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

Випускна кваліфікаційна робота бакалавра

«СИСТЕМА УПРАВЛІННЯ РЕЗЕРВУВАННЯМ РЕСУРСІВ (ЗА ПРИКЛАДОМ СЕРВІСУ AIRBNB) (BACKTEND)»

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

Виконавці проекту

№	П.І.Б.	Група
1	Карпов Д.Р.	КН-П-201
2	Лупашко П.П.	КН-П-201
3	Козлов Я.Я.	КН-Д-202
4	Воробьев А.І.	КН-Д-202
5	Гольденберг В.О	КН-А-201

Автор звіту

_____ Лупашко Павло Павлович

Одеса – 2024

АНОТАЦІЯ

Проект був започаткований для створення нової системи онлайн-бронювання, яка базується на існуючій платформі airbnb.com. Функціональні можливості та інтерфейс користувача були модернізовані. Розробка нової системи розпочалася з урахуванням вимог керівників департаментів та основної концепції поточної системи airbnb.com.

Ця робота описує дослідження, розробку та тестування системи бронювання, що має на меті полегшити процес купівлі-продажу послуг між користувачами та компаніями. Система пропонує широкий спектр функцій, зокрема:

- **Реєстрація та авторизація:** Користувачі та компанії можуть легко створити облікові записи та отримати доступ до системи.
- **Аутентифікація:** Надійні механізми аутентифікації гарантують безпеку користувачів та захист їхніх даних.
- **Оренда:** Користувачі можуть шукати та бронювати послуги, пропоновані компаніями, на платформі.
- **Додавання та управління готелями:** Компанії можуть додавати інформацію про свої готелі, керувати ними та приймати онлайн-бронювання.
- **Доступ до інформації:** Користувачі та компанії мають доступ до інформації про свої бронювання, клієнтів та зароблені кошти.

Головними завданнями були покращення зручності використання інтерфейсу системи. У цій роботі об'єктом дослідження є алгоритм, що забезпечує комплексну оцінку результатів за численними критеріями в умовах

невизначеності початкових даних. Предмет дослідження включає фактор, який дозволяє користувачеві легко подорожувати та комбінувати об'єкти і рейси найпростішим можливим способом.

Цілями дослідження є:

- Огляд систем бронювання, що дозволяють користувачам взаємодіяти з іншими користувачами та компаніями без необхідності проходити складні процедури.
- Розробка програмного забезпечення з урахуванням досягнутих раніше результатів, а також проведення його тестування в різних умовах і на різних пристроях.
- Впровадження програмного забезпечення для захисту даних у систему з метою підвищення рівня інформаційної безпеки створеної системи.
- Методи дослідження, що застосовуються для досягнення цілей, базуються на комбінації методів системного аналізу, моделювання та когнітивної методології.

Для розробки клієнтської частини було використано веб-фреймворк Angular разом з HTML, TypeScript і SCSS. Ця дослідницька робота зосереджена на створенні системи бронювання з підтримкою специфічного пошукового фільтру.

ЗМІСТ

ВСТУП

Все більше людей використовують комп'ютери та мобільні пристрої в своєму повсякденному житті завдяки швидкому зростанню Інтернету за останні роки. Бронювання послуг стало однією з таких завдань. На ринку систем бронювання домінують кілька розробників програмного забезпечення та компаній, і їхні послуги корисні в багатьох ситуаціях. Системи бронювання застосовуються повсюди: в ресторанах, лікарнях, перукарнях, школах та інших установах. Вони можуть бути дуже складними через різноманітність пристроїв та користувацьких уподобань, що ускладнює процес розробки.

Дуже гнучкі системи бронювання часто є надто складними для користувачів та адміністрації. Це призводить до зайвих кліків та активації непотрібних функцій. Системи бронювання повинні бути логічними і простими у використанні як для ринку, так і для користувача.

У рамках даного звіту розглядається клієнтська частина, а також описується весь цикл її розробки. Звіт складається з трьох розділів: загальні характеристики системи, деталі створення клієнтської частини, результати розгортання і тестування. Висновки відображають досягнуті результати в цій частині проекту, а загальний висновок є сукупністю звітів всіх учасників проекту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

СИСТЕМА УПРАВЛІННЯ РЕЗЕРВУВАННЯМ РЕСУРСІВ (за прикладом сервісу airbnb.com)

Загальний опис

Програмне забезпечення (ПЗ) призначене для створення розподіленої системи керування ресурсами, що забезпечує можливість проактивного управління, включаючи резервування. Функції системи покривають повний життєвий цикл управління ресурсами: створення, актуалізація, використання та видалення. Головна особливість системи полягає в тому, що пріоритет використання обмежених ресурсів залежить від терміну їх резервування.

Призначення

Програмне забезпечення орієнтоване на створення контейнера для управління обмеженими ресурсами на основі аналізу їх резервування. Обробка конкурентних запитів на обмежені ресурси повинна враховувати показники пріоритетності, серед яких важливим є часовий показник моменту резервування. ПЗ розробляється за прикладом сервісу бронювання обмеженої кількості місць, але має бути універсальним і здатним змінювати своє цільове призначення.

Вимоги до функціоналу

1. Головна сторінка

На головній сторінці системи забезпечено відображення фільтрів, пошуку і каталогу апартаментів, що значно полегшує користувачам доступ до необхідної інформації та сприяє їхній ефективній навігації по сервісу. Завдяки цим функціональним можливостям користувачі можуть швидко знаходити потрібні об'єкти для бронювання, використовуючи зручні фільтри і швидкий пошук. Крім того, на головній сторінці знаходяться інструменти для авторизації та реєстрації нових користувачів або посилання на них, що робить процес взаємодії з системою ще більш зручним і простим. Це дозволяє новим користувачам швидко і без зайвих зусиль зареєструватися та отримати доступ до усіх функцій сервісу.

2. Реєстрація користувачів

Користувач ПЗ має змогу зареєструватись як за допомогою авторизаційних даних (логіну/паролю), так і за допомогою облікових записів популярних соціальних мереж. При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - телефон або електронна пошта. Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.

3. Авторизація та особистий кабінет користувача

За фактом успішної авторизації користувачу надається доступ до особистого кабінету. Це окремий інформаційний ресурс (сторінка), що являє собою основне робоче місце користувача ПЗ. Повинен забезпечити усі необхідні засоби для створення замовлень. Також засобами особистого кабінету має бути передбачено можливість зміни персональних даних.

4. Процес бронювання

Користувач повинен мати можливість перейти на сторінку будь-якого із запропонованих на головній сторінці готелів, отримати на ній ціну за ніч, опис, фото та всю іншу інформацію про готель. Користувач має бути забезпечений інтерфейсом для вибору дат і кількості гостей для бронювання на сторінці апартаментів. Після заповнення цих даних, має бути сторінка бронювання з необхідною інформацією для здійснення бронювання, а саме інформація про банківську картку, повідомлення компанії готелю, фото і номер телефону.

Технології та інструменти

1. Мови програмування

- **C#**: Основна мова програмування для серверної частини, що забезпечує високу продуктивність та безпеку.

2. База даних

- **MySQL**: Використовується як основна база даних. Дані нормалізовані за трьома формами. Зображення зберігаються окремо від бази даних.
- **MySQL Workbench**: Інструмент для адміністрування бази даних та написання складних SQL-запитів.

3. Фреймворк

- **ASP.NET Core**: Використовується для створення веб-додатків та API.

Основні інструменти фреймворку:

- **Сервіси**: Концепція сервісів для розбиття логіки проекту на окремі модулі з використанням dependency injection.
- **Атрібути**: Налаштування логіки поведінки контролерів та валідації даних.
- **Middleware**: Використання GlobalExceptionHandlingMiddleware для обробки виключень.
- **CORS**: Налаштування крос-доменного доступу.
- **appsettings.json та secrets.json**: Безпечне та зручне налаштування серверу.

4. Бібліотеки

- **ASP.NET Identity**: Використовується для реєстрації, автентифікації, авторизації, валідації даних користувача, хешування та зберігання паролів, керування ролями. Інтеграція з JWT для автентифікації та авторизації.
- **EF Core**: Використовується для роботи з базою даних. Code First стратегія, AsNoTracking та IQueryable для оптимізації роботи з базою даних.
- **LINQ**: Використовується разом з EF Core для написання запитів до бази даних.
- **Swagger**: Використовується для демонстрації та тестування API на етапі розробки.
- **Google OAuth 2.0**: Використовується для інтеграції соціальних мереж та інших акаунтів.

5. Інші інструменти

- **Docker**: Використовується для створення контейнерів, що дозволяє ефективно розгортати додаток на будь-якій системі. Створені Dockerfile та docker-compose.yaml.
- **Git**: Використовується для контролю версій коду, підтримки командної розробки.
- **GitHub**: Використовується для розміщення проекту та налаштування CI/CD.

Вимоги до функціоналу. Опціональна частина

1. Розміщення та управління

Компанія має можливість додати свій готель або апартаменти через спеціальну сторінку, де вона заповнює всю необхідну інформацію про об'єкт. Ця процедура не лише спрощує процес реєстрації нових об'єктів, але й забезпечує повну інтеграцію з системою управління, що дозволяє компанії ефективно керувати усіма аспектами своєї гостьової діяльності.

Крім цього, компанія має доступ до спеціальної сторінки хостингу, де вона може здійснювати моніторинг і управління своїми готелями і клієнтами. Ця функція не тільки сприяє оптимальному використанню ресурсів і підвищує ефективність обслуговування гостей, але і забезпечує компанії зручний інтерфейс для взаємодії з її власною мережею готелів.

Архітектура

ПЗ має бути спроектоване за розподіленою клієнт-серверною архітектурою з чітко відокремленими розподіленими вузлами системи:

- **Mockup**: Візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.
- **Backend**: Серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.
- **Web App**: Додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів усіх категорій.

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контролю версій (вибір конкретної системи узгоджується групою проекту).

Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:

- **root**
    - **Backend**
    - **Web**
    - **Mockup** (вихідні файли або посилання на зовнішній проект)

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

- **Web**: Angular, TypeScript, SCSS, HTML
- **Mockup**: Визначається ментором з дизайну

Команда проекту

Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проекту. Мінімальна команда проекту - два учасники, у такому складі один учасник виконує модуль Backend, другий - Frontend. Рекомендована команда проекту - три учасники, у такій команді кожен з учасників відповідає за одну з частин: Mockup, Backend, Frontend.

Звітність

За результатами виконання завдань із створення ПЗ кожен з учасників команди проекту 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 проекту, 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

Архітектура платформи складається з таких частин:

«Frontend».

«Design»,

«Backend»,

BACKEND

Імперативним аспектом веб-архітектури є те, що серверна частина повинна забезпечувати високу продуктивність, безпеку та масштабованість протягом усієї платформи. Кожна функціональність повинна бути розроблена з урахуванням цих фундаментальних принципів.

Платформа підтримує REST API, що дозволяє забезпечити максимальну гнучкість та інтеграцію з іншими системами. REST API дозволяє уникнути перевантаження серверу при обробці великої кількості запитів та забезпечує швидкий доступ до ресурсів.

Основні вимоги до бекенд-системи включають:

1. **Безпека**: Всі дані користувачів повинні бути захищені, а доступ до ресурсів повинен бути контролюваний за допомогою рольової авторизації.
2. **Масштабованість**: Система повинна легко масштабуватися для обробки зростаючої кількості користувачів та даних.
3. **Продуктивність**: Висока швидкість обробки запитів та мінімальний час відгуку на запити користувачів.

Для виконання цих критеріїв необхідно, щоб розробник бекенду розумів та застосовував принципи проектування високопродуктивних систем. Нижче наведено основні принципи та патерни, які були використані при розробці бекенду:

1. REST API Architecture

Архітектура REST API була обрана для забезпечення простої та гнучкої взаємодії між клієнтськими та серверними додатками. REST API використовує стандартні HTTP методи (GET, POST, PUT, DELETE) для виконання CRUD операцій (створення, читання, оновлення, видалення) над ресурсами.

2. N-Layered Architecture

Проект був розподілений на наступні шари:

- **View Layer**: Контролери та точки входу, що відповідають за обробку HTTP-запитів та повернення відповідей.
- **Business Logic Layer**: Сервіси, що реалізують бізнес-логіку додатку.
- **Data Access Layer**: Взаємодія з базою даних через репозиторії.

- **Core Layer**: Зберігання основних сутностей та констант, що використовуються у всіх інших шарах.

3. Role-Based Authorization Using JWT Token

Для забезпечення безпеки та контролю доступу до ресурсів була реалізована рольова авторизація з використанням JWT (JSON Web Token). Це дозволяє призначати користувачам певні ролі та обмежувати доступ до ендпоінтів на основі цих ролей.

4. Repository Pattern

Репозиторний патерн забезпечує абстракцію від конкретної бази даних та дозволяє легко змінювати або розширювати логіку доступу до даних. Це забезпечує чітке розподілення відповідальності роботи з кожною сутністю.

5. Unit of Work Pattern

Патерн "Unit of Work" дозволяє об'єднувати операції над кількома репозиторіями в одну транзакцію, що забезпечує атомарність та цілісність даних.

6. Chain of Command Pattern (Middleware)

Middleware реалізує патерн "Chain of Command", що дозволяє розподіляти обробку запитів на кілька окремих етапів, кожен з яких виконує свою специфічну функцію, наприклад, автентифікацію, логування або обробку помилок.

7. Decorator Pattern

Декораторний патерн дозволяє додавати нові можливості та поведінку до об'єктів динамічно, підтримуючи принципи гнучкості та повторного використання коду.

8. SOLID Principles

Принципи SOLID забезпечують високу якість коду та легкість його підтримки:

- **Single Responsibility Principle (SRP)**
- **Open/Closed Principle (OCP)**
- **Liskov Substitution Principle (LSP)**
- **Interface Segregation Principle (ISP)**
- **Dependency Inversion Principle (DIP)**

9. Low Coupling, High Cohesion

Код розроблений з низьким рівнем зв'язності та високим рівнем когезії, що забезпечує кращу підтримку та розвиток системи.

10. DRY Principle

Принцип "Don't Repeat Yourself" (DRY) дозволяє уникнути дублювання коду, забезпечуючи меншу кількість помилок та спрощуючи підтримку коду.

11. KISS Principle

Принцип "Keep It Simple, Stupid" (KISS) забезпечує простоту та зрозумілість коду, зменшуючи його складність та покращуючи читабельність.

12. YAGNI Principle

Принцип "You Aren't Gonna Need It" (YAGNI) дозволяє уникнути розробки функціональності, яка може ніколи не знадобитися, зосереджуючись на поточних потребах проекту.

РОЗДІЛ 2

РЕАЛІЗАЦІЯ BACKEND

Відповідно до моделі «фази розробки», розробка бекенду була складним та багатогранним процесом. З часом ми покращили безпеку та функціональність, оскільки це був перший масштабний онлайн-додаток, який ми розробляли. Кожен великий етап розробки цього додатка був ретельно протестований.

1. Вимоги до розробки

Розробка бекенд-системи вимагала чіткого визначення вимог для забезпечення високої надійності та безпеки. Основні вимоги до розробки включали:

- Можливість бронювання помешкань з підтвердженням через електронну пошту.
- Реєстрація користувачів з автентифікацією та авторизацією.
- Підтвердження електронної пошти та можливість зміни паролю з підтвердженням.
- Забезпечення доступу до певних ендпоінтів тільки для користувачів з відповідною роллю.
- Інтеграція з Google для створення акаунтів (OAuth 2.0).
- Можливість оновлення персональних даних користувача.

- Перегляд та фільтрація помешкань з підтримкою пагінації.
- Створення нових помешкань лише зареєстрованими користувачами з підтвердженою поштою.

2. Вибір інструментів розробки

Для розробки бекенду були обрані наступні технології:

- **Node.js**: Вибраний як основний середовище для виконання JavaScript на серверній стороні, забезпечуючи високу продуктивність та масштабованість.
- **Express.js**: Використаний для створення веб-фреймворку з метою спрощення маршрутизації та обробки HTTP-запитів.
- **MongoDB**: Обраний як база даних завдяки своїй гнучкості та можливості зберігати дані у вигляді документів.
- **OAuth 2.0**: Використаний для забезпечення безпечної автентифікації та авторизації через Google.

3. Реалізація функціональності

3.1 Бронювання помешкань

Користувачі можуть бронювати помешкання через зручний інтерфейс. Після успішного бронювання надсилається підтвердження на електронну пошту. Для цього було реалізовано інтеграцію з сервісом відправки електронних листів, що забезпечує своєчасну доставку повідомлень користувачам.

3.2 Реєстрація, автентифікація, авторизація

Користувачі можуть реєструватися, автентифікуватися та авторизуватися на платформі. Реалізовано функціонал підтвердження електронної пошти для

забезпечення додаткової безпеки. Для зміни електронної пошти необхідно підтвердження з попередньої адреси. Зміна паролю можлива тільки після введення поточного паролю, що додає рівень захисту.

3.3 Підтвердження пошти

Механізм підтвердження електронної пошти реалізовано через відправку повідомлення з посиланням для підтвердження. Це дозволяє забезпечити достовірність вказаних даних користувачем.

3.4 Доступ до ендпоінтів

Доступ до певних ендпоінтів можливий лише для користувачів з відповідною роллю. Це реалізовано через використання middleware, яке перевіряє ролі користувачів перед обробкою запитів.

3.5 Використання Google для створення аккаунту (OAuth 2.0)

Інтеграція з Google через OAuth 2.0 дозволяє користувачам швидко створювати акаунти та автентифікуватися, використовуючи свої Google-акаунти. Це значно спрощує процес реєстрації для користувачів.

3.6 Оновлення даних користувача

Користувачі можуть оновлювати свої персональні дані, такі як ім’я, електронну пошту, пароль та зображення профілю. Реалізовано перевірку та підтвердження змін для забезпечення безпеки даних.

3.7 Перегляд та фільтрація помешкань

Користувачі можуть переглядати доступні помешкання та фільтрувати їх за різними критеріями, такими як місцезнаходження, ціна, кількість кімнат тощо.

Для оптимізації процесу додано пагінацію, яка дозволяє зменшити навантаження на сервер при обробці великих обсягів даних.

3.8 Створення нових помешкань

Створення нових помешкань можливе тільки для зареєстрованих користувачів з підтвердженою електронною поштою. Це забезпечує контроль за якістю та достовірністю інформації, що додається на платформу.

3.9 Перегляд власних помешкань

Користувачі можуть переглядати список своїх власних помешкань, керувати ними та оновлювати інформацію.

3.10 Додавання помешкань у вішліст

Користувачі можуть додавати помешкання до списку бажаних для подальшого перегляду та бронювання. Це дозволяє зберігати улюблені варіанти та швидко до них повертатися.

4. Інтеграція API

Реалізовано інтеграцію з API для обміну даними між клієнтським та серверним додатками. Це включає мережеві запити для отримання, збереження та оновлення інформації про користувачів та помешкання.

5. Підходи

5.1 REST API Architecture

Архітектура REST API була обрана для розробки бекенду, оскільки вона

забезпечує простоту та гнучкість у взаємодії між клієнтськими та серверними додатками. REST API дозволяє використовувати HTTP методи (GET, POST, PUT, DELETE) для виконання CRUD операцій (створення, читання, оновлення, видалення) над ресурсами. Це забезпечує масштабованість та легкість інтеграції з іншими системами.

5.2 N-Layered Architecture

Для структуризації коду та розподілу відповідальностей ми використали N-рівневу архітектуру. Проект був розподілений на наступні шари:

- **View Layer**: Включає контролери та точки входу, що відповідають за обробку HTTP-запитів та повернення відповідей.
- **Business Logic Layer**: Містить сервіси, що реалізують бізнес-логіку додатку.
- **Data Access Layer**: Відповідає за взаємодію з базою даних, включаючи репозиторії для роботи з сутностями.
- **Core Layer**: Зберігає основні сутності та константи, що використовуються у всіх інших шарах.

5.3 Role-Based Authorization Using JWT Token

Для забезпечення безпеки та контролю доступу до ресурсів була реалізована рольова авторизація з використанням JWT (JSON Web Token). Це дозволяє призначати користувачам певні ролі та обмежувати доступ до ендпоінтів на основі цих ролей. JWT токен генерується під час автентифікації та передається з кожним запитом, що дозволяє серверу перевіряти права доступу.

5.4 Repository Pattern

Репозиторний патерн був використаний для розподілення відповідальності роботи з кожною сутністю. Репозиторії забезпечують абстракцію від конкретної бази даних та дозволяють легко змінювати або розширювати логіку доступу до даних.

5.5 Unit of Work Pattern

Патерн "Unit of Work" був застосований для вирішення проблеми взаємодії декількох сутностей та транзакцій. Він дозволяє об'єднувати операції над кількома репозиторіями в одну транзакцію, що забезпечує атомарність та цілісність даних.

5.6 Chain of Command Pattern (Middleware)

Патерн "Chain of Command" був реалізований у вигляді middleware для обробки запитів. Middleware дозволяє розподіляти обробку запитів на кілька окремих етапів, кожен з яких виконує свою специфічну функцію, наприклад, автентифікацію, логування або обробку помилок.

5.7 Decorator Pattern

Декораторний патерн був використаний для розширення функціональності існуючих класів без зміни їх вихідного коду. Це дозволяє додавати нові можливості та поведінку до об'єктів динамічно, підтримуючи принципи гнучкості та повторного використання коду.

6. Принципи розробки

6.1 SOLID Principles

Принципи SOLID були дотримані для забезпечення високої якості коду:

- **Single Responsibility Principle (SRP)**: Кожен клас має лише одну відповідальність.
- **Open/Closed Principle (OCP)**: Класи відкриті для розширення, але закриті для модифікації.
- **Liskov Substitution Principle (LSP)**: Об'єкти підкласів повинні заміняти об'єкти батьківських класів без порушення функціональності.
- **Interface Segregation Principle (ISP)**: Користувачі не повинні залежати від інтерфейсів, які вони не використовують.
- **Dependency Inversion Principle (DIP)**: Високорівневі модулі не повинні залежати від низькорівневих модулів. Обидва типи модулів повинні залежати від абстракцій.

6.2 Low Coupling, High Cohesion

Код був розроблений з низьким рівнем зв'язності та високим рівнем когезії, що забезпечує кращу підтримку та розвиток системи. Низька зв'язність означає, що компоненти системи мають мінімальні залежності один від одного, а висока когезія означає, що компоненти мають добре визначені, вузько спрямовані обов'язки.

6.3 DRY Principle

Принцип "Don't Repeat Yourself" (DRY) був дотриманий для уникнення дублювання коду. Вся спільна логіка була винесена в окремі методи або сервіси, що дозволяє зменшити кількість помилок та спростити підтримку коду.

6.4 KISS Principle

Принцип "Keep It Simple, Stupid" (KISS) був використаний для забезпечення

простоти та зрозумілості коду. Це дозволяє зменшити складність системи та покращити читабельність коду.

6.5 YAGNI Principle

Принцип "You Aren't Gonna Need It" (YAGNI) був дотриманий для уникнення розробки функціональності, яка може ніколи не знадобитися. Це дозволяє зосередитися на поточних потребах проекту та уникнути зайвих витрат часу та ресурсів.

РОЗДІЛ 3

ОЦІНКА

Потреби серверної частини повинні бути детально розглянуті після ретельного проектування архітектури програми та розробки її основних компонентів. Оцінка потреб користувачів може слугувати основою для формулювання майбутніх цілей і напрямків розвитку. Важливо розуміти, що кожен етап розробки має враховувати вимоги користувачів, адже саме вони визначають кінцевий успіх проєкту.

У розділах «ЗАГАЛЬНІ ХАРАКТЕРИСТИКИ СИСТЕМИ» і «РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ» ми узагальнили початкові критерії розробки веб-додатка. В цих розділах ми детально розглядаємо вимоги як в контексті архітектури додатка, так і в контексті його реалізації. Архітектура додатка є основою, яка визначає структуру та взаємодію між компонентами, а реалізація

серверної частини охоплює аспект забезпечення надійності, продуктивності та безпеки.

Однак важливо пам'ятати, що навіть найретельніше планування не обов'язково гарантує успіх. Кожен етап розробки повинен включати постійну перевірку і коригування, щоб відповідати змінним вимогам користувачів і новим технологічним викликам. Залучення користувачів до процесу оцінки і тестування може надати цінну інформацію для покращення функціональності і зручності використання веб-додатка, що в кінцевому рахунку сприятиме досягненню його успіху на ринку.

Основи

Спочатку платформа airbnb.com була взята за приклад для створення нашої платформи. Зокрема, дизайн і функціональність повинні були бути схожими на airbnb.com, з кількома покращеннями макета і виправленнями. Враховуючи це в контексті архітектури додатків, є багато рішень, які потрібно розглянути. Створення численних класів стилів, що реалізують бажану функціональність, може виконати цю вимогу. Функціональність односторінкового додатка є ідеальним рішенням для нашого випадку. Коли програма розроблена з обраним підходом, легше регулювати доступ, оскільки користувачам можна обмежити доступ до певних компонентів. Таким чином, сайт може бути адаптований до інформаційних вимог конкретного користувача та потреб бізнесу. Архітектура на основі компонентів дозволяє розробляти кожен компонент окремо, що значно полегшує процес розробки програмного забезпечення. Вимога може вважатися виконаною в поточному контексті, оскільки вона була реалізована під час впровадження. В результаті обраної архітектури один додаток можна розробити швидше, оскільки розміром

проекту легше керувати.

Масштабованість

Вимога до розширення веб-додатків тісно пов'язана з початковими критеріями, оскільки інформація з "Основи" легко дозволяє масштабувати існуючий додаток. Від того, як побудований фундамент, залежить легкість масштабування. Додавання нових компонентів до додатка може ускладнити його роботу і викликати проблеми. У багатьох випадках ці проблеми виникають через залежності між класами або через невиявлені помилки під час розробки. Ці помилки можуть призвести до краху всієї системи у виробничому середовищі. Добре продумана архітектура спрощує масштабування і запобігає виникненню проблем. Компоненти веб-додатків є самостійними, тому додавання нових не повинно впливати на існуючі компоненти, за умови правильного управління взаємодією між ними. Сильна залежність між компонентами може викликати труднощі. Окремі компоненти веб-додатка також легко підтримувати завдяки створеній архітектурі. Уражену частину додатка можна виправити, якщо помилки виявлені у виробничому середовищі.

Коли вимога розглядається в контексті реалізації, її можна вважати виконаною. Додавання нових компонентів до веб-додатка досить просто завдяки функціональним можливостям, наданим основним фреймворком. Основний обсяг роботи, необхідної для забезпечення функціонування додатка як компонентного веб-додатка, виконується безпосередньо у самому компоненті. Кожен компонент вимагає певного налаштування, а точка входу додатка повинна бути визначена заздалегідь. Управління всіма основними конфігураціями може створити проблеми в майбутньому, якщо кількість

компонентів значно зросте.

Узгодженість інтерфейсу користувача та досвіду

Вимога узгодженості інтерфейсу користувача і користувацького досвіду у веб-додатку була розбита на кілька розділів. Розглянуті компоненти стосуються таких проблем, як єдність окремих елементів інтерфейсу користувача та більш широкого дизайну інтерфейсу користувача.

Перший компонент критеріїв показує, що повинен бути обмежений набір загальних стилів макета, що всі види по всьому веб-додатку повинні дотримуватися зазначених правил. Загальний дизайн доступних на даний момент додатків дотримується однакових правил.

Елементи інтерфейсу користувача, що використовуються для взаємодії з користувачем, повинні бути уніфіковані для кожного випадку використання відповідно до другої частини вимоги. Це означає, що на кожній сторінці всі текстові поля і кнопка, що використовуються для додавання нового елемента, повинні виглядати однаково. Розроблений дизайн просто допомагає розробникам правильно впроваджувати елементи інтерфейсу користувача, але вимагає відданості дизайнера в команді.

У третьому розділі критеріїв зазначено, що позиціонування елементів інтерфейсу користувача має бути узгодженим у всіх перспективах. Наприклад, якщо елемент рядка пошуку присутній поверх таблиці даних, він повинен бути присутнім у всіх місцях. Коли розташування елементів довільно змінюються в різних місцях, користувачеві стає складніше ефективно керувати додатком. Знову ж таки, система дизайну може допомогти розробникам правильно реалізувати користувальницький інтерфейс, але їх створення

вимагає зобов'язань дизайнера і знання.

Четвертий розділ вимоги стосується питання масштабування елементів інтерфейсу користувача. Знову ж таки, розроблена система дизайну допомагає розробникам, пропонуючи класи корисних стилів, які можуть бути використані для опису розміру елемента. Цю умову можна вважати частково виконаною, з урахуванням поточних компонентів, а також самого веб-додатка. Ця проблема повинна бути вирішена в майбутньому, як і розширення нинішньої системи проектування з новими загальними компонентами для кращого задоволення цілей онлайн-програми.

Можливі майбутні вдосконалення сайту

Незважаючи на те, що кілька проблем були виявлені і згодом вирішені під час роботи над цим продуктом, є ще кілька, які повинні бути вирішені в майбутньому.

Їх проблеми з іншими альтернативними компонентами будуть обмежені самим компонентом. Оскільки наявні компоненти підтримують кешування та повторне отримання даних, інші можливі компоненти повинні реалізувати можливість забезпечити кращий та більш чуйний користувальницький досвід. Нові компоненти також повинні бути розділені на менші частини, щоб зробити їх більш зрозумілими та спростити подальшу розробку.

Однією з проблем, яка може виникнути в майбутньому, є розмір компонентів, які складають сайт. Якщо розміри компонентів стають занадто великими, перехід від однієї частини програми до іншої може зайняти багато часу. Якщо один клієнт має велику кількість відкритих вікон і занадто багато з них завантажуються або підтримуються в пам'яті одночасно, це може викликати

проблеми. Для вирішення цих проблем необхідно розробити механізм використання простору веб-застосунку.

До технічних проблем, які залишаються у веб-додатку, як зазначалося раніше, ще належить їх виправити. Кількість спільних компонентів в системі проектування в даний час досить мінімальна. Цього слід дотримуватися, особливо з огляду на необхідність послідовного інтерфейсу користувача та досвіду по всій онлайн-програмі.

ВИСНОВКИ

Система, побудована і реалізована під час написання цього документу, все ще знаходиться на ранній стадії, з багатьма компонентами, які можна поліпшити і додати. Робота, виконана для серверної частини додатку, була повністю пояснена у документі, і, як зазначається в розділі «ОЦІНКА», було зроблено багато роботи, але в майбутньому доведеться зробити ще більше. Сподіваємося, що недоліки системи та відсутні елементи будуть вирішені в майбутньому.

З огляду на початкову точку процесу проектування архітектури і висновки, наведені в розділі «ОЦІНКА», можна з упевненістю стверджувати, що починати процес проектування після значного обсягу реалізації не бажано. Найскладніші моменти розробки, що виникли, були пов'язані з тим, що був значний обсяг відсутніх робіт для проектної системи. Наприклад, відсутність системи проектування і процес включення її в існуюче застосування потребували найбільших зусиль.

Можна навіть стверджувати, що це вимагало більших зусиль, ніж решта реалізації. В результаті ігнорування дизайну можна мати серйозні наслідки в майбутньому. Це питання слід вирішувати і краще враховувати в майбутніх подіях.

Дуже важливо розробити систему дизайну як частину загальної архітектури, оскільки вона приносить користь як кінцевому користувачеві, так і розробнику. Це пов'язано з тим, що система проектування допомагає розробнику створити більш узгоджений користувальницький досвід з меншою роботою, використовуючи спільні компоненти системи проектування та інші утиліти. Оскільки елементи інтерфейсу користувача відомі по всій системі, узгодженість інтерфейсу користувача дозволяє кінцевому користувачеві більш ефективно керувати системою.

Регулювання кількості деталізації веб-програми, використовуючи фреймворк ASP.NET Core як основу для серверної частини додатків, дозволило повністю використовувати існуючі компоненти, які раніше були реалізовані до початку архітектурного проектування.

На даний момент немає вагомих причин критикувати архітектуру системи. Однією з переваг використання фреймворку ASP.NET Core в серверній частині, як зазначалося раніше, є те, що додатки технологічно незалежні. Це дозволяє легко оновлювати та змінювати технології, що використовуються в процесі розробки додатків. В результаті можлива найкраща технологія для кожної даної програми.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ КЛІЄНТСЬКА ЧАСТИНИ

	Ресурс
1.	C#; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/dotnet/csharp/
2.	T-SQL; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/sql/t-sql/language-elements
3.	ASP.NET; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/aspnet/core
4.	ASP.NET Core; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/aspnet/core

5.	ASP.NET Identity; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity

6.	EF Core; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/ef/core
7.	LINQ; Розробник: Microsoft; Документація: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq
8.	Swagger; Розробник: SmartBear; Документація: https://swagger.io/docs/
9.	Docker; Розробник: Docker Inc.; Документація: https://docs.docker.com/
10.	Google OAuth 2.0; Розробник: Google; Документація: https://developers.google.com/identity/protocols/oauth2
11.	GitHub; Розробник: GitHub; Документація: https://docs.github.com/
12.	MySQL; Розробник: Oracle; Документація: https://dev.mysql.com/doc/

13.	MySQL Workbench; Розробник: Oracle; Документація: https://dev.mysql.com/doc/workbench/en/