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»

Розробники:

№ з/п	ПІБ	Роль
1	**Горелік М.А.**	Розробка (frontend)
2	**Дмітріев М.Ю.**	Розробка (fullstack)
3	**Міхов І.С.**	Розробка (backend)

Ментори:

№ з/п	ПІБ, посада	Напрямок
1	Самойленко Д.М.	Розробка ПЗ

АНОТАЦІЯ

Робота присвячена дослідженням, розвиненню та впровадженню користувальницьких інтерфейсів, головним призначенням яких є взаємодія між користувачами з урахуванням їх пристроїв. Додаткові функції системи включають реєстрацію, авторизацію, аутентифікацію, налаштування профілю, вибору теми інтерфейсу додатку, динамічно-згенерований контент, пошук по додатку, месенджер, завантаження користувацького контенту, тощо.

Об'єктом дослідження у рамках даної роботи виступає алгоритм відображення контенту в залежності від стану ПЗ, динамічне відтворення та відображення даних та станів що змінюються або відновлюються в процесі користування ПЗ. Предметом дослідження була обрана комплексна система обміну користувацьким контентом.

У якості задач дослідження було встановлене наступне:

- провести огляд інформаційних систем, що мають функціонал аутентифікації користувачів, роботу з файлами, метриками користування ПЗ користувачами та їх застосування у генерації контенту
- визначити програмні засоби (інтерфейси API), які дозволяють отримувати дані про користувача з третіх сервісів, встановити їх особливості та ступінь вживаності для поставлених завдань.
- скласти програмне забезпечення, реалізувавши у ньому результати, досягнуті у попередніх задачах, провести випробування за різних умов та для різних пристроїв.
- впровадити у систему програмні засоби захисту даних для покращення показників інформаційної безпеки створеної системи.

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

При складанні звіту використано 16 посилань на друковані та електронні джерела інформації.

ЗМІСТ

ВСТУП

З розвитком і поширенням інформаційних технологій у людства з'являється багато можливостей для комунікацій, поширенню інформації, поширенню реклами, відпочинку в інтернеті, інтернет знайомств, блогінгу, інше. Стратегія розвитку інформаційного суспільства в Україні визначає, що «комунікаційні технології є необхідним інструментом соціально-економічного прогресу, 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, у світі ця цифра трохи менше і складає 60%. З кожним роком ця цифра збільшується мінімум на 4%.

Наразі немає одного стандарту інтернет пристроїв який був би зазначений певними законодавчими органами. Тому однією з важливих та основних проблем наразі являється створення правильного і функціонального адаптивного зображення сторінки - коректне відображення контенту для різних типів пристроїв.

У той же час мультифункціональні додатки в яких є більше можливостей отримують більший успіх. Додаток, що створюється, має проєктну назву “Paradigma” і являється мультифункціональним додатком що має можливості для швидкого старту.

Детальне технічне завдання до проекту наводиться далі. У рамках даного звіту розглядається клієнтська частина, що являє собою веб додаток. Звіт складається з шести розділів у яких описується загальна характеристика системи, деталі створення клієнтської частини та результати її розгортання та тестування. Висновки відбивають результати, що досягнуті у рамках даної частини проекту, загальний висновок є сукупністю звітів усіх учасників проекту.

Вимоги до функціоналу. Обов'язкова частина

Загальний опис.

Програмне забезпечення (система поширення статичного контенту), 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.

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 - Node.js, Typescript, Nest.js, PostgreSQL, TypeORM, Passport.js

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

Преамбула

Загальна частина серверної розробки з якої починається робота над проектом - вибір платформи завдяки якій буде виконуватися розробка. Для проекту “Paradigma” одними з найважливіших характеристик системи є:

- швидкість відповіді на запити клієнта серверу
- швидкість відповіді на запити серверу до бази даних
- оптимізоване завантаження та скачування файлів з файлового серверу
- стискання файлів та інформації у відповіді сервером
- швидкий пошук

Зважаючи на ці характеристики був зроблений вибір на користь JavaScript оточення побудованого на JavaScript–рушієві Chrome V8 - Node.js. На початок 2021 року Node.js є лідером з кількості розробників які використовують поруч з основною для себе мовою програмування. Така статистика є логічною спираючись на простоту завантаження серверних додатків на більшість хостинг-сервісів. Node.js використовує JavaScript як основну мову програмування - яка є однією з найпопулярніших мов упродовж багатьох років, і тренд якої не спадає з часом. Спроможність використовувати одну мову для веб та серверної розробки є одним з ключових факторів популярності Node.js.

Бек-енд частина будь-якого сервісу повинна бути невід’ємно пов'язаною з його фронт-енд частиною незважаючи на те що технічно ці частини розробляються різними розробниками. У команді потрібно завжди проектувати виконання будь-якого функціоналу з обох сторін - комплексно. Опис будь-якого завдання виконується з виділенням окремих підзадач для фронт та бек-енду відповідно.

Як приклад комплексного завдання можна зазначити функціонал обробки помилок. Сервер повинен виконувати окремі дії для збереження своєї

стабільної роботи незалежно від вхідних даних чи кількості одночасних звернень. Якщо сервер стикається з помилкою - він має:

- зберегти життєдіяння та свою стабільність
- коригувати алгоритм обробки запиту, врахувавши всі ймовірні помилки
- докладно довести до розробника що сталося за допомогою системи логування, який був контекст системи під час помилки
- у відповіді на запит зазначити код помилки, та користувацько-доброзичливий текст помилки

Як можна побачити - єдиний сервіс незважаючи на кількість різних його частин має функціонувати як єдиний організм, з цим гаслом наша команда приступила до виконання завдання.

РОЗДІЛ 1. База даних - вибір та проектування

Перш ніж почати писати код потрібно підібрати необхідні технології на базі яких починати розробку. База даних є важливою частиною будь-якого сервісу. Майже кожна дія користувача має бути зафіксована та використана для покращення його взаємодії з ПЗ. Бази даних бувають різних типів, але у своїй більшості на стан 2022 року найпопулярнішими рішеннями є документно-орієнтована та реляційні. На початку розробки проекту для бази даних була обрана документна база даних - MongoDB.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати

горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових серверних машин.

Після доповнення функціоналу і ускладнення запитів пошуку та запису до бази даних було помічено деградацію швидкості відповідей. Була проведена аналізаційна робота по тестуванню БД великою кількістю запитів, яка показала що документо-орієнтована система не підлягає запитам проекту. У якості альтернативи MongoDB було застосовано реляційну базу даних PostgreSQL.

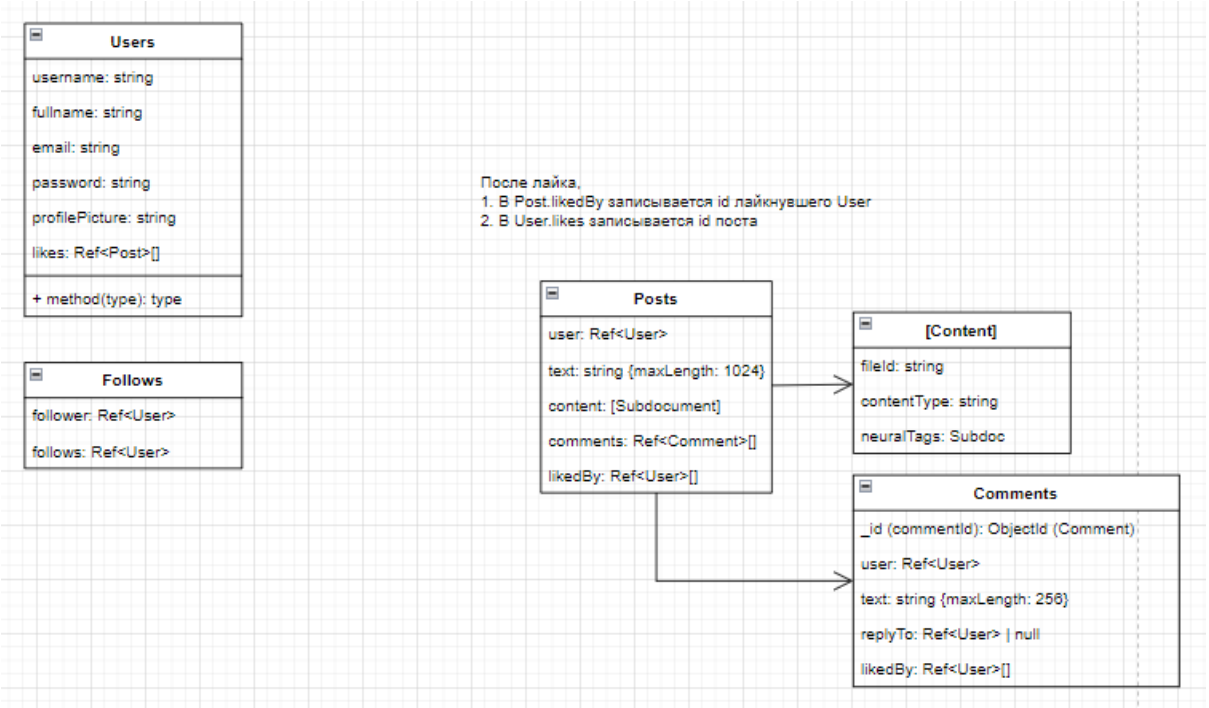
PostgreSQL - об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite). Прототип був розроблений в Каліфорнійському університеті Берклі в 1987 році під назвою POSTGRES, після чого активно розвивався і доповнювався. В червні 1990 року з'явилась друга версія із переробленою системою правил маніпулювання та роботи з таблицями, у 1991 році — третя версія, із доданою підтримкою одночасної роботи декількох менеджерів збереження, покращеним механізмом запитів і доповненою системою внутрішніх правил. В цей час POSTGRES використовувався для реалізації великих систем, таких як: система аналізу фінансових даних, пакет моніторингу функціональності потоків, база даних відстеження астероїдів, система медичної інформації, кілька географічних систем. POSTGRES також використовувався як навчальний інструмент в декількох університетах. 1992 року POSTGRES став головною СКБД наукового комп'ютерного проекту Sequoia 2000. 1993 року кількість користувачів подвоїлась. Стало зрозуміло, що для підтримки й подальшого розвитку необхідні великі витрати часу на дослідження баз даних, тому офіційно проект Берклі було зупинено на версії 4.2. 1994 року Andrew Yu і Jolly Chen додали інтерпретатор мови SQL, вдосконалили сирцевий код і виклали в Інтернеті свою реалізацію під назвою Postgres95. 1996 року програмний продукт було перейменовано на PostgreSQL із початковою версією 6.0. Подальшою підтримкою й розробкою займається група спеціалістів у галузі баз даних, які добровільно приєднались до цього проекту.

Після повторних навантажувальних тестів використовуючи PostgreSQL у якості основної бази даних - результати нас задовольнили. У якості мови доступу до даних PostgreSQL використовує SQL.

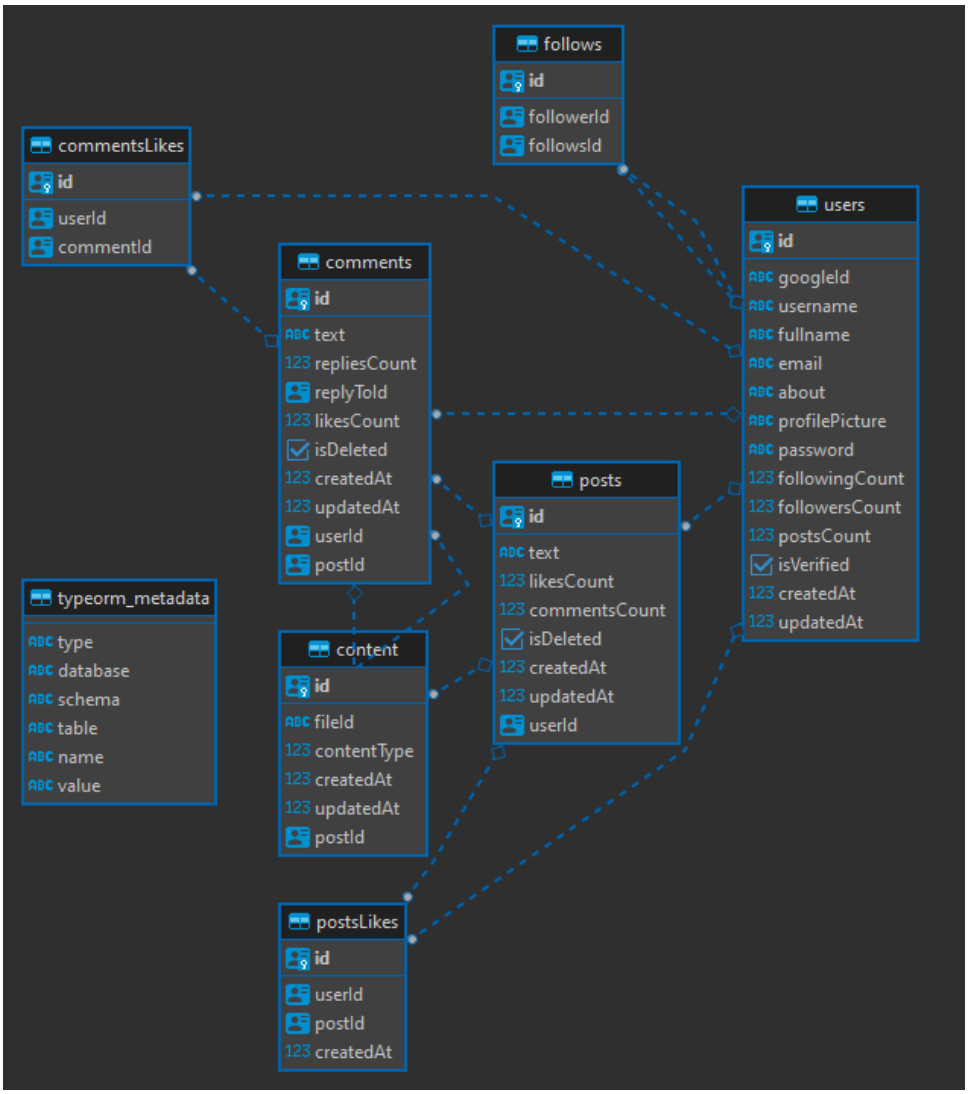
SQL — це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення і вилучення даних за допомогою використання системи керування і адміністративних функцій.

На сьогоднішній день SQL широко використовується у продакшн-коді, але для більш-менш простих запитів до бази даних використовуються ORM системи - спосіб предствити відношення в базі даних у вигляді колекцій та об’єктів у площі платформи на якій виконується розробка ПЗ. В об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна навести адресну книгу, яка містить список людей разом з кількома телефонами і кількома адресами. В термінах об'єктно-орієнтованого програмування вони будуть представлені об'єктами класу «Людина», які міститимуть такі атрибути: ім'я, список (або масив) телефонів і список адрес. В якості ORM була використана система TypeORM. Завдяки поширеності та підтримці проекту зі сторони розробників та інтеграцією “з коробки” з основним бек-енд фреймворком проекту - Nest.js ми зробили вибір у сторону цієї технології.

Перш ніж почати працювати з базою даних - потрібно виконати початкове проектування відношень сутностей. Перша версія виглядала наступним чином:



У зв’язку з поширенням функціоналу та технічних оптимізацій, фінальна модель відношень виглядає наступним чином:



РОЗДІЛ 2. Nest.js у якості бек-енд фреймворку

Сучасна кількість варіантів основ для розробки бек-енд REST API дуже великий. Майже кожна сучасна мова програмування має у собі задатки для розробки фреймворків завдяки яким інші програмісти можуть розробляти свої продукти. Node.js також має багатий вибір фреймворків для розробки бек-енду, з найпопулярніших:

- express
- fastify
- Nest.js
- koa
- та інші

Насамперед основною причиною вибору Nest.js у якості бек-енд фреймворка була зацікавленість у увазі розробника до архитектури майбутніх API. Структура проекту та інструменти взаємодії дозволяють легко поширювати проект і не заплутуватися під час його підтримки. Завдяки використанням патерну проектування Dependency Injection та іншим архітектурним паттернам - розробникам вдалося розбити архітектуру майбутніх проектів на індивідуальні модулі, які можно імортувати один в одного тим самим запозичувати доступ до окремих функціональностей окремих модулів.

Велика проблема інших фреймворків полягає у недостатній увазі до архітектури майбутніх проектів на їх платформі. Nest.js виграє у всіх основних конкурентів саме за цією характеристикою, пропонуючи Angular-like строгу архітектуру, до якої новим розробникам на Nest.js треба звикнути для отримання найбільшої продуктивності.

Nest.js має свою CLI (Command Line Interface) - программу з інтерфейсом доступа через командну строку, яка дозволяє інтегрувати, видаляти, доповняти модулі проекту, а також полегшує компіляцію та деплой на хостинг-сервіси. Nest.js “з коробки” має більшість адаптованих для своєї платформи пакетів-бібліотек для гнучкої розробки проектів різного об’єму та направлень. Nest.js має налагоджену підтримку Typescript, мови програмування яка доповнює простий JavaScript типізацією задля уникнення більшості помилок зв'язаними з динамічністю типів JavaScript. У великих проектах Typescript - це хороший вибір, бо з ростом складності проекту прослідкувати всі типи, джерела помилок та процеси буде дуже складно.

РОЗДІЛ 3. Валідація вхідних даних на сервері

Дуже важливо перевіряти які дані надходять до серверу. Некоректні дані можуть погано по-різному вплинути на систему в цілому:

- зміна даних на нелогічні значення
- доступ до приватних даних
- знищення даних
- SQL-ін’єкції
- надмінне завантаження сервера
- доступ до приватного функціоналу тощо.

Для того щоб зменшити вірогідність таких сценаріїв було використано модуль “class-validator” який займається валідацією даних які приходять до серверу з клієнтів. Як це працює: наприклад наш сервер має можливість реєстрації нових користувачів. Ми знаємо що користувач повинен надіслати адрес електронної пошти, пароль, та за бажанням - ім’я користувача. Наша задача - зробити класс, який буде являти собою той самий об’єкт запиту користувачем з зазначеними зверху даними. Це буде виглядати наступним чином:

```
export class RegistrationDto{
     @IsEmail()
     @IsNotEmpty()
     public email: string

     @IsNotEmpty()
     @IsString()
     @MinLength(6)
     @MaxLength(64)
     public password: string

     @IsOptional()
     @IsString()
     @MinLength(3)
     @MaxLength(64)
     public username: string
}
```

Як можна побачити, class-validator також має можливість перевіряти дані які надходять на різні кількісні та якісні параметри, такі як чи є параметр строкою, чи його довжина більша за 3 та інші. Також можно помічати дані які є опціональними, чи навпаки не мають бути пустими. Nest.js інтегрував цю бібліотеку у свій фреймворк, тому помилки валідації коректно працюють без детального налаштування, але по бажанню є можливість кастомізувати поведінку ПЗ при фіксації некоректних вхідних даних. Наприклад можна налагодити поведінку сервера при отриманні даних не зазначених в цьому класі.

У назві класу зазначена абревіатура DTO (Data Transfer Object) - один із шаблонів проектування, який використовують для передачі даних між підсистемами програми. Саме такі об’єкти перевіряються бібліотекою найчастіше.

РОЗДІЛ 4. End-to-End тестування

Тестування - така ж важлива частина розробки ПЗ як і написання функціональностей. Без тестування неможливо знайти помилки які не побачив програміст під час своєї роботи. З зростанням складності проекту, важко гарантувати що стара функціональність працює так само як думалось спочатку, а нова функціональність яка базується на функціональності попередніх модулів ПЗ також може бути некоректною з логічної точки зору, бо базується на старих помилках. Саме тому підхід програмування TDD (Test Driven Development) зараз існує та практикується у сучасних проектах, де кожна функціональність не починає розроблятися поки для неї не будуть написані тести. Однак у випадку нашого ПЗ не був використаний цей підхід.

Основні типи тестування ПЗ:

- Unit тестування
- End-to-End тестування

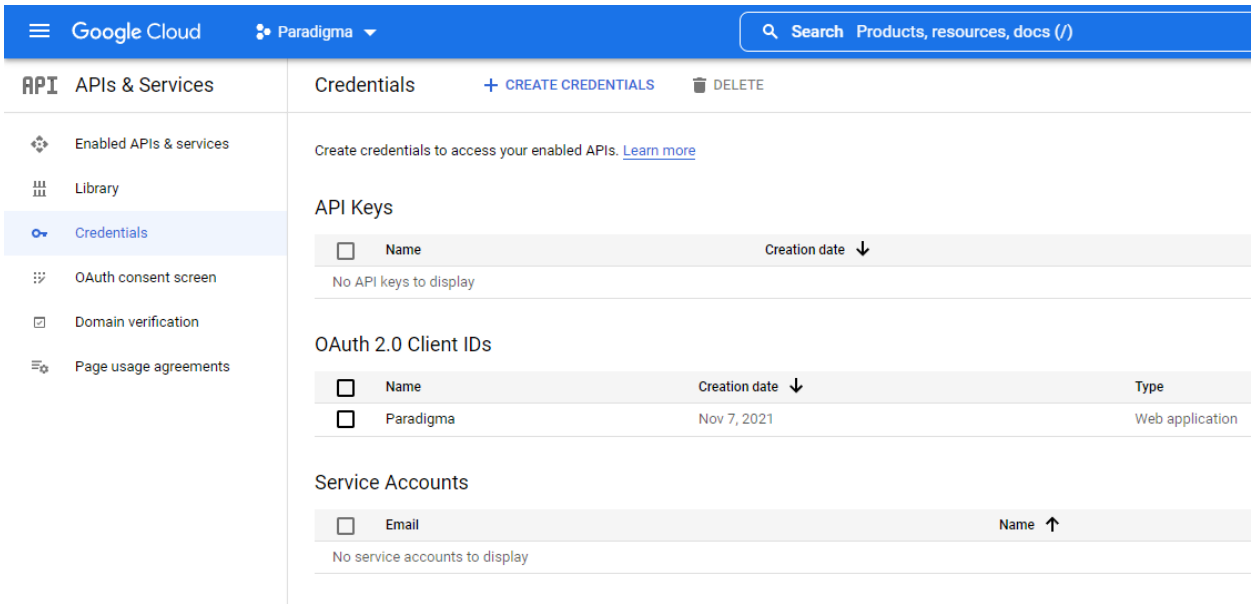
Різниця між цими тестами полягає у контексті у якому тестується функціональність. У Unit тестуванні тестується кожна окрема функція модулю через фейкування вхідних параметрів. Під час цього типу тестування можна поглинутись у процес праці кожної відокремленої функції продукту. Але протестувати кожну функцію може бути дуже непростми з точки зору часу завданням, і не завжди воно є змістовним та потрібним.

End-to-End тестування емулює працю окремих модулів проекту в цілому, емулюючи конкретний контекст системи та запит з клієнтської частини до серверу. Тому перевіряється праця системи в цілому як і її реакція на несподівані, виняткові ситуації, які може створити розробник під час розробки тестів. У проекті Paradigma кожен модуль покритий End-to-End тестуванням.

РОЗДІЛ 5. Google аутентифікація

Сучасні великі сервіси створюють навколо себе мережу пов'язаних сервісів для роботи з якими потрібен лише один аккаунт. Google має велику кількість таких сервісів, і використовує один обліковий запис для роботи з ними. Ця велика корпорація може собі дозволити поширювати свої облікові записи за даними користувачів для інших сервісів. API Google автентифікації відкритий для всіх хто має намір використовувати обліковий запис Google як один з способів аутентифікації користувачів на третіх платформах. Для роботи потрібно лише мати підтверджений Google аккаунт, та перейти у консоль розробника Google, де створити свій додаток. Google згенерує ключі доступу завдяки яким буде проводитися автентифікація через сервера Google. Авжеж це сторонній сервіс який має ліміт по використанню у безкоштовному використанні, але його достатньо для використання у малому проекті.

Для підтримки декількох способів аутентифікації у проекті Paradigma було використано пакет - passport.js та його доробку для архітектури Nest.js - @nest/passport. Цей пакет абстрагує аутентифікаційні методи, чим дозволяє використовувати один інтерфейс для великої кількості різних видів аутентифікації.



Для коректної роботи Google авторизації потрібно налаштувати passport-стратегію Google авторизації на бек-енді, а в Google Cloud Console вказати перенапрямок для вдалої та невдалої авторизації. Далі під час вдалої

авторизації - Google надішле публічні дані користувача, які Paradigma може використовувати у своїх цілях.

РОЗДІЛ 6. Пагінація

Інколи даних буває забагато для одночасного відображення. Припустимо що потрібно отримати публікації користувача. Але користувач має більше тисячі записів в таблиці свої постів в базі даних. Такий великий струм даних може викликати незручність в користуванні, або в загалом порушення працездатності клієнтської частини проекту. Тому потрібно дозувати дані які надходять до користувача по його запиту, така стратегія називається пагінацією.

Найпростіша пагінація використовує курсори у запитах на сервер для отримання інформації щодо позиції звідки починати вибирати записи для відправки на клієнт. Таким курсором може бути унікальний ідентифікатор останнього відправленого запису по особливому сортуванню, або більш складний ідентифікатор якщо пагінація залежить від багатьох станів у системі.

Аналіз отриманих результатів, висновки та пропозиції

Виконана кваліфікаційна робота має практичну орієнтацію на розробку веб-доданка, функціонал якого збігається з існуючим сервісом розміщення фотографій “Instagram”, однак має набір деяких переваг, серед яких вважаю можливим виокремити механізм оптимізованої обробки файлів зображень, що зменшує обсяг пакетів даних які передаються по мережі, який потребував розробки стороннього сервісу, однак має суттєвий вплив на весь функционал розробленого програмного забезпечення.

Проведено огляд систем аутентифікації, показано, що найбільш поширеними є системи з введенням авторизаційних даних платформи чи використовування третіх API облікових записів для швидкого старту та токен-орієнтовані системи. Було використано комбіновану систему яка дозволила абстрагуватися від типу авторизації на рівні системи, та порівняти облікові записи користувачів на одному рівні незважаючи на різність підходів входу у систему. У якості третього API для отримання користувацьких даних була використана платформа Google Authentication.

Проведено аналіз систем обробки зображень та алгоритмів стиснення файлів для досягнення найбільш оптимізованого по часу обробки та якістю вихідних зображень. Було розроблено сторонній сервіс який щільно пов’язаний з основною системою.

Під час розробки серверної частини проекту велику увагу було приділено до тестування функціоналу. Кожний модуль системи має відокремлений тестувальний End-to-End модуль що забезпечує комфорт та впевненість під час розробки нових функціональностей чи зміни існуючої. Тестування також підвищує загальну стабільність проекту, бо дозволяє знаходити неявні проблеми та вразливості набагато раніше, та вирішувати їх.

Проведено огляд систем та практик розробки систем з захистом користувацьких даних. Були використані більшість практик щодо реалізації безпечного користування системою. Шифрування паролів за допомогою алгоритму bcrypt та використання JWT (Json Web Token) для транспортування зашифрованих аутентифікаційних даних між клієнтом і сервером, що забезпечує отримання контексту с даними користувача можливим лише на сервері який має потрібні ключі для розшифрування токену.

У проекту є великий потенціал на розростання функціоналу. Одними з найважливіших та найпріоритетніших є:

- метрики користувацького досвіду з контентом;
- використання метрик для покращення алгоритму генерування персоналізованої стрічки контенту;
- використання штучного інтелекту для генерування метадати зображень, яка має бути використана для покращення роботи алгоритму генерування персоналізованої стрічки контенту;
- блогер-профіль користувача, який має в собі метрикову інформацію щодо взаємодії інших користувачів з його контентом;
- підтримка багатомовності;

Використані ресурси

1. Неженцев М.М. Разработка web-сайта как переходный этап в преподавании между графическим дизайном и программированием https://cyberleninka.ru/article/n/razrabotka-web-sayta-kak-perehodnyy-etap-v-prepodavanii-mezhdu-graficheskim-dizaynom-i-programmirovaniem
2. Круг, С. Веб-дизайн: книга Стива Круга или «не заставляйте меня думать!» / С. Круг. – 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2008. – 224 с.
3. Кантелон Майк, Хартер Марк, Головайчук TJ, Райлих Натан. Node.js в действии 2-е издание. - Пер. с англ. - СПб
4. Саймон Холмс. Стек MEAN. Изд. Manning. - Пер. с англ. - СПб
5. Сэмми Пьюривал. Основы разработки Веб-приложений. Изд. Manning. - Пер. с англ. - СПб
6. Марио Каскиаро, Лучано Маммино. Шаблоны проектирования Node.js. Изд. PACKT. Пер. с англ. ДМК.
7. Хэррон Дэвид. Node.js Разработка серверных веб-приложений на JavaScript. Пер. с англ. ДМК.
8. Документація Nest.js - https://docs.nestjs.com/
9. Інтернет-ресурс Wanago https://wanago.io/
10. Шелли Пауэрс. Изучаем Node. Переходим на сторону сервера. Пер. с англ. Питер
11. Кирилл Сухов. Node.js. Путеводитель по технологии. Изд. ДМК
12. Рогов Е. В. PostgreSQL изнутри. — М.: ДМК Пресс, 2022. — 660 с.
13. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.: ил.
14. Новиков Б. А. Основы технологий баз данных: учеб. пособие / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. — 2-е изд. — М.: ДМК Пресс, 2020. — 582 с.
15. Даніїл Корреа, Крег Лім. Practical Nest.js: Develop clean MVC web applications.
16. Грег Маголан, Патрік Хауслі, Адрієн де Перетті, Джей Белл, Девід Гуярро. Nest.js: A Progressive Node.js Framework. Изд. Packt