято рішення використовувати JSON Web Token. Це стандарт токена доступу на основі JSON за стандартом RFC 7519[10]. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи. Все що потрібно для між серверної авторизації - використання загального секретного ключа використовуваного при генерації та розшифруванні токена.

Для забезпечення безпеки, від підроблення токенів, при підписанні заголовка та корисного навантаження виконується алгоритм HMAC з використанням SHA-256.

РОЗДІЛ 3. ВПРОВАДЖЕННЯ

3.1. Впровадження розподіленої системи віддачі, збереження та обробки зображень

Згідно з рекомендаціями та технічним завданням, архітектура сервісу зображень виглядає наступним чином:

- Backend - NodeJS, з додатково використаними бібліотеками «HTTP», «DotEnv», «Mongoose», «Prettier», «Formidable», «Express», «Nodemon», «Sharp».
- База данних - MongoDB.
- Проксі сервер - Nginx.

Файлова структура сервісу зображень виглядає наступним чином

- root
  - src
    - www
      - uploads
    - controllers
    - models
    - services
    - types
    - utils

Для забезпечення максимально можливої для NodeJS швидкості веб серверу було прийнято рішення використовувати нативний NodeJS модуль http. [11] Це дало можливість оптимізувати час віддачи зображень та максимальну пропускну можливість.

Також, великий приріст продуктивності веб серверу, в разі розміщення на багатоядерному сервері, вдалось зробити завдяки кластеризації веб сервера. Для цього була додатково використанна бібліотека “cluster”.

```
import { cpus } from 'os';
import cluster from 'cluster';

// ...
```

```
if (cluster.isPrimary) {
  console.log(`Primary process ${process.pid} is running`);

  for (let i = 0; i < cpus().length; i++) {
    cluster.fork();
  }

  cluster.on('exit', () => {
    console.log(`Worker process ${process.pid} died`);
    cluster.fork();
  });
} else {
  console.log(`Worker process ${process.pid} is running`);

  // Launch web server
}
```

Як можно побачити, з наведеного вище коду, при запуску веб серверу, ПЗ отримує кількість фізичних ядер процесору, та створює на кожному з них дочірній інстанс. У випадку припинення роботи дочірнього процесу, він запуститься заново в автоматичному режимі.

Всі завантаженні та оброблені зображення містяться в директорії uploads.

Модель зображення в базі даних виглядає наступним чином:

```
class ImageModel {
  @prop()
  public _id: Types.ObjectId;


  @prop()
  public originalExtension: string;


  @prop({ default: null })
  public deleted: boolean;
}
```

Для видалення зображень було обрано механізм м’якого видалення даних – це шаблон проєктування, при якому сутність не видаляється зі

сховища, а лише помічається як видалена. Це дозволило пришвидшити процес видалення даних, запобігає випадкове видалення даних та дозволяє відновити видаленні дані. Із недоліків використання цього шаблону проектування можна виділити зростання розміру хранилища та обов'язковість додавання додаткового фільтру для кожного запиту.

3.2. Процес створення публікації та завантаження зображень

З метою покращення користувацького досвіду від використання ПЗ нами була розроблена форма створення публікації з трьома основними етапами:

1. Вибір зображення із файлової системи користувацького пристрою (див. рис. 3)
2. Редагування обраного зображення (див. рис. 4)
3. Форма створення публікації (див. рис. 5)

На кожному з етапів передбачено повернення на попередній етап для зручності користувача, так само як і відображення поточного етапу, що можна побачити у верхній частині модального вікна.

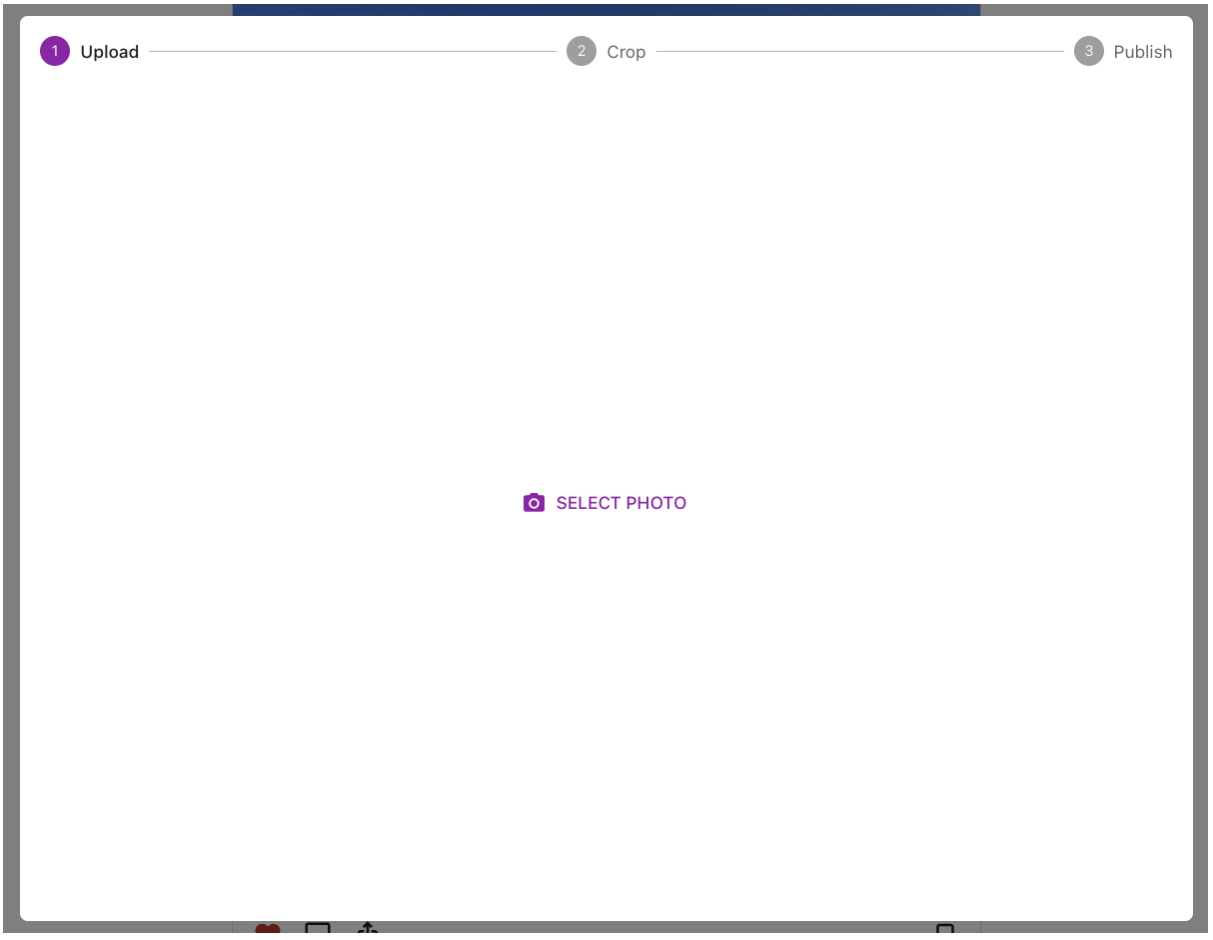

Рис 3. Модальне вікно з формою для вибору зображення

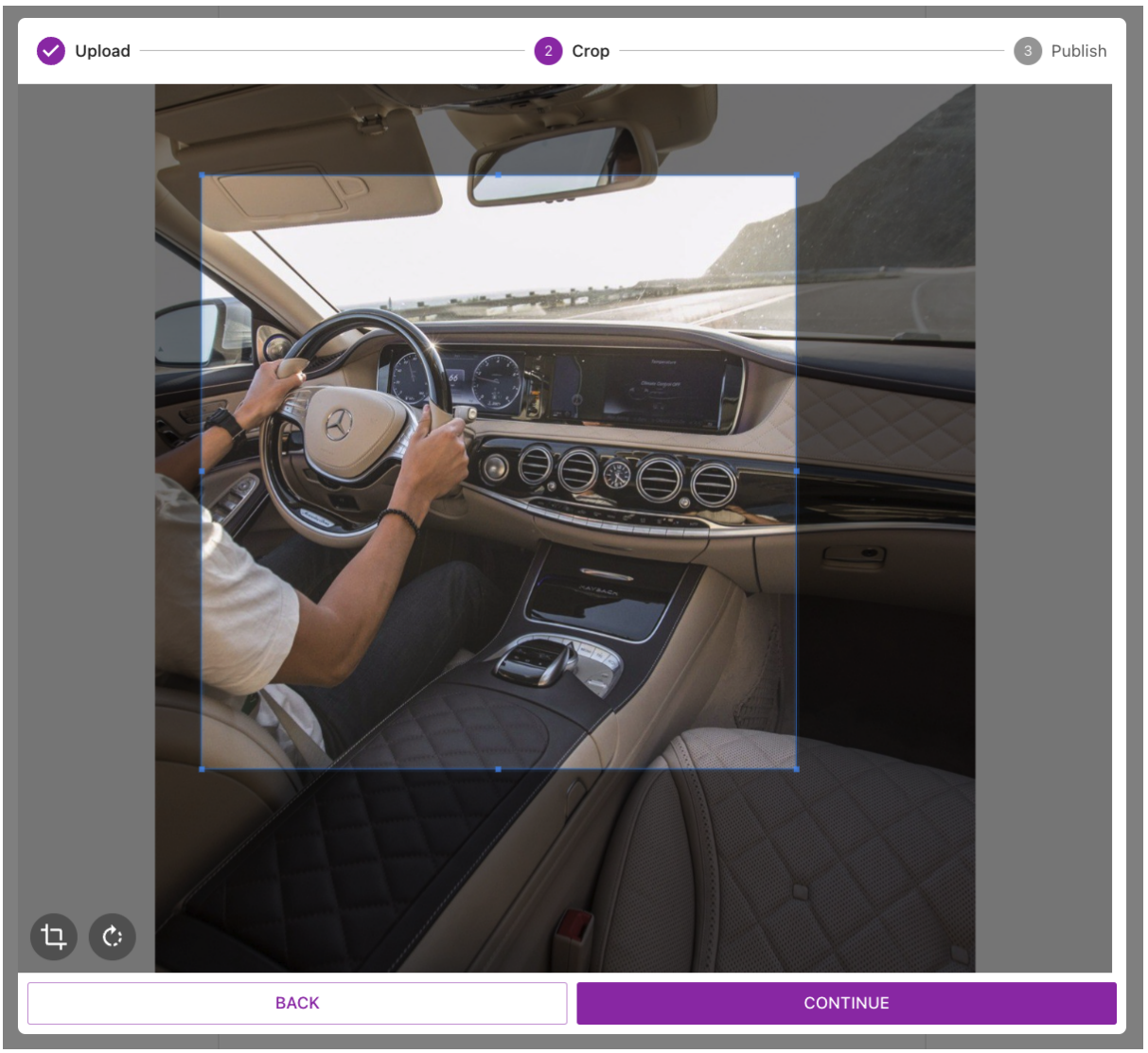

Рис 4. Модальне вікно з формою редагування зображення

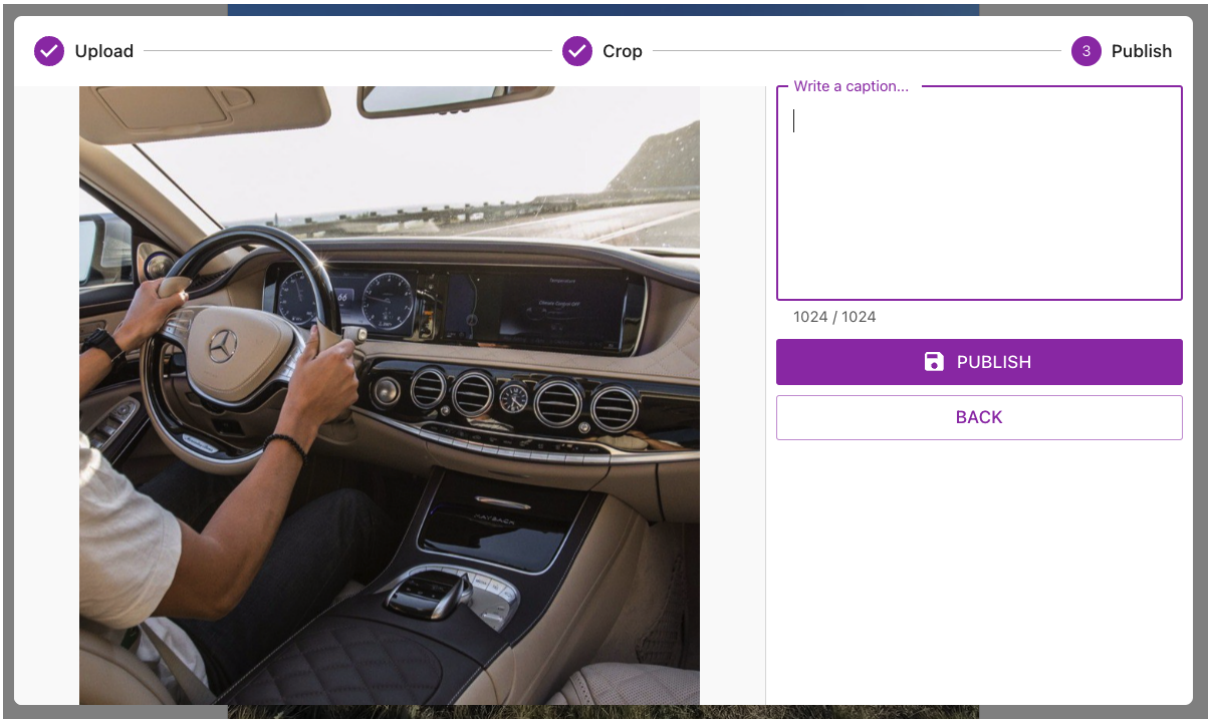

Рис 5. Модальне вікно з формою створення публікації

Через те, що більшість користувачів ПЗ будуть використовувати мобільні пристрої, з здебільшого обмеженою кількістю інтернет-трафіку. Було прийнято рішення стискати зображення перед їх завантаженням та відправкою на сервер для економії інтернет трафіку.

Процес завантаження зображення на сервер виглядає наступним чином: (див. рис. 6)

1. Спочатку зображення завантажується на основний Backend ПЗ
2. Після цього Backend завантажує зображення на сервер збереження, обробки та віддачі зображень
3. В свою чергу, після вдалого завантаження, сервіс зображень повертає унікальний ідентифікатор завантаженого зображення, котрий Backend сохраняє для подальшого отримання доступу до зображення.

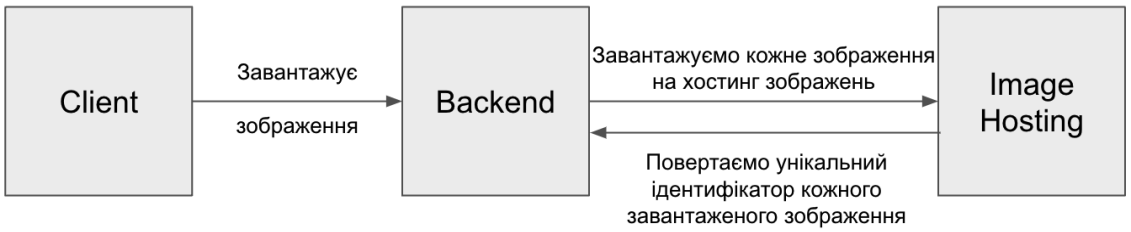

Рис 6. Схема завантаження зображення

3.3. Оптимізація та обробка зображень

У зв'язку з тим, що більшість користувачів ПЗ використовують мобільні пристрої, які, як відомо, не завжди мають високошвидкісне підключення до мережі інтернет, а також частіше за все мають ліміт на використання інтернет трафіку.

Одним із ефективних засобів вирішення цієї проблеми є використання алгоритмів стиснення даних до зображень, що зберігаються в цифровому виді. В результаті стиснення зменшується розмір зображення, що зменшує час передачі зображення по мережі і економить простір для зберігання. Стиснення зображень розділяють на стиснення з втратами якості і стиснення без втрат. Стиснення без втрат більш підходить для штучно побудованих зображень, таких як графіки, іконки програм, або для спеціальних випадків, наприклад, якщо зображення призначені для подальшої обробки алгоритмами розпізнавання зображень. Алгоритми стиснення з втратами при збільшенні степені стиснення, як правило, породжують добре помітні людському оку артефакти.

Після проведення огляду наявних методів стиснення зображень було обрано використовувати WebP – формат ущільнення зображень з втратами і без втрат якості, запропонований компанією Google Inc. у 2010 році. Заснований на алгоритмі стиснення нерухомих зображень (ключових кадрів) з відеокодека VP8.

Використані в WebP технології стиснення з втратами дозволяють домогтися скорочення розміру файлу на 25–34%, порівняно з файлами JPEG аналогічної якості, і на 26% в режимі кодування без втрат у порівнянні з максимальним рівнем стиснення PNG[14]. Тому WebP може виступати як повноцінна заміна форматів JPEG, GIF і PNG, забезпечуючи при цьому вищий ступінь стиснення і швидкість декодування.

Однією з проблем при використанні формату ущільнення зображень WebP є відсутність підтримки старими браузерами. Для вирішення цієї проблеми всі завантажені користувачами зображення потрібно конвертувати в формат який підтримує більшість браузерів – JPG. Нами було прийнято рішення використовувати HTML-елемент <picture> який є контейнером для одного або більше елементів <source> і одного елемента <img> для забезпечення оптимальної версії зображення для різних розмірів екрану. Браузер розгляне кожен із дочірніх елементів <source> і вибере один, що відповідає кращому збігу; якщо збігів серед елементів <source> не буде знайдено, то буде обраний файл, вказаний атрибутом src елемента <img>. Потім вибране зображення відображається у просторі, зайнятому елементом <img>.

Використання HTML-елементу <picture> також дозволяє браузеру вибрати оптимальне зображення, user agent аналізує атрибути srcset, media, та type елемента <source> і вибирає сумісне зображення, яке найкраще відповідає поточному макету сторінки, характеристикам пристрою відображення тощо.

Після огляду наявних модулів обробки зображень було визначено, що найшвидшою є бібліотека обробки та конвертації зображень Sharp.[15] Саме вона і була використана для обробки зображень.

3.4. Впровадження розподіленої системи збереження, обробки та віддачі повідомлень в режимі реального часу

Згідно з рекомендаціями та технічним завданням, архітектура сервісу зображень виглядає наступним чином:

- Backend - NodeJS, з додатково використаними бібліотеками «Express», «DotEnv», «AMQLIB», «Prettier», «Morgan», «JsonWebToken», «Nodemon», «pg», «Redis», «Socket.IO», «UUID».
- База данних - Apache Cassandra.
- Проксі сервер - Nginx.

Одним із основних завдань при розробці системи повідомлень було зробити її здатною до гнучкого масштабування, адже з ростом кількості користувачів дуже швидко зростає і кількість повідомлень які вони створюють. Для забезпечення можливості гнучкого горизонтального масштабування сервісу

повідомлень було прийнято рішення балансувати навантаження між усіма працюючими сервісами повідомлень.

Для комунікації між основним бекендом та сервісами повідомлень було прийнято рішення створити чергу повідомлень. Для реалізації черги було прийнято використовувати RabbitMQ, це платформа, що реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP згідно з ISO/IEC 19464[16]. Завдяки використанню черги повідомлень ми маємо можливість створювати та додавати до загальної системи повідомлень нові сервера для балансування навантаження між ними. Загальна схема взаємодії основного бекенду, черги повідомлень та сервісів повідомлень зображена на рисунку 7.

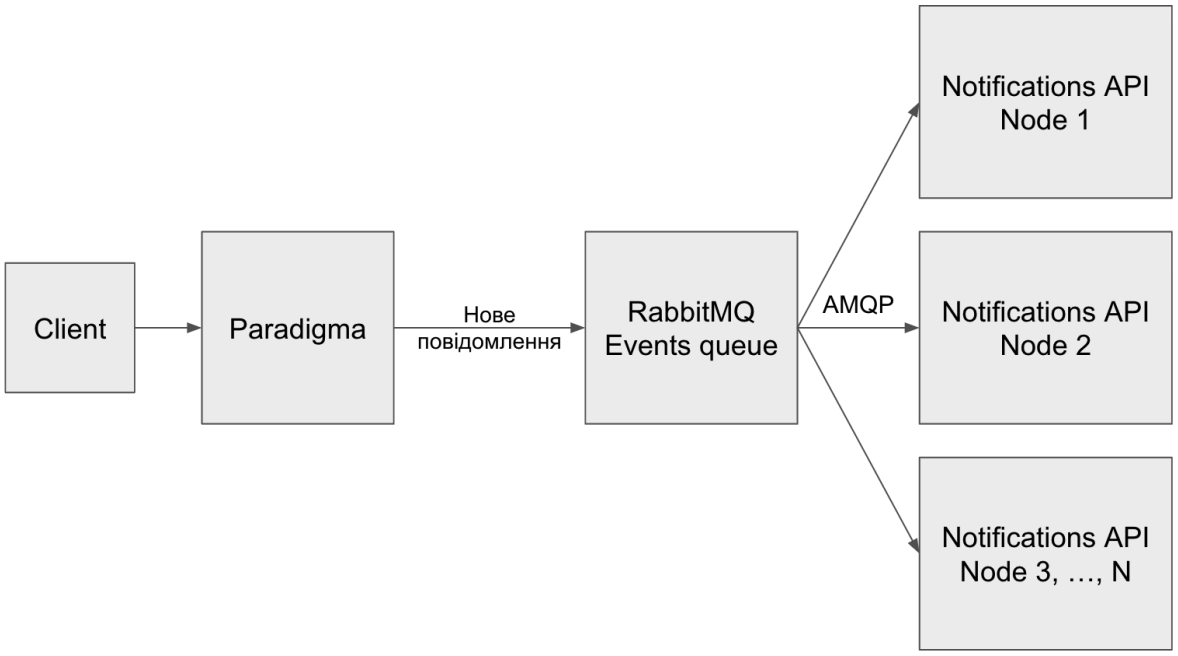

Рис 7. Схема взаємодії основного бекенду, черги повідомлень та сервісів повідомлень.

Для балансування навантаження між вузлами, до яких користувачі підключенні з використанням протоколу WebSocket, було використано Redis, як

Алгоритм роботи сервісу виглядає наступним чином:

1. Користувач №1 здійснює дію. Наприклад: підписується на оновлення користувача №2.

2. Основний бекенд обробляє запит та додає нове повідомлення до черги повідомлень.
3. Один із вузлів отримує повідомлення для обробки:
    a. Зберігаємо сповіщення в БД.
    b. Якщо користувач №2 онлайн і підключений до поточного вузла, надсилає йому сповіщення.
    c. Якщо користувач №2 не підключений до поточного вузла з використанням протоколу WebSocket, але він онлайн, надсилаємо подію до Redis. (див рис. 8)

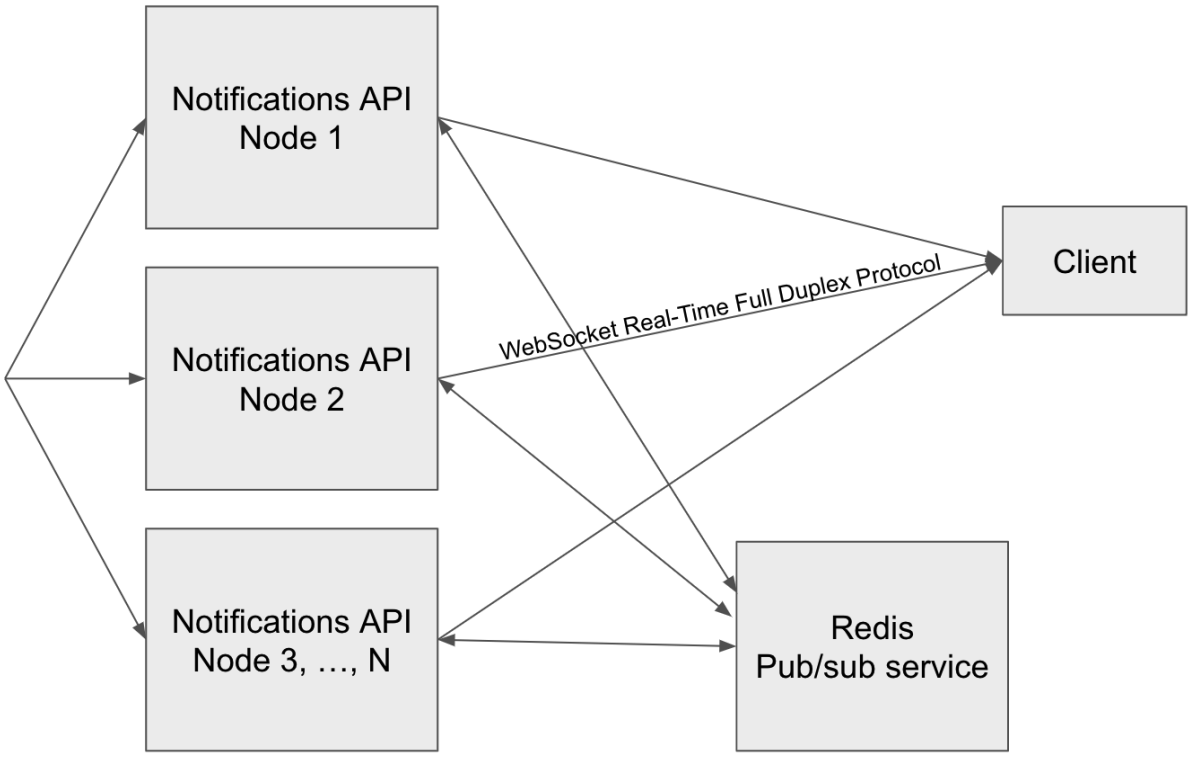

Рис 8. Схема взаємодії сервісів повідомлень

РОЗДІЛ 4. РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

Результатом випробувань є розроблений веб-доданок, основною функцією якого є поширення статичного контенту. Додатковими функціями системи є реєстрація, авторизація та автентифікація, засоби створення контенту, відгуки, пошук користувачів, повідомлення, особистий кабінет користувача, розподілена система збереження, віддачі та обробки зображень, розподілена система обробки, збереження та видачі повідомлень в режимі реального часу.

Створено ПЗ має багато переваг серед конкурентів, серед яких вважаю важливим виокремити механізм оптимізованої обробки файлів зображень, що зменшує обсяг пакетів даних які передаються по мережі, який потребував розробки стороннього сервісу, однак це значно покращує досвід користувача від використання ПЗ, за рахунок зменшення кількості інтернет трафіку та пришвидшення завантаження зображень.

Під час виконання роботи було:

- Проведено огляд та впроваджено інформаційні технології для ефективного розроблення програмного забезпечення. Було встановлено напрямки розвитку та удосконалення командної розробки ПЗ.
- Проведено аналіз модулів обробки та алгоритмів стиснення зображень для забезпечення найбільшого ступеня стиснення і швидкості декодування для зменшення розміру зображення, що зменшує час передачі зображення по мережі і економить простір для його зберігання.
- Проведено дослідження, розроблення та впровадження інтернет-технології для побудови сервіс-орієнтованих систем та систем розподіленої обробки інформації.
- Проведено аналіз наявних мережевих протоколів для реалізації комунікації між різними сервісами та комунікації в режимі реального часу. Було встановлено їх слабкі места та встановлено напрямки удосконалення та способи їх масштабування.
- Проведено випробування ПЗ за різних умов та для різних пристроїв.

- Складено програмне забезпечення, в якому було реалізовано результати, досягнуті у попередніх задачах.

Використання моно репозиторію для основної частини ПЗ дозволило:

- Повторно використовувати код - подібні функціональні можливості або програмні інтерфейси можна абстрагувати в спільні бібліотеки та безпосередньо включати в проекти без потреби в менеджері пакетів залежностей.
- Зробити коміти атомарними. Коли проекти, що працюють разом, містяться в окремих репозиторіях, необхідні релізи, що синхронізують версій одного проекту з іншим для їх спільної роботи. Кожний коміт - це працююча версія програми.
- Спростити керування залежностями. Оскільки всі залежності, на які посилаються проекти, існують в одній кодовій базі.
- Організувати співробітництво між різними командами (Backend та Frontend). Через те, що одні команди можуть поліпшувати проекти, над якими працювали інші команди.

Для спрощення та пришвидшення розгортання ПЗ, усі сервіси, бази даних, та черги повідомлень було поміщено в ізольовані Linux-контейнери з використанням Docker. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Для спрощення та пришвидшення розробки ПЗ, для основних частин проекту, було реалізовано паттерн безперервної інтеграції та безперервного розгортання. Для реалізації цього паттерну, для автоматизації розгортання Backend було використано хмарну PaaS-платформу Heroku, для автоматизації розгортання Frontend було використано хмарну платформу Vercel.

Використання черги повідомлень для реалізації комунікації між сервісом повідомлень і основним бекендом дозволило:

- Під час пікових навантажень можливо швидко створювати та додавати до загальної системи обробки повідомлень нові сервіси обробки повідомлень та балансувати між ними навантаження.
- При зменшенні навантаження та кількості користувачів онлайн можливо тимчасово вимкнути деякі сервера для зменшення

- Уникнути додаткових налаштувань при додаванні нового серверу для обробки повідомлень. Сервіс під'єднується до загальної черги повідомлень та починає автоматично обробляти нові повідомлення.
- У разі системних збоїв на одному, або навіть усіх серверах із сервісами повідомлень, але при умові що сервер з чергою повідомлень працює в штаному режимі, всі необробленні повідомленя будуть збереженні та обробленні при відновленні роботи одного чи більше серверів з сервісом повідомлень.

ВИСНОВКИ

У сучасних мережевих інформаційних технологіях все частіше використовуються системи розподіленої обробки та збереження даних. Вони дозволяють підвищити ефективність задоволення інформаційних потреб користувачів, забезпечити гнучкість і оперативність прийнятих ним рішень і ін.

Подібні системи дають можливість користувачам звертатися до будь-яких даних які в них зберігаються і часом надають нові, раніше невідомі, можливості роботи з інформацією. При цьому виникають нові проблеми, які вирішуються шляхом використання нових технологій розглянутих в цьому дослідженні.

В рамках цієї роботи проведено огляд інформаційних систем, програмного забезпечення з використанням сервісно-орієнтованих систем, систем розподіленої обробки та збереження даних. У якості прикладу обрано для реалізації систему поширення статичного контенту (соціальна мережа).

Під час виконання роботи були використані методи гнучкого масштабування. Завдяки цьому користувачі веб-додатку мають змогу швидкого доступу до великої кількості даних, можливість у поширенні, створенні та перегляду наявного контенту, який надає засоби перегляду, обговорення та взаємодії без необхідності завантаження.

Систему було випробувано шляхом розміщення на хмарних серверах для проведення тестових сеансів комунікації. Результати випробування підтвердили загальну працездатність створеної системи.

Перспективою подальших досліджень може стати:

- Розробка сервісу комунікації користувачів у режимі реального часу
- Впровадження сервісу збирання та обробки метрик користувачів
- Створення алгоритмів розумної користувацької стрічки
- Використання штучного інтелекту для обробки та аналізу статичного контенту

Це дозволить масштабувати ПЗ та покращити користувацьких досвід.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Атака «людина посередині», від 26 травня 2022; URL: https://uk.wikipedia.org/wiki/Атака_«людина_посередині»
2. Levenberg Josh, Rachel Potvin, "Why Google Stores Billions of Lines of Code in a Single Repository". Communications of the ACM. 2018. URL: https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext
3. Dorothy Ordogh, Pants and Monorepos, 2018. URL: https://www.youtube.com/watch?v=IL6LBWNi3fE
4. GOODE, DURHAM, "Scaling Mercurial at Facebook – Facebook Code", 2018. URL: https://engineering.fb.com/2014/01/07/core-data/scaling-mercurial-at-facebook/
5. Aimee Lucido, “Uber Technology Day: Monorepo to Multirepo and Back Again”, 2018. URL: https://www.youtube.com/watch?v=lV8-1S28ycM
6. Santacroce Ferdinando, Olsso Aske, Voss Rasmus, Narebski, Jakub, Git: Mastering Version Control. Packt Publishing Ltd., 2016, с. 756.
7. I. Fette, Google, Inc., A. Melnikov, Isode Ltd., The WebSocket Protocol, 2011; URL: https://datatracker.ietf.org/doc/html/rfc6455
8. Публікація-підписка (шаблон проєктування), вересень 2021; URL: https://uk.wikipedia.org/wiki/Публікація-підписка_(шаблон_проєктування)
9. Офіціальна документація ПЗ Redis, Redis Pub/Sub; URL: https://redis.io/docs/manual/pubsub/
10. M. Jones, Microsoft, J. Bradley, Ping Identity, N. Sakimura, NRI, JSON Web Token (JWT), 2015; URL: https://datatracker.ietf.org/doc/html/rfc7519
11. Alex. M., Node.js performance vs Hapi, Express, Restify, Koa & More, 2017; URL: https://raygun.com/blog/nodejs-vs-hapi-express-restify-koa/
12. Мартін Клепман, "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems", 2017 р., 611 с., ISBN: 978-1449373320
13. Сєм Ньюман, "Building Microservices: Designing Fine-Grained Systems", 2021 р., 612 с., ISBN: 978-1492034025

14. Jyrki Alakuijala, Vincent Rabaud, "Lossless and Transparency Encoding in WebP", 2017p.; URL: https://developers.google.com/speed/webp/docs/webp_lossless_alpha_study#results
15. Ловел Фюллер, “Sharp: High performance Node.js image processing”, 2021p.; URL: https://sharp.pixelplumbing.com/performance
16. “ISO/IEC 19464:2014(en) Information technology — Advanced Message Queuing Protocol (AMQP) v1.0 specification”, 2014p.; URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:19464:ed-1:v1:en