МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА REST API СЕРВІСУ ОНЛАЙН ОГОЛОШЕНЬ ЗАСОБАМИ GO»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.І. Євса
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Євса Даниїлу Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка REST API сервісу онлайн оголошень засобами Go

керівник роботи Лісняк Андрій Олександрович,к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Практична реалізація коду.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація та тестування веб-сайту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	16.02.2023	
3.	Обробка методичних та теоретичних	03.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	17.04.2023	
5.	Розробка третього розділу.	15.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2023	

Студент _____ Д.І. Євса
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка REST API сервісу онлайн оголошень засобами Go»: 43 с., 28 рис., 19 табл., 13 джерел.

БД, ІНФОРМАЦІЙНА СИСТЕМА, API, ECHO, GOLANG, POSTGRESQL, UML.

Об’єкт дослідження – мова програмування Go.

Мета роботи: вивчити мову програмування Go, розробити JSON API для сайту оголошень.

Методи дослідження – аналіз, вивчення та узагальнення, моделювання.

У кваліфікаційній роботі бакалавра досліджується предметна область, а також розглядаються вимоги до системи та засоби реалізації поставленого завдання. При розробці системи було проведено аналіз предметної області, сформовано технічне завдання, було обрано фреймворк та архітектура розробки, розроблена функціональна, концептуальна та логічна модель системи. Також була оформлена супровідна документація, документування та тестування API.

SUMMARY

Bachelor’s Qualifying Paper «Development of an Online Advertisement Service REST API using Go»: 43 pages, 28 figures, 19 tables, 13 references.

DB, INFORMATION SYSTEM, API, ECHO, GOLANG, POSTGRESQL, UML.

The object of the study is Go programming language.

The aim of the study is learning the Go programming language, develop a JSON API for an ad site.

The methods of research are analysis, study and generalization, modeling.

The bachelor's qualification work examines the subject area, as well as considers the system requirements and means of implementing the task. During the development of the system, an analysis of the subject area was carried out, a technical task was formed, a framework and a development architecture were selected, and a functional, conceptual and logical model of the system was developed. The accompanying documentation, API documentation and testing were also designed.

ЗМІСТ

ВСТУП

Розробка інтернет-маркетплейсів є великою потребою, оскільки все більше людей шукають зручний спосіб придбання будь-яких товарів в Інтернеті. Мова програмування Go вважається мовою загального призначення, але основне застосування – розробка вебсервісів та клієнт-серверних програм. Go має багато корисних функцій та можливостей для створення високоякісних сервісів.

Отже, актуальною задачею є створення функціонального та зручного для користувача інтернет сайту для розміщення будь-яких оголошень, який має корисні функції, такі як створення та пошук оголошень, реєстрацію акаунту, панель адміністратора та залишення коментарів до оголошень.

Метою курсової роботи є розробка інтернет сайту для розміщення оголошень засобами мови програмування Go.

Задачі, які необхідно розв’язати для досягнення поставленої мети:

– проаналізувати існуючі фреймворки Golang;
– проаналізувати предметну область;
– спроєктувати схему даних;
– реалізувати базу даних;
– реалізувати інтерфейс;
– протестувати роботу сайту.

Об’єкт дослідження – процес розробки інтернет-сайту оголошень.

Предмет дослідження – мова програмування Go.

Методи дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії, системний аналіз.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Характеристика мови програмування Go

Go представляє компільовану статично типізовану мову програмування від компанії Google. Мова Go призначена для створення різноманітних додатків, але насамперед це веб-сервіси та клієнт-серверні програми. Хоча також мова має можливості для роботи з графікою, низькорівневими можливостями тощо. Робота над мовою Go розпочалася у 2007 у надрах компанії Google. Одним із авторів є Кен Томпсон, який, до речі, є й одним із авторів мови Сі (поряд із Денисом Рітчі). 10 листопада 2009 року мова була анонсована, а в березні 2012 року вийшла версія 1.0. При цьому мова продовжує розвиватись. Поточною версією на момент написання є версія 1.20, яка вийшла у лютому 2023 року [1 – 4].

Мова Go розвивається як open source, тобто представляє проєкт з відкритим вихідним кодом, і всі його коди та компілятор можна знайти та використовувати безкоштовно.

Go є кросплатформним, він дозволяє створювати програми під різні операційні системи – Windows, Mac OS, Linux, FreeBSD. Код має переносимість: програми, написані для однієї з цих операційних систем, можуть легко з перекомпіляцією перенесені на іншу ОС.

Основні особливості мови Go:

– компільований – компілятор транслює програму на Go у машинний код, зрозумілий для певної платформи;

– статично типізований;

– присутній збирач сміття, який автоматично очищує пам'ять;

– підтримка роботи з мережними протоколами;

– підтримка багатопоточності та паралельного програмування.

В даний час Go знаходить широке застосування у різних сферах.

Зокрема, серед відомих проєктів, які застосовують Go, можна знайти такі: Google, Dropbox, Netflix, Kubernetes, Docker, Twitch, Uber, CloudFlare та інші.

1.2 Аналіз популярних фреймворків Go для розробки вебдодатків

Веб-фреймворки Golang використовуються для безпосереднього написання інтерфейсів прикладного програмування (API) і веб-служб. Під час створення невеликих додатків фреймворки не є необхідними, але вони потрібні в програмному забезпеченні на виробничому рівні. Навіть знаючи функції та володіючи знаннями, кодування програми налагодження робочого рівня займає величезну кількість часу. З цієї причини часто використовуються фреймворки. Фреймворки надають додаткові функції та служби, якими можуть користуватися інші розробники, які хочуть додати подібні функції до свого програмного забезпечення, а не самостійно писати повноцінне програмне забезпечення.

Розглянемо наступні найпопулярніші фреймворки.

Gin [5]. Фреймворк Gin очолює список фреймворків Go з точки зору популярності завдяки мінімалістичному фреймворку та продуктивності. Здебільшого він використовується для створення REST API для серверної частини, якщо програміст хоче розробити односторінкову програму за допомогою інтерфейсу. Цей фреймворк використовує HTTP-маршрутизатори для обробки трафіку Golang і ідеально підходить для початківців разом із багатою документацією, наданою на GitHub. Він підтримує найважливіші бібліотеки та функції, що робить його найкращим фреймворком Golang для розробки високопродуктивних REST API.

Найбільший недолік полягає в тому, що він підходить для невеликих додатків, але не підходить для розробки великих серверних додатків або складних функцій серверів корпоративного рівня.

Beego [6]. Фреймворк Beego використовується для швидкої розробки

REST API, веб-додатків і серверних служб у Golang. Його часто розглядають подібно до веб-фреймворку Django в Python і містять особливі функції Golang, такі як інтерфейси та вбудовування структур. Він не вимагає встановлення сторонніх розробників. Це повноцінний фреймворк Model-View-Controller (MVC) із власними бібліотеками та вбудованим інструментом, який допомагає знаходити зміни коду, відомим як Bee Tool.

Крім того, він розділений на вісім модулів, які можна ігнорувати або використовувати за потреби. Він інтегрує карту об'єктних зв'язків (ORM), яка допомагає організовувати базу даних програми разом із інструментами обробки сеансів і системами журналювання. Не тільки це, але він також містить обробник кешу та бібліотеки для роботи з елементами HTTP. Ще одна чудова функція полягає в тому, що вона добре працює з інструментами командного рядка, подібно до того, як Django використовує командний рядок.

Єдиний недолік beego полягає в тому, що через його високу функціональність і широкі можливості він не дуже підходить для новачків.

Echo. Echo-фреймворк, який використовується в Go, є ще одним високопродуктивним, розширюваним і мінімалістичним веб-фреймворком у Golang. Він має високооптимізований HTTP-маршрутизатор із нульовим динамічним розподілом пам'яті, який розумно визначає пріоритети маршрутів. Він використовується для створення надійних і масштабованих REST API, які можна легко організувати в групи. Він автоматично встановлює сертифікати TLS від Let's Encrypt і забезпечує підтримку HTTP/2, що підвищує швидкість і забезпечує кращий досвід роботи з користувачем. Він також містить багато вбудованих проміжних програм, які можна використовувати, і розробники навіть можуть визначити власні, які можна встановити на рівні кореня, групи або маршруту. Він підтримує зв'язування даних для запитів HTTP, включаючи JSON, XML або дані форми. Для відтворення даних він містить API для надсилання різних HTTP-відповідей, зокрема JSON, XML, HTML, файлів і вкладень. Шаблони можна

відобразити за допомогою будь-якого механізму шаблонів і мати налаштовану центральну обробку помилок HTTP.

Мінус використання echo framework полягає в тому, що він підтримується лише одним розробником, а код оновлюється рідко.

Kit. Фреймворк Kit — це набір інструментів програмування для створення надійних мікросервісів у Golang, які можна підтримувати. Це набір пакетів і передових практик, які забезпечують комплексний, надійний і надійний спосіб створення мікросервісів для організацій будь-якого розміру. Go — чудова мова загального призначення, але мікросервіси потребують певної спеціалізованої підтримки.

Таким чином, структура комплекту забезпечує безпеку Remote Procedure Call (RPC), можливість спостереження за системою та інтеграцію в інфраструктуру. Він складається з кількох взаємопов'язаних пакетів, які разом утворюють впевнену структуру для побудови великих сервіс-орієнтованих архітектур (SOA) і роблять Golang першокласною мовою для написання мікросервісів у будь-якій організації. Він був розроблений для сумісності, і розробники можуть вільно вибирати бази даних, компоненти, платформи та архітектуру, яка їм найкраще підходить.

Недоліком використання go-kit є те, що через інтенсивне використання інтерфейсів накладні витрати на додавання API до служби дуже високі.

Fasthttp. Фреймворк fasthttp забезпечує швидкий HTTP-сервер і клієнтський API, який було створено як альтернативу net/http через обмеження можливостей оптимізації. Він оптимізований для підвищення швидкості та може легко обробляти понад 100 тис. qps і більше 1 млн одночасних підключень Keep-Alive на сучасному обладнанні. Він також оптимізований для низького використання пам'яті та забезпечує легку підтримку оновлення підключення за допомогою RequestCtx.Hijack.

Fasthttp API розроблено з можливістю розширювати існуючі клієнтські та серверні реалізації або писати власні клієнтські та серверні реалізації з нуля. Обробнику запитів надається багато додаткової корисної інформації, як-от адреси сервера та клієнта для кожного запиту.

1.3 Технічне завдання

Функціональне призначення системи: Інформаційна система «GoAds» призначена для користувачів, які мають потребу щось продати, або купити.

Експлуатаційне призначення системи: Система повинна експлуатуватися будь-яким бажаючим щось продати або купити.

Мета створення системи: Система прискорює процес пошуку товару, або продаж товару. Не обов’язково їхати в магазин для того, щоб знайти потрібну річ, і також може зв’язатися з продавцем напряму.

Цей сайт має виконувати наступні функції:

– адміністрування сайту;
– реєстрація акаунту;
– відновлення паролю;
– перегляд оголошень;
– створення оголошень;
– фільтр та пошук оголошень;
– перегляд коментарів до оголошень;
– загальний чат.

1.4 Вимоги до системи

Робота із сайтом відбувається наступним чином. Спочатку потрібно зареєструватися, далі можна як створити оголошення, так і переглянути існуючі. У оголошення можна вказати назву, опис, додати ціну та фото. У існуючих оголошеннях можна переглядати опис, фото, додати коментарі.

На сайті для адміністратора є доступ до редагування оголошень та коментарів, якщо були порушені правила розміщення оголошень.

1.4.1 Вимоги до структури системи

Сайт для клієнтів повинен складатися з наступних сторінок:

– головна сторінка, на якій відображається 10 випадкових оголошень з фото;

– реєстрація або авторизація;

– пошук оголошень з фільтром;

– створення оголошення;

– редагування оголошення;

– додавання коментарів до оголошень.

Пункти, з яких складається оголошення:

– назва;

– опис;

– ціна;

– фото.

Панель для адміністратора повинна складатися з наступного:

– редагування оголошень;

– редагування ролей користувачів.

1.5 Огляд використовуваних технологій

За допомогою онлайн ресурсу DrawIO було виконано діаграми діяльності сайту, для якого проєктується інформаційна система. СКБД використовувалася PostgreSQL [7, 11]. Для розробки використовувалася IDE GoLand.

У даній роботі була розроблена серверна частина сайту. Налагоджено її за допомогою фреймворку echo. Також були використані такі сервіси: Amazon 3S, Golang-migrate, Mailgun, Postman, та інші.

1.6 Висновки до розділу 1

У цьому розділі було розглянуто мову програмування Golang та її найпопулярніші фреймворки Gin, Beego, Echo, Kit та Fasthttp. Було сформовано технічне завдання, вимоги до системи та було оглянуто технології, які будуть використовуватися в майбутньому.

2 ПРОЄКТУВАННЯ САЙТУ

2.1 Концептуальне проєктування

Для використання веб-додатку «GoAds» потрібно мати веб-браузер. Після входу на сайт користувач має змогу зареєструватися, створювати нові оголошення, змінювати, видаляти їх, а також залишати коментарі під іншими оголошеннями. Також є функція чату. Діаграма прецедентів для веб-сайту складається з одного актора і чотирьох прецедентів (рис. 2.1). На даній діаграмі актором є користувач сайту, а прецедентами дії, які він має змогу виконувати з ПЗ.

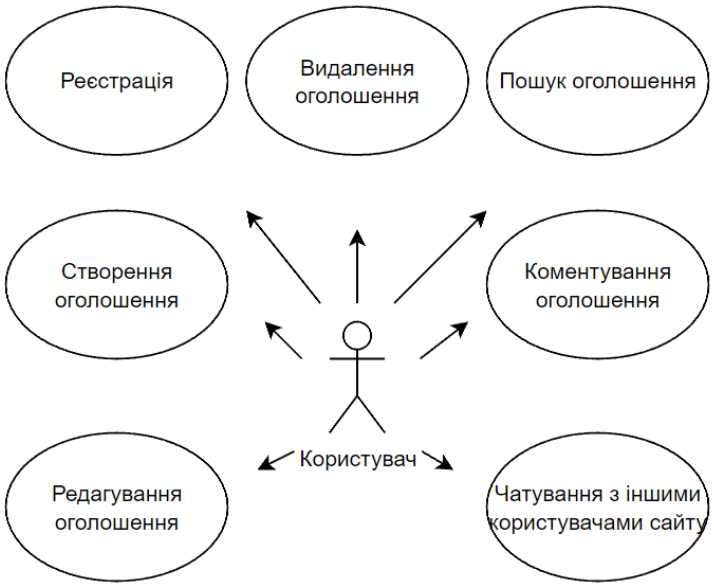

Рисунок 2.1 – Діаграма прецедентів

2.2 Проєктування структури зберігання даних

Для зберігання даних додатку було вирішено використовувати СКБД PostgreSQL, яка реалізує невеликий, швидкий, автономний, високонадійний,

повнофункціональний механізм баз даних SQL. PostgreSQL має багато функцій, які допомагають розробникам створювати додатки, адміністраторам захищати цілісність даних і створювати відмовостійке середовище, а також допомагають вам керувати своїми даними, незалежно від того, наскільки великий чи малий набір даних. Крім того, що PostgreSQL є безкоштовним і відкритим вихідним кодом, він дуже розширюваний. Наприклад, можна визначати власні типи даних, створювати власні функції, навіть писати код з різних мов програмування без перекомпіляції бази даних. Отже, для додатку «GoAds» було спроєктовано такі сутності: users, roles, user_to_roles, passwords_recovery, advertisements, gallery, sessions, comments (рис. 2.2).

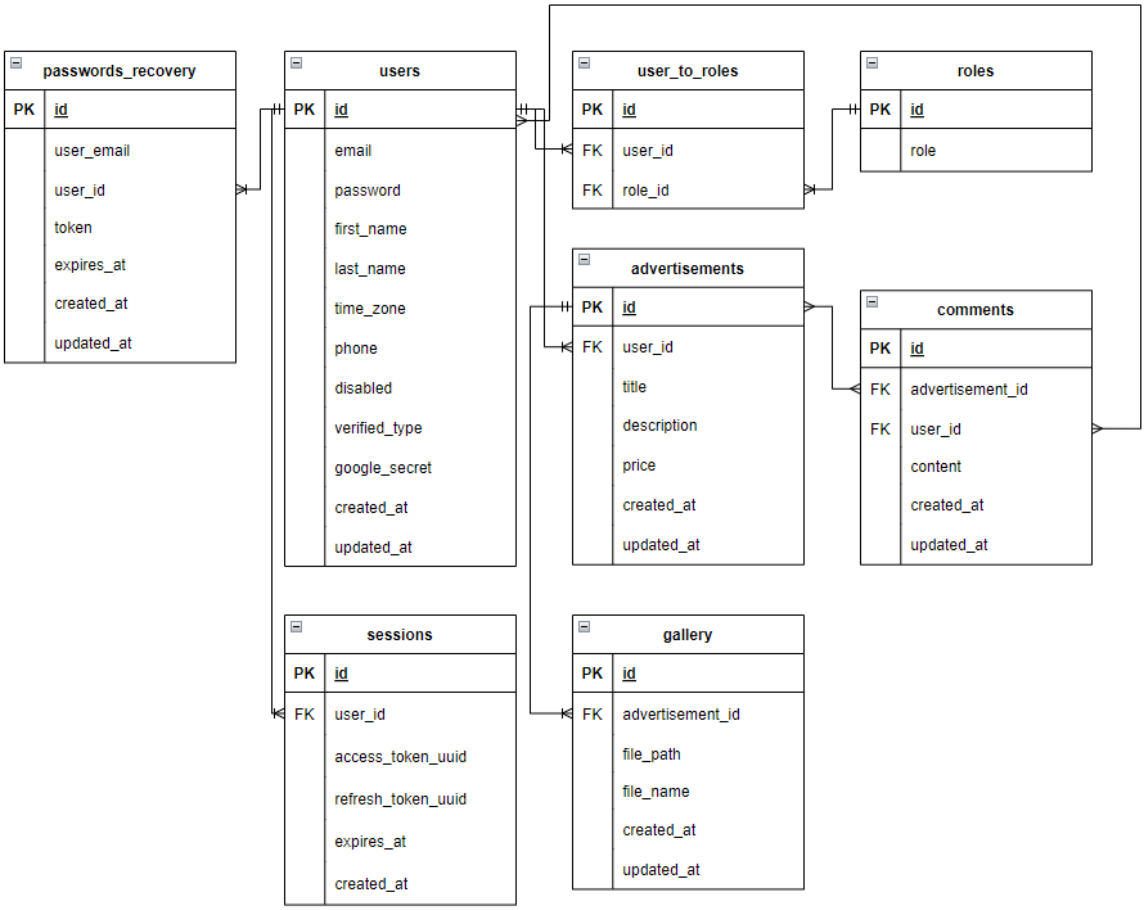

Рисунок 2.2 – ER діаграма

Сутність User зберігає всю інформацію про користувача – це ім’я, прізвище, електрона адреса, номер телефону, пароль, часовий пояс. Сутність

Roles містить назву ролей, та їх ідентифікаційний номер. Сутність User to roles міст інформацію про ролі користувачей. Сутність Sessions містить інформацію про сессії користувачів, вони потрібні задля роботи access токенів. Сутність Password recovery потрібна для функції відновлення паролю. Сутність Advertisements містить інформацію про оголошення – це назва, опис, ціна та власника оголошеня. Сутність Gallery містить фотографії до оголошень. Сутність Comments містить коментарі до оголошень.

2.3 Логічне проєктування інформаційної системи

Інфологічна (концептуальна) модель повинна включати такий формалізований опис предметної області, яке легко буде "читатися" не тільки фахівцями по базах даних, а й іншими працівниками.

Інфологічне проєктування, перш за все, пов'язано зі спробою уявлення семантики предметної області в моделі БД. Реляційна модель даних в силу своєї простоти і лаконічності не дозволяє відобразити семантику, тобто зміст предметної області.

Проблему подання семантики давно цікавили розробників, і в сімдесятих роках було запропоновано кілька моделей даних, названих семантичними моделями. До них можна віднести семантичну модель даних, запропоновану Хаммером і Мак-Леоном в 1981 році, функціональну модель даних Шипман, також створену в 1981 році, модель "сутність-зв'язок", запропоновану Ченом (Chen) в 1976 році, і ряд інших моделей. У всіх моделей були свої позитивні і негативні сторони, але випробування часом витримала лише остання. І зараз саме модель Чена "сутність-зв'язок", стала фактичним стандартом при инфологической моделюванні баз даних.

Сутність – це об’єкт будь-якої природи (реальний або уявний), що може бути ідентифікований певним способом, який вирізняє його від інших об’єктів. Сутності містять різні атрибути. Атрибут – це поіменована логічно

неподільна властивість (характеристика) сутності.

У наступних таблицях (2.1 – 2.8) будуть розглянуті сутності, атрибути, типи даних та ключі.

Таблиця «User» зберігає інформацію про користувачів. Опис таблиці представлено нижче.

Таблиця 2.1 – Таблиця для сутності Users

Атрибут	Тип даних	Ключ
id	Текстовий	Первинний
email	Текстовий	
password	Текстовий	
first_name	Текстовий	
last_name	Текстовий	
time_zone	Текстовий	
phone	Текстовий	
disable	Логічний	
verified_type	Текстовий	
google_secret	Текстовий	
created_at	Дата/час	
updated_at	Дата/час	

Таблиця «Roles» зберігає інформацію про ролі. Опис таблиці представлено нижче.

Таблиця 2.2 – Таблиця для сутності Roles

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
role	Текстовий	

Таблиця «User to roles» зберігає інформацію про користувачів та їх ролі. Опис таблиці представлено нижче.

Таблиця 2.3 – Таблиця для сутності User to roles

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
user_id	Текстовий	Зовнішній
role_id	Числовий	Зовнішній

Таблиця «Passwords recovery» зберігає інформацію про токени для відновлення паролів. Опис таблиці представлено нижче.

Таблиця 2.4 – Таблиця для сутності Passwords recovery

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
user_email	Текстовий	
user_id	Текстовий	
token	Текстовий	
expires_at	Дата/час	
created_at	Дата/час	
updated_at	Дата/час	

Таблиця «Sessions» зберігає інформацію про сессії користувачів. Опис таблиці представлено нижче.

Таблиця 2.5 – Таблиця для сутності Sessions

Атрибут	Тип даних	Ключ
id	Текстовий	Первинний
user_id	Текстовий	Зовнішній

Продовження табл. 2.5

Атрибут	Тип даних	Ключ
access_token_uuid	Текстовий	
refresh_token_uuid	Текстовий	
expires_at	Дата/час	
created_at	Дата/час	

Таблиця «Advertisements» зберігає інформацію про оголошення користувачів. Опис таблиці представлено нижче.

Таблиця 2.6 – Таблиця для сутності Advertisements

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
user_id	Текстовий	Зовнішній
title	Текстовий	
description	Текстовий	
price	Текстовий	
created_at	Дата/час	
updated_at	Дата/час	

Таблиця «Gallery» зберігає інформацію про фотографії до оголошень. Опис таблиці представлено нижче.

Таблиця 2.7 – Таблиця для сутності Gallery

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
advertisement_id	Числовий	Зовнішній
file_path	Текстовий	
file_name	Текстовий	

Продовження табл. 2.7

Атрибут	Тип даних	Ключ
created_at	Дата/час	
updated_at	Дата/час	

Таблиця «Comments» зберігає інформацію про коментарі до оголошень. Опис таблиці представлено нижче.

Таблиця 2.8 – Таблиця для сутності Comments

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
advertisement_id	Числовий	Зовнішній
user_id	Текстовий	Зовнішній
content	Текстовий	
created_at	Дата/час	
updated_at	Дата/час	

2.4 Висновки до розділу 2

У цьому розділі було проведено концептуальне проєктування, а саме – була створеня діаграма прецедентів, також було проведено проєктування системи зберігання даних у якій були спроєктовані сутності бази даних та ER-діаграма.

3 РЕАЛІЗАЦІЯ

3.1 Первинне налаштування проєкту

Кожна програма Go складається з пакетів. Кожен вихідний файл Go починається з оголошення пакета, який містить лише ім'я пакета. Тож для початку потрібно створити головний файл main.go, у ньому оголосити пакет та функцію main (рис. 3.1).

```
package main
import "fmt"
func main()  {
        fmt.Println("Hello world!")
}
```

Рисунок 3.1 – Найпростіша програма на мові програмування Go

Далі потрібно створити config файл (рис. 3.2), котрий буде містить інформацію про підключення до бази даних, порт серверу та токен доступу на основі JSON (JWT).

```
database:
  dialect: postgres
  host: localhost
  port: 5432
  user: postgres
  name: ads
  password: 111111
server:
  address: 8080
jwt:
  key: example_key
  refreshKey: example_refresh_key
  accessTokenLifespan: 30 # in minutes
  refreshTokenLifespan: 2880 # in minutes
s3:
  bucketName: advertisements.bucket
mailgun:
  domainName: sandbox12c920d206d44166a53355f85b9eac3c.mailgun.org
  APIKey: 0e9760f2683ddb40d5594f22d2615cef-2de3d545-faccf518
```

Рисунок 3.2 – Файл з налаштуваннями проєкту config.yml

Далі потрібно створити функцію, яка буде зчитувати інформацію з config файлу (рис. 3.3).

```
package config
import (
	"GoAds/domain"
	"context"
	"fmt"
	cfg "github.com/aws/aws-sdk-go-v2/config"
	"github.com/aws/aws-sdk-go-v2/service/s3"
	"github.com/mailgun/mailgun-go/v4"
	"github.com/spf13/viper"
	"log"
	"os"
	"time"
)
type config struct {
	Database struct {
		Dialect  string
		Host     string
		Port     string
		User     string
		Name     string
		Password string
	}
	Server struct {
		Address string
	}
	JWT struct {
		Key                string
		RefreshKey         string
		AccessTokenLifespan  time.Duration
		RefreshTokenLifespan time.Duration
	}
	S3 struct {
		BucketName string
	}
	Mailgun struct {
		DomainName string
		ApiKey     string
	}
}

var C config
var BucketClient *s3.Client
var MG *mailgun.MailgunImpl
func createBucket() (*s3.Client, error) {
	bucketConfig, err := cfg.LoadDefaultConfig(context.TODO())
	if err != nil {
		log.Fatalf("failed to load configuration, %v", err)
		return nil, err
	}
	client := s3.NewFromConfig(bucketConfig)
	return client, nil
}
func ReadConfig(configPath, configName, configType string) {
	Config := &C
	viper.AddConfigPath(configPath)
	viper.SetConfigName(configName)
	viper.SetConfigType(configType)
```

```
		if err := viper.ReadInConfig(); err != nil {
			fmt.Println(err)
			log.Fatalln(err)
		}
		if err := viper.Unmarshal(&Config); err != nil {
			fmt.Println(err)
			os.Exit(1)
		}
		if C.JWT.Key == "" || C.JWT.RefreshKey == ""{
			log.Println(domain.ErrEmptyJWTKey)
			os.Exit(1)
		}
		var err error
		BucketClient, err = createBucket()
		if err != nil {
			log.Println(err)
			os.Exit(1)
		}
		MG = mailgun.NewMailgun(C.Mailgun.DomainName, C.Mailgun.ApiKey)
}
```

Рисунок 3.3 – Функція для зчитування інформації з файлу config.yml

Наступним кроком буде створення головного файлу, з котрого починається вся робота проєкту (рис. 3.4). В ньому ініціалізується робота логгеру, створюється підключення до бази даних, ініціалізуються шляхи до сторінок (routes) та запускається сервер.

```
package main
import (
	"GoAds/config"
	"GoAds/infrastructure/datastore"
	"GoAds/infrastructure/routes"
	"GoAds/registry"
	"fmt"
	"github.com/labstack/echo"
	logrus "github.com/sirupsen/logrus"
	"log"
	"os"
	"time"
)
func init() {
	logrus.SetOutput(os.Stdout)
	logrus.SetFormatter(&logrus.JSONFormatter{})

	logLevel, err := logrus.ParseLevel(os.Getenv("LOG_LEVEL"))
	if err != nil {
		logLevel = logrus.InfoLevel
	}
	logrus.SetLevel(logLevel)
}
func main() {
	config.ReadConfig("./config", "config", "yml")
	db := datastore.NewDB()
	db.LogMode(true)
	defer db.Close()
```

```
		r := registry.NewRegistry(db)
		e := echo.New()
		e.Use(loggingMiddleware)
		e = routes.NewRouter(e, r.NewAppController())
		fmt.Println("Server listen at http://localhost" + ":" + config.C.Server.Address)
		if err := e.Start(":" + config.C.Server.Address); err != nil {
			log.Fatalln(err)
		}
	}
	func loggingMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
		return func(c echo.Context) error {
			start := time.Now()
			res := next(c)
			logrus.WithFields(logrus.Fields{
				"method":     c.Request().Method,
				"path":       c.Path(),
				"status":     c.Response().Status,
				"latency_ns": time.Since(start).Nanoseconds(),
				"user_ip":    c.Request().RemoteAddr,
			}).Info("request details")
			return res
		}
	}
```

Рисунок 3.4 – Головний файл main.go

Далі потрібно створити підключення до бази даних (рис. 3.5).

```
	package datastore
	import (
		"GoAds/config"
		"fmt"
		"github.com/jinzhu/gorm"
		_ "github.com/jinzhu/gorm/dialects/postgres"
		"log"
	)
	func NewDB() *gorm.DB {
		dbURI := fmt.Sprintf("host=%s user=%s dbname=%s sslmode=disable password=%s port=%s", config.C.Database.Host, config.C.Database.User, config.C.Database.Name, config.C.Database.Password, config.C.Database.Port)
		db, err := gorm.Open(config.C.Database.Dialect, dbURI)
		if err != nil {
			log.Fatalln(err)
		} else {
			fmt.Println("Successfully connected to PostgreSQL!")
		}
		return db
	}
```

Рисунок 3.5 – Підключення до бази даних, файл db.go

Наступним кроком буде створення шляхів до сторінок сайту, буде розглянуто приклад шляху до пошуку, створення, редагування та видалення оголошень (рис. 3.6). Також у проєкті реалізована система ролей. При

реєстрації користувачу за замовчуванням видається роль «User» та роль «Advertisement», котра надає доступ до шляхів оголошень. Тож якщо користувач не має ролі «Advertisement», йому буде відмовлено в доступі до сторінки.

```
package routes
import (
	"GoAds/domain"
	"GoAds/domain/model"
	"GoAds/interface/controller"
	"github.com/labstack/echo"
	"github.com/labstack/echo/middleware"
	"log"
	"strings"
)
func Auth(next echo.HandlerFunc) echo.HandlerFunc {
	return func(c echo.Context) error {
		authHeader := c.Request().Header.Get("Authorization")
		headerParts := strings.Split(authHeader, " ")
		if len(headerParts) != 2 {
			return domain.ErrInvalidAccessToken
		}
		if headerParts[0] != "Bearer" {
			return domain.ErrInvalidAccessToken
		}
		claims, err := controller.ParseToken(headerParts[1])
		userID, userRoles := claims.UserID, claims.UserRoles

		if err != nil {
			log.Println(err)
			return domain.ErrInvalidAccessToken
		}
		c.Set(domain.SessionDataKey, domain.SessionData{
			UserID:    userID,
			UserRoles: userRoles,
		})
		return next(c)
	}
}
func Role(roleID ...int) echo.MiddlewareFunc {
	return func(next echo.HandlerFunc) echo.HandlerFunc {
		return func(c echo.Context) error {
			userRoles := getUserRolesID(c)
			for _, v := range roleID {
				for _, v2 := range userRoles {
					if v == v2 {
						return next(c)
					}
				}
			}
			return domain.ErrForbidden
		}
```

```
		}
	}
	func getUserRolesID(c echo.Context) []int {
		rawSessionData := c.Get(domain.SessionDataKey)
		sessionData, _ := rawSessionData.(domain.SessionData)
		return sessionData.UserRoles
	}
	func NewRouter(e *echo.Echo, c controller.AppController) *echo.Echo {
		e.Use(middleware.Recover())
		advertisementGroup := e.Group("/advertisements", Auth,
Role(model.RoleAdvertisementID, model.RoleAdminID))
		{
			advertisementGroup.GET("", func(context echo.Context) error { return
c.Advertisement.GetAdvertisements(context) })
			advertisementGroup.GET("/:id", func(context echo.Context) error { return
c.Advertisement.GetOneAdvertisement(context) })
			advertisementGroup.PUT("/:id", func(context echo.Context) error { return
c.Advertisement.UpdateAdvertisement(context) })
			advertisementGroup.POST("", func(context echo.Context) error { return
c.Advertisement.CreateAdvertisement(context) })
			advertisementGroup.DELETE("/:id", func(context echo.Context) error { return
c.Advertisement.DeleteAdvertisement(context) })
		}
	}
```

Рисунок 3.6 – Шляхи до оголошень, файл routes.go

Далі було створено структуру даних Advertisement, котра містить інформацію про оголошення. Структури в Go представляють тип даних, що визначається розробником і служать уявленням будь-яких об'єктів (рис. 3.7).

```
	package model
	import "time"
	type Advertisement struct {
		ID uint `gorm:"primary_key" json:"id,omitempty"`
		Title string `json:"title,omitempty"`
		Description string `json:"description,omitempty"`
		Price int `json:"price,omitempty"`
		CreatedAt *time.Time `json:"created_at,omitempty"`
		UpdatedAt *time.Time `json:"updated_at,omitempty"`
		User User `json:"user,omitempty"`
		UserID string `json:"user_id,omitempty"`
	}
```

Рисунок 3.7 – Структура даних оголошення

Наступним кроком було створення контроллеру, який містить створення, пошук, редагування та видалення оголошень (рис. 3.8).

```
package controller
import (
	"GoAds/domain/model"
	"GoAds/usecase/interfactor"
	"errors"
	"log"
	"net/http"
)
type advertisementController struct {
	advertisementInterfactor interfactor.AdvertisementInterfactor
}
type AdvertisementController interface {
	GetAdvertisements(c Context) error
	GetOneAdvertisement(c Context) error
	UpdateAdvertisement(c Context) error
	CreateAdvertisement(c Context) error
	DeleteAdvertisement(c Context) error
}
func NewAdvertisementController(ad interfactor.AdvertisementInterfactor) AdvertisementController {
	return &advertisementController{ad}
}
func (ac *advertisementController) GetAdvertisements(c Context) error {
	var a []*model.GetAdvertisementsResponseData
	var orderQuery string
	offset := c.QueryParam("offset")
	limit := c.QueryParam("limit")
	if limit == "" {
		limit = "10"
	}
	priceSort := c.QueryParam("priceSort")
	if priceSort == "cheap" {
		orderQuery = "price ASC"
	} else if priceSort == "expensive" {
		orderQuery = "price DESC"
	}
	dateSort := c.QueryParam("dateSort")
	if dateSort == "oldest" {
		orderQuery = "advertisements.created_at ASC"
	} else if dateSort == "newest" {
		orderQuery = "advertisements.created_at DESC"
	}
	a, err := ac.advertisementInterfactor.Get(a, limit, offset, orderQuery)
	if err != nil {
		return err
	}
	return c.JSONPretty(http.StatusOK, a, "  ")
}
func (ac *advertisementController) GetOneAdvertisement(c Context) error {
	var a []*model.GetAdvertisementsResponseData
	id := c.Param("id")
	a, err := ac.advertisementInterfactor.GetOne(a, id)
	if err != nil {
		return err
	}
	return c.JSONPretty(http.StatusOK, a, "  ")
}
func (ac *advertisementController) CreateAdvertisement(c Context) error {
	var advertisement model.AdvertisementsRequestData
	advertisement.UserID = getUserID(c)
	err := BindModAndValidate(&advertisement, c)
	if err != nil {
		return c.JSONPretty(http.StatusBadRequest, err.Error(), "")
```

```
		}
		err = ac.advertisementInterfactor.Create(&advertisement)
		if !errors.Is(err, nil) {
			log.Print(err)
			return err
		}
		return c.JSONPretty(http.StatusCreated, "Status 201. Advertisement created", "  ")
	}
	func (ac *advertisementController) UpdateAdvertisement(c Context) error {
		var advertisement model.AdvertisementsUpdateRequestData
		err := BindModAndValidate(&advertisement, c)
		if err != nil {
			return c.JSONPretty(http.StatusBadRequest, err.Error(), "")
		}
		userID := getUserID(c)
		id := c.Param("id")
		_, err = ac.advertisementInterfactor.Update(&advertisement, id, userID)
		if !errors.Is(err, nil) {
			return err
		}
		return c.JSONPretty(http.StatusCreated, "Status 201. Advertisement "+id+" updated", "  ")
	}
	func (ac *advertisementController) DeleteAdvertisement(c Context) error {
		var a []*model.Advertisement
		userID := getUserID(c)
		id := c.Param("id")
		a, err := ac.advertisementInterfactor.Delete(a, id, userID)
		if !errors.Is(err, nil) {
			return err
		}
		return c.JSONPretty(http.StatusOK, "Status 200. Advertisement №"+id+" deleted", "  ")
	}
```

Рисунок 3.8 – Контроллер оголошень

Та останнім кроком було створення репозиторію – це місце, де відправляються запити до бази даних (рис. 3.9). Запити робились за допомогою бібліотеки gorm [13].

```
	package repository
	import (
		"GoAds/domain"
		"GoAds/domain/model"
		"GoAds/usecase/repository"
		"github.com/jinzhu/gorm"
	)
	type advertisementRepository struct {
		db *gorm.DB
	}
	func NewAdvertisementRepository(db *gorm.DB) repository.AdvertisementRepository {
		return &advertisementRepository{db}
	}
	func (ar *advertisementRepository) FindAll(a []*model.GetAdvertisementsResponseData, limit string, offset string, orderQuery string) ([]*model.GetAdvertisementsResponseData, error) {
		err := ar.db.Limit(limit).Offset(offset).Model(&a).Select("title, description, price, created_at, user_id").Order(orderQuery).Preload("User").Find(&a).Error
		if err != nil {
```

```
                return nil, err
        }
        return a, nil
}
func (ar *advertisementRepository) FindOne(a []*model.GetAdvertisementsResponseData, id string) ([]*model.GetAdvertisementsResponseData, error) {
        err := ar.db.Model(&a).Select("*").Where("advertisements.id = ?", id).Preload("User").Preload("Comments.User").Preload("Gallery").Find(&a).Error
        if err != nil {
                return nil, err
        }
        return a, nil
}
func (ar *advertisementRepository) Create(a *model.AdvertisementsRequestData) error {
        err := ar.db.Model(&a).Create(a).Error
        if err != nil {
                if err.Error() == domain.ErrAdvertisementAlreadyWithTitle {
                        return domain.ErrAdvertisementTitleAlreadyExists
                }
                return domain.ErrAdvertisementInternalServerError
        }
        return nil
}
func (ar *advertisementRepository) Update(a *model.AdvertisementsUpdateRequestData, id string, userID string) (*model.AdvertisementsUpdateRequestData, error) {
        result := ar.db.Model(&a).Where("id = ? and user_id = ?", id, userID).Update(a)
        if result.Error != nil {
                if result.Error.Error() == domain.ErrAdvertisementAlreadyWithTitle {
                        return nil, domain.ErrAdvertisementTitleAlreadyExists
                }
                return nil, domain.ErrAdvertisementInternalServerError
        } else if result.RowsAffected == 0 {
                return nil, domain.ErrForbidden
        }
        return a, nil
}
func (ar *advertisementRepository) Delete(a []*model.Advertisement, id string, userID string) ([]*model.Advertisement, error) {
        result := ar.db.Model(&a).Where("id = ? and user_id = ?", id, userID).Delete(&a)
        if result.Error != nil {
                return nil, result.Error
        } else if result.RowsAffected == 0 {
                return nil, domain.ErrForbidden
        }
        return a, nil
}
```

Рисунок 3.9 – Репозиторій оголошень

3.2 Документування та тестування API

Документування потрібно для того, щоб полегшати роботу розробників API та забезпечення більшої взаємодії між розробниками та споживачами API. Для кожного з методів HTTP, потрібно описати параметри запитів та

формат відповідей. Наприклад, для методу GET можна описати параметри запитів, такі як query string parameters, headers, або path parameters, та формат відповіді, такий як JSON.

Метод Get all. Коли надсилається запит на отримання всіх рекламних оголошень, потрібно вказати параметри запиту, їх чотири: limit, offset, dateSort і priceSort. Limit та offset обов'язкові. Сортувати за датою та ціною необов'язково.

Таблиця 3.1 – Таблиця з параметрами до запиту пошуку усіх оголошень

Параметр	Значення	Опис
offset	Числовий	Значення offset вказує пропустити вказану кількість рядків
limit	Числовий	Значення limit обмежує кількість оголошень для показу
dateSort	oldest / newest	Сортує оголошення за датою їх додавання
priceSort	cheap / expensive	Сортує оголошення за ціною

Запит на отримання оголошень можна зробити за прикладом, показаним на рис. 3.10.

```
/ advertisements?offset=0&limit=2
```

Рисунок 3.10 – Приклад запиту

Після цього буде надіслано кілька запитів до бази даних (рис. 3.11-3.12), щоб отримати оголошення та користувачів, щоб сформувати поле автора.

```
SELECT title, description, price, created_at, user_id FROM "advertisements" LIMIT 2 OFFSET 0
```

Рисунок 3.11 – Запит до бази даних для отримання оголошень

```
SELECT * FROM "users" WHERE ("id" IN ('664b4fa3-e467-4f0e-b98b-6b16b2a7818b','e21da3d1-4be8-4d1c-b8d0-50caad941f9c'))
```

Рисунок 3.12 – Запит до бази даних для формування поля автора

Таблиця 3.2 – Можливі відповіді на запит

Статус	Відповідь

200 OK	[
	{
	"title": "Mercedes-Benz E 280 2000",
	"description": "Автомобіль в доброму стані, все працює, повністю обслужений",
	"price": 242216,
	"created_at": "2022-06-20T01:37:12.080793+03:00",
	"author": {
	"id": "664b4fa3-e467-4f0e-b98b-6b16b2a7818b",
	"email": "admin@gmail.com",
	"first_name": "Admin",
	"last_name": "Admin"
	}
	},
	{
	"title": "Mercedes-Benz GLE 350 4 Matic 2016",
	"description": "Продам свій цілий Mercedes-Benz GLE 350 4matic у відмінному майже ідеальному стані. Куплений через Manheim. Є вся історія обслуговування.",
	"price": 1175460,
	"created_at": "2022-06-20T01:37:12.080793+03:00",
	"author": {
	"id": "664b4fa3-e467-4f0e-b98b-6b16b2a7818b",

Статус	Відповідь
	"email": "admin@gmail.com", "first_name": "Admin", "last_name": "Admin" } }]

Продовження табл. 3.2

Статус	Відповідь

500 Internal Server Error	{
	"message": "Invalid or expired access token"
	}

Метод Get one. Коли надсилається запит на отримання одного оголошення, потрібно вказати один параметр запиту – id оголошення, котре потрібно знайти.

Таблиця 3.3 – Таблиця з параметрами до запиту пошуку одного оголошення

Параметр	Значення	Опис
id	Числовий	Ідентифікаційний номер оголошення

Запит на отримання оголошення можна зробити за прикладом, показаним на рис. 3.13.

```
/ advertisements/{id}
```

Рисунок 3.13 – Приклад запиту

Після цього буде надіслано кілька запитів до бази даних (рис. 3.14-3.18), щоб отримати оголошення та користувачів, щоб сформувати поля автора, коментарів, їх авторів та фото за галереї.

```
SELECT * FROM "advertisements"  WHERE (advertisements.id = '2')
```

Рисунок 3.14 – Запит до бази даних для отримання оголошення з ідентифікаційним номером 2

```
SELECT * FROM "users" WHERE ("id" IN ('664b4fa3-e467-4f0e-b98b-6b16b2a7818b'))
```

Рисунок 3.15 – Запит до бази даних для формування поля автора оголошення

```
SELECT * FROM "comments" WHERE ("advertisement_id" IN (2))
```

Рисунок 3.16 – Запит до бази даних для формування поля коментарів

```
SELECT * FROM "users" WHERE ("id" IN ('664b4fa3-e467-4f0e-b98b-6b16b2a7818b'))
```

Рисунок 3.17 – Запит до бази даних для формування поля автора коментарів

```
SELECT * FROM "gallery" WHERE ("advertisement_id" IN (2))
```

Рисунок 3.18 – Запит до бази даних для формування галереї оголошення

Таблиця 3.4 – Можливі відповіді на запит

Статус	Відповідь
200 OK	[{ "id": 10, "title": "Mercedes-Benz S 500 long 2011", "description": "Автомобиль в одной семье с момента покупки. Рестайлинг, чистый 2011 год. Обслуживался только на официальной тех станции. Оригинальный пробег
Статус	Відповідь
	112 000 км. Автомобиль в идеальном состоянии, не требующий вложений. Весь в родной краске, без дтп, без царапин и сколов. Так же под броне плёнкой. Самый максимальный.", "price": 587730, "created_at": "2022-06-20T01:37:12.080793+03:00", "updated_at": "2022-06-20T01:37:12.080793+03:00", "author": { "id": "664b4fa3-e467-4f0e-b98b-6b16b2a7818b", "email": "admin@gmail.com", "first_name": "Admin", "last_name": "Admin" } }]

Продовження табл. 3.4

Статус	Відповідь

404 Not Found	{
	“message”: “Not Found”
	}

500 Internal Server Error	{
	"message": "Invalid or expired access token"
	}

Метод Create. У цьому запиті потрібно відправити JSON body оголошення, котре має містити такі поля, як: назва, опис та ціна.

Також у процесі створення оголошення використовуючи JWT токен, id автора оголошення автоматично буде видобутий з токену. Таким чином, в таблицю з оголошеннями буде додано запис, де буде id автора і вказана вся інформація про оголошення з переданого тіла JSON.

Ви також повинні вказати унікальну назву для оголошення, інакше ви отримаєте відповідь про помилку з повідомленням «Status 400 Bad Request. Title already exists».

Таблиця 3.5 – Таблиця з JSON Body до запиту створення оголошення

Параметр	Значення	Опис
title	Текстовий	Назва оголошення
description	Текстовий	Опис оголошення
price	Числовий	Ціна оголошення

Запит на створення оголошення можна зробити за прикладом, показаним на рис. 3.19.

/ advertisements

Рисунок 3.19 – Приклад запиту

Приклад JSON Body можна побачити на рис. 3.20.

```
{
“title”: “Mercedes S63 AMG Coupe”,
“description”: “Отличное состояние, максимальная комплектация.”,
“price”: “2827165”
}
```

Рисунок 3.20 – Приклад JSON Body

Після цього буде надіслано один запит до бази даних (рис. 3.21), котрий буде містити назву, опис та ціну оголошення, а також ідентифікатор користувача, який створює оголошення, і інформацію про дату та час створення оголошення.

```
INSERT INTO "advertisements" ("title","description","price","created_at","updated_at","user_id") VALUES ('Mercedes S63 AMG Coupe','Отличное состояние, максимальная комплектация.',2827165,'2022-12-18 20:59:56','2022-12-18 20:59:56','664b4fa3-e467-4f0e-b98b-6b16b2a7818b') RETURNING "advertisements"."id"
```

Рисунок 3.21 – Запит до бази даних для створення оголошення

Таблиця 3.6 – Можливі відповіді на запит

Статус	Відповідь

201 Created	{
	"Status 201. Advertisement created"
	}

500 Internal Server Error	{
	"message": "Invalid or expired access token"
	}

Метод Update. У цьому запиті ви повинні надіслати як тіло JSON, так і ідентифікатор оголошення в параметрі запиту одночасно.

Після отримання запиту система визначить ID користувача, який надіслав запит на зміну оголошення. Після чого, якщо ідентифікатор автора

оголошення та ідентифікатор користувача, який надіслав запит, збігаються, він буде змінений відповідно до значень, надісланих у тілі JSON.

Заголовок оголошення також перевірятиметься на унікальність, якщо в базі даних уже створено оголошення із заголовком, який відповідає новому заголовку, надісланому в JSON Body, буде повернена помилка «Status 400 Bad Request. Title already exists».

Якщо ID автора оголошення та ID користувача відрізняються, оголошення не зміниться і прийде відповідь у вигляді помилки «Status 403. Forbidden».

Таблиця 3.7 – Таблиця з параметрами до запиту оновлення оголошення

Параметр	Значення	Опис
id	Числовий	Ідентифікаційний номер оголошення

Таблиця 3.8 – Таблиця з JSON Body до запиту оновлення оголошення

Параметр	Значення	Опис
title	Текстовий	Назва оголошення
description	Текстовий	Опис оголошення
price	Числовий	Ціна оголошення

Запит на отримання оголошення можна зробити за прикладом, показаним на рис. 3.22.

```
/ advertisements/{id}
```

Рисунок 3.22 – Приклад запиту

Приклад JSON Body можна побачити на рис. 3.23.

```
{
“title”: “Mercedes E63s”,
“price”: “3000000”
}
```

Рисунок 3.23 – Приклад JSON Body

Після цього буде надіслано один запит до бази даних (рис. 3.24), котрий буде містити поля з надісланого JSON Body. Також тут буде перевірка, чи співпадають ідентифікаційні номери користувача, який надіслав запит та автора оголошення.

```
UPDATE "advertisements" SET "price" = 3000000, "updated_at" = '2022-12-18 21:15:09'  WHERE (id = '15' and user_id = '664b4fa3-e467-4f0e-b98b-6b16b2a7818b')
```

Рисунок 3.24 – Запит до бази даних для оновлення оголошення

Таблиця 3.9 – Можливі відповіді на запит

Статус	Відповідь

201 Created	{
	"Status 201. Advertisement 10 updated"
	}

403 Forbidden	{
	“Status 403. Forbidden.”
	}

500 Internal Server Error	{
	"message": "Invalid or expired access token"
	}

Метод Delete. Тут у параметрі запиту потрібно вказати id оголошення, після чого система за допомогою токена JWT визначить ідентифікаційний номер користувача, який надіслав запит на видалення оголошення, і якщо ідентифікатори автора та користувача який надіслав запит співпадають, оголошення буде успішно видалено. Але якщо ідентифікатор користувачів відрізняється, буде повернена помилка «Status 403. Forbidden».

Таблиця 3.10 – Таблиця з параметрами до запиту видалення оголошення

Параметр	Значення	Опис
id	Числовий	Ідентифікаційний номер оголошення

Запит на отримання оголошення можна зробити за прикладом, показаним на рис. 3.25.

```
/ advertisements/{id}
```

Рисунок 3.25 – Приклад запиту

Після цього буде надіслано один запит до бази даних (рис. 3.26), тут буде перевірка, чи співпадають ідентифікаційні номери користувача, який надіслав запит та автора оголошення.

```
DELETE FROM "advertisements"  WHERE (id = '15' and user_id = '664b4fa3-e467-4f0e-b98b-6b16b2a7818b')
```

Рисунок 3.26 – Запит до бази даних для оновлення оголошення

Таблиця 3.11 – Можливі відповіді на запит

Статус	Відповідь

200 OK	{
	"Status 200. Advertisement №10 deleted"
	}

403 Forbidden	{
	“Status 403. Forbidden.”
	}

500 Internal Server Error	{
	"message": "Invalid or expired access token"
	}

3.3 Висновки до розділу 3

У цьому розділі було описано процес створення проєкту, у якості прикладу було розглянуто приклад первинного налаштування проєкту, підключення проєкту до бази даних, перший запуск проєкту на локальному

серверi та етапи створення, оновлення, видалення та пошуку оголошень. Також API було описано та протестовано.

ВИСНОВКИ

Отже, в результаті виконання кваліфікаційної роботи було розроблено JSON API для сайту оголошень «GoAds» з використанням мови програмування Golang та фреймворку Echo. Для зберігання інформації було обрано СКБД PostgreSQL. За допомогою технологій Draw IO було розроблено концептуальне проєктування та сформовано технічне завдання.

Згідно поставлених завдань було проаналізовано функціональні та технічні вимоги API. В результаті чого було побудовано діаграму прецедентів та сутностей бази даних.

Було проаналізовано предметну область та спроєктовано ER-діаграму для зберігання інформації в системі. Розгорнуто новий проєкт Golang та реалізовано всі необхідні функції бізнес логіки у вигляді контролера.

Було виконано тестування та опис API.

API досить функціональний прототип, але для реального використання, його може бути вдосконалено та розроблено front-end частину.

ПЕРЕЛІК ПОСИЛАНЬ

1. The Go Programming Language. URL: https://go.dev (дата звернення: 15.03.2023)

2. Leiva N., Kashin M. Network Automation with Go: Learn how to automate network operations and build applications using the Go programming language. Birmingham : Packt Publishing, 2023. 442 p.

3. Bodner J. Learning Go: An Idiomatic Approach to Real-World Go Programming. Sebastopol : O'Reilly Media, 2021. 375 p.

4. McGrath M. GO Programming in easy steps: Learn coding with Google’s Go language. Warwickshire : In Easy Steps Limited, 2020. 192 p.

5. Mohamed Labouardy. Building Distributed Applications in Gin: A hands-on guide for Go developers to build and deploy distributed web apps with the Gin framework. Birmingham : Packt Publishing, 2021. 482 p.

6. David Li. Building Web Applications with Go and Beego: An Introduction to Backend Development. Birmingham : On Kindle Scribe, 2023. 350 p.

7. Hans-Jurgen Schonig. Mastering PostgreSQL 15: Advanced techniques to build and manage scalable, reliable, and fault-tolerant database applications, 5th Edition. Birmingham : Packt Publishing, 2023. 522 p.

8. JetBrains GoLand: не просто IDE для Go. URL: https://www.jetbrains.com/ru-ru/go/ (дата звернення: 30.03.2023)

9. Top 5 Golang Frameworks in 2022. URL: https://www.geeksforgeeks.org/top-5-golang-frameworks-in-2020/ (дата звернення: 03.04.2023)

10. Go | Руководство. URL: https://metanit.com/go/tutorial/ (дата звернення: 21.04.2023)

11. PostgreSQL: About. URL: https://www.postgresql.org/about/ (дата звернення: 30.04.2023)

12. Draw IO. URL: https://app.diagrams.net/ (дата звернення: 14.04.2023)

13. The fantastic ORM library for Golang. URL: https://gorm.io (дата звернення: 19.04.2023)