ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: **“РОЗРОБКА ДОДАТКУ ДЛЯ СПІЛЬНИХ ПОЇЗДОК ПРАЦІВНИКІВ КОМПАНІЇ, З ВИКОРИСТАННЯМ NET CORE ТА ENTITY FRAMEWORK”**

Виконав: студент 5 курсу, групи ППЗ–51

спеціальності

121 Інженерія програмного забезпечення

Пелюховського С.С.

Керівник Гаманюк І.М.

Рецензент____________________

Київ - 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
О.В. Негоденко
“__” ______ 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
Пелюховському Сергію Сергійовичу
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка додатку для спільних поїздок працівників компанії, з використанням Net Core та Entity Framework»

   Керівник роботи старший викладач Гаманюк І.М.,
   (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

   затверджені наказом вищого навчального закладу від “18” лютого 2022 року №.
2. Строк подання студентом роботи 03.06.2022.
3. Вихідні дані до роботи:
   3.1. Положення побудови веб-додатку для спільних поїздок працівників компанії;
   3.2. Методи побудови веб-додатку для спільних поїздок працівників компанії;

3.3. Існуючі інструменти для створення веб-додатку для спільних поїздок працівників компанії;

3.4. Науково-технічна література;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

4.1. Огляд архітектури існуючих інструментів для створення веб-додатку для спільних поїздок працівників компанії

4.2. Розробка структури веб-додатку для спільних поїздок працівників компанії

4.3. Програмна реалізація додатку

5. Мета, об'єкт та предмет дослідження

6. Програмні засоби реалізації

7. Інструменти використані для реалізації

8. Апробація результатів дослідження

9. Висновки

10. Дата видачі завдання: 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-14.04	Виконано
2	Дослідження існуючих додатків для організації поїздок	15.04-17.04	Виконано
3	Проектування архітектури системи	18.04-25.04	Виконано

4	Розробка додатка для спільних поїздок працівників компанії, з використанням Net Core та Entity Framework	26.04-05.05	Виконано
5	Висновки, оформлення роботи	07.05-10.05	Виконано
6	Розробка демонстраційних матеріалів	11.05-15.05	Виконано
7	Попередній захист роботи	16.05-01.06	Виконано
8	Здача роботи	03.06	Виконано

Студент ______________ Пелюховський С.С.

(підпис) (прізвище та ініціали)

Керівник роботи____________ Гаманюк І.М.

(підпис) (прізвище та ініціали)

ЗМІСТ

Стор.

РЕФЕРАТ…………………………………………………………………...…8

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ………...…..9

ВСТУП…………………………………………………………………..……10

1 РОЗДІЛ АНАЛІЗ НАЯВНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ……………………………………12

1.1 Поняття веб-додаток……………………………………………………..…12

1.2 Види веб-додатків……………………………………………………….…12

1.3 Переваги та недоліки веб-додатків……………………………………….13

1.3.1 Переваги SPA додатків…………………………………...……………13

1.3.2 Недоліки SPA додатків…………………………………………………..14

1.3.3 Переваги MPA додатків…………………………………………………14

1.3.4 Недоліки MPA додатків…………………………………………………14

1.3.5 Переваги PWA додатків…………………………………………………14

1.3.5 Недоліки PWA додатків…………………………………………………14

1.4 Архітектура та принципи роботи типового web-додатку……………...15

1.5 Html………………………………………………………………………..18

1.5.1 Коротка історія………………………………………………………….18

1.5.2 Поняття та синтаксис……………………………………………………18

1.6 CSS……………………………………………………………...……………19

1.6.1 Синтаксис CSS……………………………………………………...….20

1.7 JavaScript………………………………………………………………….....21

1.8 TypeScript…………………………………………………………………….22

1.9 React…………………………………………………………………………..24

1.10 C#...27

1.11 MySQL………………………………………………………...………28

2 РОЗДІЛ МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ……………………………………..………32

2.1 Модель предметної галузі…………………………………………..……32
2.2 Модель прецедентів (варіантів використання)...36
2.3 Нефункціональні вимоги……………………………………...……38
2.3.1 Додаткова специфікація………………………………………….38
2.4 Модель проектування……………………………………………….39
2.4.1 Проектування діяльності…………………………………...…39
2.4.2 Проектування послідовності викликів методів та взаємодії об'єктів………41
2.4.3 Проектування класів та їх взаємодію……………………………...…42
2.5 Архітектура (діаграма пакетів, діаграма компонентів)...................................44
3 РОЗДІЛ РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ……...….45
3.1 Використання узагальнення (Generic)...66
3.1.1 Параметризовані класи……………………………………..………68
3.2 Використання шаблонів проектування…………………………….…...69
3.2.1 Використання шаблону DTO…………………………...……..……70
3.2.2 Використання шаблону CQRS……………………………………….72
3.2.3 Використання Mediator Pattern (посередник)...78
3.3 Використання Code First……………………………………….……..82
3.4 Використання обробки виключень……………………………….……86
4 ВИСНОВКИ…………………………………………………..………92
ПЕРЕЛІК ПОСИЛАНЬ……………………………………………....93

РЕФЕРАТ

Пояснювальна записка до дипломного проекту роботи «Розробка додатку для спільних поїздок працівників компанії, з використанням .Net та Entity Framework» викладена на 101 с., містить 25 рис., 19 літературних джерел.

Ключові слова: ДОДАТОК, РОЗРОБКА, ПОЇЗДКИ, ПРАЦІВНИКИ КОМПАНІЇ

Об'єкт дослідження: створення опитувань задоволеності співробітників

Предмет дослідження: Розробка додатку для спільних поїздок працівників компанії, з використанням .Net та Entity Framework

Мета роботи: створити додаток, здатний створювати, редагувати, відміняти спільні поїздки працівників компанії.

Методи дослідження: аналіз програмного забезпечення для створення і проектування додатку.

Отримані результати: реалізовано і додаток, який дозволяє зручно створювати поїздки для працівників компанії.

Результати дипломної роботи планується використовувати для подальшого розвитку, додання та розширеня функціоналу.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

БД – база даних;

SPA - Single Page Application;

MPA - Multi Page Application ;

PWA - Progressive Web Application;

MAC - Media Access Control;

ICMP - Internet control message protocol;

UDP - User datagram protocol;

FTP - File Transfer Protocol;

SSH - Secure Shell;

POP3 - Post Office Protocol;

SMTP - Simple Mail Transfer Protocol;

ООП – об’єктно-орієнтоване програмування;

HTML – HyperText Markup Language;

CSS – Cascade Style Sheets;

HTTP – HyperText Transfer Protocol;

JS – JavaScript;

HTTP – протокол передачі даних;

ERP - enterprise resource planning;

DNS - Domain Name System;

IP - Internet Protocol;

TCP - Transmission Control Protocol;

IDE – Інтегроване середовище розробки (Integrated Development Environment);

ВСТУП

В сучасному світі більшість людей витрачає певний час для того, щоб потрапити до роботи використовуючи на це, здебільшого, публічний транспорт. Не завжди поїздка комфортна та швидка, а від цього, в деякому сенсі, може залежати настрій на роботу протягом дня. Аналогом звичної поїздки є додаток “Uber Shuttle”, це микро бус, який курсує з точки А в точку Б за графіком, і якщо наш маршрут співпадає з маршрутом шатла, то ми можемо забронювати по цьому маршруту місце, та прибути в точку збору в конкретний час. Плюсом цього додатку є швидка та комфортна поїздка (враховуючи погодні умови ззовні), можливість відслідковувати транспорт в реальному часі за допомогою GPS. Недоліком додатку є те, що шатл курсує за графіком, і графік може не співпадати з графіком користувача, на приклад треба бути на точці Б в 9:00, а шатл може прибути на 8:55. Користувач заздалегідь не знає, чи буде шатл на даному маршруті. Шатл, як вид транспорту, коштовний, і коштує більше ніж публічний транспорт.

Тому було б прийнято рішення створити аналог додатку в розрізі компанії. Так як співробітників багато, і всі живуть в різних районах/містах, ті, у яких є машина, можуть підібрати колег по дорозі на роботу, так як точка висадки у всіх одна - це офіс.

Об’єктом дослідження є додаток для спільних поїздок працівників компанії з використанням .Net та Entity Framework для бекенду та бібліотеку “React” для фронтенду - це кросплатформений додаток, який дозволяє створити, забронювати, відмінити поїздку з можливістю обговорення деталей в чаті.

Предмет дослідження - створення додатку, з використанням інформаційних технологій.

Мета - автоматизація створення, бронювання та відслідковування поїздок через додаток.

В процесі дослідження вирішувалися наступні завдання: аналіз засобів, технологій, платформ та мов програмування для створення додатку, здійснення

моделювання та проектування додатку використовуючи UML діаграми, здійснення реалізації додатку використовуючи обрані мови програмування.

Наукова новизна дослідження полягає в моделюванні процесу додатку, використовуючи нові мови, бібліотеки та технології розробки.

Практична значущість результатів дослідження полягає в отриманні моделі процесу моделювання та створення додатку, як базової моделі загального процесу.

РОЗДІЛ 1

АНАЛІЗ НАЯВНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ

1.1 Поняття веб-додаток

Веб-додаток – це програмне забезпечення або програма, яку можна відкрити за допомогою будь-якого браузера. Зовнішній інтерфейс веб програми розробляється за допомогою таких мов програмування: HTML, CSS, Javascript, які підтримуються на будь-якому браузері (Opera, Chrome, Mozilla). У той час як для написання серверної частини (Back-end) може використовуватися будь-яка інша мова програмування або фреймворк, Python, Php, Ruby, Java, C#.

Основні переваги веб-додатків:

- веб додатки можуть застосовуватися на будь-якій операційній системі (Linux, Mac, Windows), оскільки всі вони підтримують сучасні браузери;
- у зв’язку з тим, що у веб-додатку використовується той самий код порівняно з desktop додатками їх набагато легше підтримувати;
- додаток простіше програмувати оскільки він не включає багато роботи з елементами ПК (ядро,процесор, відеокарта);
- на відміну від мобільних додатків, для веб-додатків не потрібно схвалення жодних платформ, щоб випустити свою програму;
- веб додатки це більш економний варіант для будь-якого підприємства оскільки веб-програми не вимагають підписки або покупки ліцензій, а можуть використовуватися як SaaS-сервіс, що значно дешевше

1.2 Види веб-додатків

SPA (Single Page Application) – односторінковий інтерактивний додаток. Важливо, що він не тільки знаходиться на одній сторінці, а й, подібно до

повноцінної програми, є інтерактивним. Так інформаційний веб-сайт може складатися з однієї сторінки, але, по суті, не бути SPA. В одно сторінковому веб-додатку користувач, перемикаючись між вкладками, залишається на одній сторінці. Причому підвантажуються та оновлюються лише необхідні частини контенту, що грає на користь швидкості SPA.

Приклад односторінкової програми – Gmail. Основна мова створення SPA – JavaScript. Невеликий односторінковий додаток можна зробити за допомогою бібліотеки jQuery. Однак цей варіант не найоптимальніший для великих проектів. Найкраще використовувати фреймворки Vue, React або Angular.

MPA (Multi Page Application) – традиційні багатосторінкові веб-програми. Коли користувач взаємодіє з веб-сайтом, завантажуються нові HTTP-сторінки. Тому обмін даними відбувається повільніше, ніж у SPA. Особливо, якщо є проблеми з інтернет-з’єднанням або хостингом веб сайту.

Приклади MPA – інтернет-магазини, такі як Rozetka та Amazon.

PWA (Progressive Web Application) прогресивна програма близька за своїми можливостями, функціями та якістю користувацького досвіду до нативних комп’ютерних та мобільних додатків.

1.3 Переваги та недоліки веб-додатків

1.3.1 Переваги SPA додатків

Основними перевагами SPA є велика швидкість додатку. Велика швидкість досягається завдяки тому, що контент, який не змінюється, не потрібно перезавантажувати. Також перевагою SPA є кешування. Зберігання в SPA ефективніше в роботі з кешем.

1.3.2 Недоліки SPA додатків

Користувачі можуть відключити JavaScript у своїх браузерах. У цьому випадку веб-додаток може частково або повністю не працювати.

Проблеми із безпекою - SPA є більш вразливими для атак (XSS).

1.3.3 Переваги MPA додатків

Можливість використання готових рішень, таких як OpenCart, WordPress, Joomla та інші. SEO (Search Engineering Optimization) - пошукові двигуни пристосовані для індексації багатосторінкових програм.

1.3.4 Недоліки MPA додатків

Швидкість взаємодії. MPA перезавантажують контент, коли користувач взаємодіє із ним.

Складність розробки. Потрібно окремо писати frontend та повноцінний backend.

1.3.5 Переваги PWA додатків

Працюють на будь-якому пристрої. Інтерфейс PWA підлаштовується під ширину екрана комп’ютера або будь-якого мобільного пристрою.

На домашньому екрані пристроїв можна закріплювати іконку веб-додатку.

1.3.5 Недоліки PWA додатків

На відміну від мобільних програм, PWA не представлені в магазинах AppStore та Google Play.

Споживання енергії вище, ніж у простого веб-додатку.

1.4 Архітектура та принципи роботи типового web-додатку

В основі клієнт-серверної архітектури лежать два компоненти: клієнт і сервер.

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних

функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати декілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше.

Функції, які реалізуються на сервері:

- зберігання, доступ, захист і резервне копіювання даних;
- обробка клієнтського запиту;
- відправлення результату (відповіді) клієнту.

Функції, які реалізуються на стороні клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Архітектура клієнт-сервер визначає принципи спілкування між комп'ютерами, а правила і взаємодії визначені в протоколі.

Мережевий протокол – це набір правил, за якими відбувається взаємодія між комп'ютерами в мережі.

Мережеві протоколи:

TCP/IP – набір (стек) протоколів передачі даних. TCP/IP – це позначення всієї мережі, яка працює на основі двох протоколів – TCP і IP.

TCP (Transfer Control Protocol) – протокол, який служить для встановлення надійного з'єднання між двома пристроями, передачі інформації і підтвердження її отримання.

IP (Internet Protocol) – інтернет протокол, який відповідає за правильність доставки повідомлень за певною адресою. При цьому дані розбиваються на пакети, які можуть доставлятися по-різному.

MAC (Media Access Control) – протокол, за допомогою якого відбувається ідентифікація мережевих пристроїв. Всі пристрої, підключені до інтернету, мають свою унікальну MAC адресу.

ICMP (Internet control message protocol) – протокол, який відповідає за обмін інформацією, але не використовується для передачі даних.

UDP (User datagram protocol) – протокол, який керує передачею інформації, але інформація не проходить перевірку при отриманні. Даний протокол працює швидше, ніж TCP.

HTTP (Hyper Text Transfer Protocol) – протокол передачі гіпертексту, на основі якого працюють всі сайти. Він запитує необхідні дані у віддаленій системи (веб-сторінки, файли).

FTP (File Transfer Protocol) – протокол передачі файлів зі спеціального файлового сервера на комп'ютер користувача.

SSH (Secure Shell) – протокол, який служить для забезпечення віддаленого керування системою по захищеному каналу.

POP3 (Post Office Protocol) – стандартний протокол поштового з'єднання, який відповідає за доставку пошти.

SMTP (Simple Mail Transfer Protocol) – протокол, який визначає правила для передачі пошти. Відповідає за повернення або підтвердження про доставку, оповіщення про помилку.

Існують концепції побудови системи клієнт-сервер:

1. Слабкий клієнт – потужний сервер. У такій моделі вся обробка інформації перенесена на сервер, а у клієнта права доступу суворо обмежені. Сервер відправляє відповідь, яка не вимагає додаткової обробки. Клієнт взаємодіє з користувачем: складає та відправляє запит, приймає результат і виводить інформацію на екран.

2. Сильний клієнт – концепція, в якій частина обробки інформації надається клієнтові. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп'ютер клієнта.

Система (додаток), яка заснована на клієнт-серверній взаємодії, включає три основних компоненти: уявлення даних, прикладний компонент, компонент управління ресурсами і їх зберігання.

У поточному додатку використовується дворівнева архітектура, яка складається з двох вузлів:

- сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси;
- клієнт, який представляє користувацький інтерфейс.

Принцип роботи полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо, без використання сторонніх ресурсів (рис. 1.4).

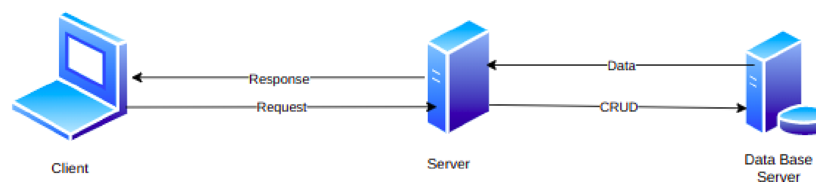


Рисунок 1.4 - Схема роботи клієнт-серверної архітектури

1.5 Html

HTML (HyperText Markup Language) — це мова опису, яка визначає структуру веб-сторінки.

1.5.1 Коротка історія

У 1990 році, як частина свого бачення Інтернету , Тім Бернерс-Лі визначив концепцію гіпертексту , яку Бернерс-Лі формалізував наступного року за допомогою розмітки, в основному на основі SGML. IETF почав формально специфікувати HTML у 1993 році, а після кількох чернеток випустив версію 2.0 у 1995 році. У 1994 році Бернерс-Лі заснував W3C для розробки Інтернету. У 1996

році W3C взяла на себе роботу з HTML і через рік опублікувала рекомендацію HTML 3.2. HTML 4.0 був випущений в 1999 році і став стандартом ISO в 2000 році.

У той час W3C майже відмовився від HTML на користь XHTM, що спонукало до заснування незалежної групи під назвою WHATWG у 2004 році. Завдяки WHATWG робота над HTML5 тривала: дві організації випустили перший проект у 2008 році та остаточний стандарт у 2014 році. Остання версія, яка здебільшого використовуються це Html5.

1.5.2 Поняття та синтаксис

HTML-документ — це відкритий текстовий документ, структурований з елементами. Елементи оточені відповідними відкриваючими та закриваючими тегами . Кожен тег починається та закінчується кутовими дужками (<>). Існує кілька порожніх або недійсних тегів, які не можуть містити текст, наприклад <img>.

Ви можете розширити теги HTML за допомогою атрибутів , які надають додаткову інформацію, що впливає на те, як браузер інтерпретує елемент (рис. 1.5.2):

Рисунок 1.5.2 - Приклад синтаксису html тегу

Файл HTML зазвичай зберігається з розширенням .htm або .html, обслуговується веб-сервером і може бути відтворений будь - яким веб-браузером.

1.6 CSS

CSS (Cascading Style Sheets) використовується для стилізації та компонування веб-сторінок — наприклад, для зміни шрифту, кольору, розміру та інтервалу вмісту, розбиття його на кілька стовпців або додавання анімації та інших декоративних функцій. Як ми вже згадували раніше, CSS — це мова для визначення того, як документи представляються користувачам — як вони оформлені, оформлені тощо.

Документ зазвичай є текстовим файлом, структурованим за допомогою мови розмітки — HTML є найпоширенішою мовою розмітки, але ви також можете зустріти інші мови розмітки, такі як SVG або XML.

Презентувати документ користувачеві означає перетворити його у форму, яку може використовувати ваша аудиторія. Браузери , такі як Firefox , Chrome або Edge, призначені для візуального представлення документів, наприклад, на екрані комп’ютера, проектора чи принтера.

CSS можна використовувати для дуже простих стилів тексту документа — наприклад, для зміни кольору та розміру заголовків і посилань. Його можна використовувати для створення макета — наприклад, для перетворення окремого стовпця тексту в макет із основною областю вмісту та бічною панеллю для пов’язаної інформації. Його навіть можна використовувати для таких ефектів, як анімація.

1.6.1 Синтаксис CSS

CSS — це мова, заснована на правилах — розробник визначає правила, вказуючи групи стилів, які слід застосовувати до певних елементів або груп елементів на вашій веб-сторінці.

Наприклад, можна зробити, щоб основний заголовок на сторінці відображався у вигляді великого червоного тексту. У наступному коді показано дуже просте правило CSS, яке дозволить досягти стилю, описаного вище (рис. 1.6.1):

```
h1 {
    color: red;
    font-size: 5em;
}
```

Рисунок 1.6.1 - Приклад CSS правила

У наведеному вище прикладі правило CSS відкривається за допомогою селектора . Це правило вибирає HTML елемент, який ми збираємося стилізувати. У цьому випадку ми стилізуємо заголовки першого рівня (<h1>). Тоді ми маємо набір фігурних дужок { }.

Всередині дужок буде одна або кілька декларацій , які мають форму пар властивостей і значень . Ми вказуємо властивість (colorу наведеному вище прикладі) перед двокрапкою, а значення властивості вказуємо після двокрапки (red у цьому прикладі).

Цей приклад містить два оголошення, одне для, color а інше для font-size. Кожна пара визначає властивість елемента(ів), який ми вибираємо (<h1> у цьому випадку), а потім значення, яке ми хотіли б надати цьому властивості.

1.7 JavaScript

JavaScript було створено для того, щоб “зробити веб-сторінки живими”.

Програми на цій мові називаються скриптами. Їх можна писати прямо на сторінці в коді HTML і вони автоматично виконуються при завантаженні сторінки.

Скрипти надаються та виконуються як простий текст. Для запуску їм не потрібна спеціальна підготовка чи компілятор.

У цьому плані JavaScript дуже відрізняється від іншої мови програмування — Java.

Коли мову JavaScript було створено, спочатку вона мала іншу назву: “LiveScript”. Але тоді була дуже популярна мова програмування Java, тому було вирішено, що позиціонування нової мови як “молодшого брата” Java допоможе у її популяризації.

Але з часом JavaScript значно виріс і став повністю незалежною мовою програмування зі своєю специфікацією ECMAScript, і зараз немає нічого спільного з Java.

Сьогодні JavaScript може виконуватися не тільки у браузері, але й на сервері, або на будь-якому пристрої, який має спеціальну програму — рушій JavaScript.

Браузер має вбудований рушій, який деколи називають “віртуальною машиною JavaScript”.

Різні рушії мають різні “кодові назви”. Наприклад:

- V8 – в Chrome, Opera та Edge.
- SpiderMonkey – в Firefox.

Є також інші кодові назви як “Chakra” для IE, “JavaScriptCore”, “Nitro” і “SquirrelFish” для Safari, та інші.

Є як мінімум три чудові особливості в JavaScript:

- Цілковита інтеграція з HTML/CSS.
- Прості речі робляться просто.
- Підтримується всіма сучасними браузерами і увімкнений усталено.

JavaScript – це єдина браузерна технологія, яка суміщає ці три речі. Це і робить його унікальним. Ось чому JavaScript найбільш поширений засіб створення браузерних інтерфейсів. До слова, JavaScript також дозволяє створювати сервери, мобільні застосунки, тощо.

1.8 TypeScript

TypeScript - це обертка над JavaScript. Хоча будь-який правильний код JavaScript є також правильним кодом TypeScript, TypeScript також має мовні функції, які не є частиною JavaScript. Найвидатнішою функцією, унікальною для TypeScript, - тією, яка дала ім'я TypeScript - це, як зазначалося, сильний набір тексту: змінна TypeScript пов'язана з типом, таким як рядок, число або логічне значення, яке повідомляє компілятору, який тип даних він може вмістити. Окрім того, TypeScript підтримує умовивід типу і включає в себе будь-який тип, що означає, що змінні не повинні чітко призначати свої типи програмістом.

TypeScript також призначений для об'єктно-орієнтованого програмування - JavaScript, не так вже й багато. Такі поняття, як успадкування та контроль доступу, які не є інтуїтивно зрозумілими в JavaScript, просто реалізувати в TypeScript. Крім того, TypeScript дозволяє вам реалізовувати інтерфейси.

Тим не менш, немає жодної функціональності, яку ви можете кодувати в TypeScript, а також кодувати в JavaScript. Це тому, що TypeScript не компілюється в загальноприйнятому розумінні - таким чином, наприклад, C ++ компілюється у двійковий виконуваний файл, який може працювати на вказаному обладнанні. Натомість компілятор TypeScript перекодовує код TypeScript у функціонально еквівалентний JavaScript. Потім отриманий JavaScript можна запустити де завгодно будь-який код JavaScript, від веб-браузера до сервера, обладнаного Node.js.

TypeScript вийшов у відкритому коді в 2012 році після розробки в Microsoft. (Програмний гігант залишається розпорядником та головним розробником проекту).

У той час Microsoft намагалася розширити Bing Maps як конкурента Google Maps, а також запропонувати веб-версії свого пакета Office - і JavaScript був основною мовою розробки для цих завдань. Але розробникам, по суті, було важко писати програми в масштабі флагманських пропозицій Microsoft за допомогою JavaScript. Тому вони розробили TypeScript, щоб полегшити створення додатків на

рівні підприємства для роботи в середовищах JavaScript. Це суть думки під слоганом для мови на офіційному веб-сайті проекту TypeScript: "JavaScript, що масштабується".

Чому TypeScript краще підходить для такого роду робіт, ніж звичайний JavaScript? Ну, ми можемо сперечатися назавжди про переваги об'єктно-орієнтованого програмування, але реальність така, що багато розробників програмного забезпечення, які працюють над великими корпоративними проектами, звикли до цього, і це допомагає при повторному використанні коду у розмірі кулі. Також не слід нехтувати мірою, в якій інструментарій може підвищити продуктивність розробника. Як зазначалося, більшість корпоративних середовищ розробки середовища підтримують фонову поступову компіляцію, яка може помітити помилки під час роботи. (Поки ваш код синтаксично правильний, він все одно буде транслюватися, але отриманий JavaScript може не працювати належним чином; вважайте перевірку помилок еквівалентом перевірки правопису.) Ці середовища розробки також можуть допомогти рефакторувати код, коли заглиблюєтесь у свій проекту.

Коротше кажучи, TypeScript використовується, коли потрібні корпоративні функції та інструменти такої мови, як Java, але потрібен код для виконання в середовищі JavaScript. Теоретично можна написати стандартний JavaScript, який компілятор TypeScript генерує самостійно, але це займе у набагато більше часу, і кодовій базі буде важче спільно зрозуміти та налагодити велику команду.

Також у TypeScript є ще одна акуратна хитрість: можна встановити компілятор для націлювання на певне середовище виконання JavaScript, браузер чи навіть мовну версію. Оскільки будь-який добре сформований код JavaScript також є кодом TypeScript, його можна, наприклад, взяти код, записаний у специфікацію ECMAScript 2015, який включав ряд нових синтаксичних функцій, та скомпілювати його в код JavaScript, який відповідав би застарілим версіям мову.

1.9 React

React — це бібліотека розробки інтерфейсу користувача на основі JavaScript. Нею керують Facebook і спільнота розробників з відкритим кодом. Хоча React є бібліотекою, а не мовою, він широко використовується у веб-розробці. Бібліотека вперше з'явилася в травні 2013 року і зараз є однією з найбільш часто використовуваних інтерфейсних бібліотек для веб-розробки.

React пропонує різні розширення для всієї архітектурної підтримки додатків, таких як Flux та React Native, за межами простого інтерфейсу.

Сьогодні популярність React перевершила популярність усіх інших фреймворків розробки інтерфейсу. Ось чому:

Легке створення динамічних програм: React полегшує створення динамічних веб-додатків, оскільки вимагає менше кодування та пропонує більше функціональних можливостей, на відміну від JavaScript, де кодування часто стає складним дуже швидко.

Покращена продуктивність: React використовує Virtual DOM, тим самим створюючи веб-додатки швидше. Віртуальний DOM порівнює попередні стани компонентів і оновлює лише ті елементи реальної DOM, які були змінені, замість того, щоб знову оновлювати всі компоненти, як це роблять звичайні веб-додатки.

Компоненти для багаторазового використання: Компоненти є будівельними блоками будь-якої програми React, і одна програма зазвичай складається з кількох компонентів. Ці компоненти мають свою логіку та елементи керування, і їх можна повторно використовувати в додатку, що, у свою чергу, значно скорочує час розробки програми.

Односпрямований потік даних: React слідує за односпрямованим потоком даних. Це означає, що при розробці програми React розробники часто вкладають дочірні компоненти в батьківські компоненти. Оскільки дані надходять в одному напрямку, стає легше налагоджувати помилки та знати, де виникає проблема в програмі в даний момент.

Невелика крива навчання: React легко навчитися, оскільки він переважно поєднує базові концепції HTML і JavaScript з деякими корисними доповненнями. Проте, як і у випадку з іншими інструментами та фреймворками, вам доведеться витратити деякий час, щоб отримати належне розуміння бібліотеки React.

Його можна використовувати як для розробки веб-додатків, так і для мобільних: ми вже знаємо, що React використовується для розробки веб-додатків, але це ще не все, що він може зробити. Існує фреймворк під назвою React Native, похідний від самого React, який дуже популярний і використовується для створення красивих мобільних додатків. Таким чином, насправді React можна використовувати для створення як веб-, так і мобільних додатків.

Спеціальні інструменти для легкого налагодження: Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження веб-додатків React швидшим і простішим.

Перераховані вище причини з лишком виправдовують популярність бібліотеки React і те, чому вона використовується великою кількістю організацій і підприємств.

React пропонує деякі видатні функції, які роблять його найбільш поширеною бібліотекою для розробки інтерфейсних програм. Ось список цих помітних особливостей.

JSX – це синтаксичне розширення JavaScript. Це термін, який використовується в React для опису того, як має виглядати інтерфейс користувача. Ви можете писати структури HTML у той самий файл, що й код JavaScript, використовуючи JSX (рис. 1.9).

```
const world = "world";
const hello = <h1>Hello, {world}</h1>;
```

Рисунок 1.9 - Приклад синтаксису JSX коду

Наведений вище код показує, як JSX реалізовано в React. Це не рядок і не HTML. Замість цього він вбудовує HTML у код JavaScript.

Віртуальний DOM — це полегшена версія Real DOM від React. Справжні маніпуляції з DOM значно повільніше, ніж віртуальні маніпуляції з DOM. Коли стан об’єкта змінюється, Virtual DOM оновлює лише цей об’єкт у справжньому DOM, а не всі (рис 1.9).

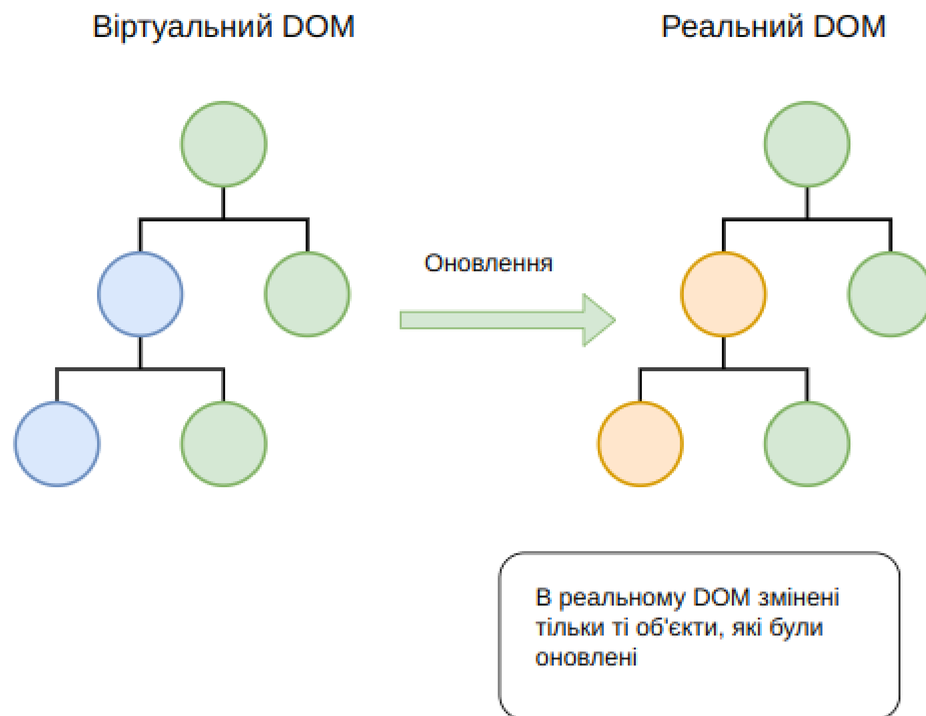


Рисунок 1.9 - Приклад роботи віртуальної DOM

1.10 C#

C# (вимовляється як «See Sharp») — сучасна, об'єктно-орієнтована та безпечна для типів мова програмування. C# дозволяє розробникам створювати багато типів безпечних і надійних програм, які працюють у .NET. C# має свої коріння в сімействі мов C і буде відразу знайомий програмістам C, C++, Java та JavaScript. Цей огляд містить огляд основних компонентів мови в C# 8 і раніше.

Якщо ви хочете вивчити мову на інтерактивних прикладах, спробуйте ознайомитися з підручниками C#.

C# — це об'єктно-орієнтована, компонентно-орієнтована мова програмування. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, роблячи C# природною мовою для створення та використання програмних компонентів. З моменту свого виникнення в C# були додані функції для підтримки нових робочих навантажень і нових методів проектування програмного забезпечення. За своєю суттю C# є об'єктно-орієнтованою мовою. Ви визначаєте типи та їх поведінку.

Кілька функцій C# допомагають створювати надійні та довговічні програми. Збір сміття автоматично відновлює пам'ять, зайняту недоступними невикористаними об'єктами. Типи, які допускають значення NULL, захищають від змінних, які не посилаються на виділені об'єкти. Обробка винятків забезпечує структурований і розширений підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Синтаксис мовного інтегрованого запиту (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела. Підтримка мови для асинхронних операцій забезпечує синтаксис для побудови розподілених систем. C# має уніфіковану систему типів. Усі типи C#, включаючи примітивні типи, такі як int і double, успадковуються від одного кореневого object типу. Усі типи мають набір загальних операцій. Цінності будь-якого типу можна зберігати, транспортувати та використовувати узгоджено. Крім того, C# підтримує як визначені користувачем типи посилань, так і типи значень. C# дозволяє динамічно розподіляти об'єкти та вбудовано зберігати полегшені структури. C# підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типів. C# надає ітератори, які дають змогу реалізаторам класів колекції визначати користувацьку поведінку для клієнтського коду.

1.11 MySQL

MySQL - це система управління базами даних, яка використовується для підтримки реляційних баз даних. Це програмне забезпечення з відкритим кодом, що підтримується корпорацією Oracle. Оскільки це система баз даних з відкритим кодом, вихідний код можна змінювати відповідно до наших потреб. MySQL - це масштабована, швидка і надійна система управління базами даних, яка може працювати на будь-якій платформі, такі як Windows, Unix, Linux і т.д., і може бути встановлена на робочому столі або будь-якій серверній машині. LAMP - це платформа для веб-розробки, що використовує Linux як операційну систему, веб-сервер apache, реляційну систему управління базами даних mysql та об'єктно-орієнтований сценарій PHP. Існує багато топ-сайтів, що використовують mysql. Крім цього, є численні корпорації, які використовують mysql як свою систему управління реляційними базами даних. Небагато прикладів включають Youtube, Facebook, Twitter і т.д. MySQL працює на моделі клієнт-сервер, сервер MySQL є основним, що обробляє всі команди.

У мережевому середовищі клієнт-сервер сервер MySQL доступний як окрема програма. Крім того, він доступний у вигляді бібліотеки, яку можна пов'язати з окремим додатком. Існує кілька корисних програм, що підтримують адміністрування бази даних MySQL. З іншого боку, клієнти mysql встановлюються на комп'ютерах у мережі. Інструкції надсилаються від клієнта mysql на сервер mysql і тоді сервер mysql діє на нього відповідно. Незважаючи на те, що mysql встановлений на одній машині, він здатний надсилати бази даних в декілька локацій, і користувачі можуть отримати доступ до одних і тих же за допомогою різних клієнтських інтерфейсів MySQL. Результати відображаються, коли ці інтерфейси передають оператори SQL серверам. Немає потреби в навчанні нових команд, оскільки функцією mysql можна керувати лише за допомогою існуючих команд SQL. Реплікація даних та розподіл таблиць також можна виконати в mysql, що дозволяє користувачам мати кращу продуктивність та більшу довговічність. Для зберігання та доступу до даних можна використовувати

декілька двигунів зберігання даних, як NDB, InnoDB тощо. MySQL написаний на C і C++ і доступний на численних платформах, включаючи Windows, Linux, Mac та інші. Ця реляційна система управління базами даних підтримує мільйони записів у базах даних та багатьох типів даних, таких як безпідписані та підписані цілі числа до 8 байт, двійкові, варчар, подвійний, char, float, час, blob, enum, текст, дата, рік, дата, часова мітка, варбінарні та просторові типи OpenGIS. Також підтримуються типи рядків з фіксованою та змінною довжиною. Mysql може підключитися до сервера mysql, використовуючи безліч протоколів, таких як TCP / IP тощо. Поряд з цим він також підтримує кілька інструментів адміністрування клієнтів та утиліт, таких як MySQL Workbench та кілька програм командного рядка.

Основні переваги використання MySQL:

- Масштабованість:

Mysql забезпечує відмінну масштабованість для управління та координації роботи з глибоко вбудованими додатками, використовуючи менший слід навіть у масивних складах, що містять величезну кількість даних. Гнучкість на вимогу - чудова особливість Mysql. Підприємства електронної комерції можуть повністю налаштувати унікальні вимоги до сервера баз даних, оскільки mysql - це рішення з відкритим кодом.

- Безпека:

Mysql - одна з найзахищеніших систем управління базами даних у всьому світі, яка використовується найпопулярнішими веб-додатками, такими як Youtube, Facebook, Twitter, WordPress тощо. Остання версія mysql забезпечує підтримку та безпеку транзакційної обробки, що забезпечує велику користь бізнес, особливо у випадку будь-якої електронної комерції, яка вимагає великої кількості грошових переказів.

- Продуктивність:

Mysql призначений для обслуговування найвибагливіших програм та забезпечення одночасно належної швидкості. Він пропонує підвищену продуктивність, надаючи унікальні кеші пам'яті та повнотекстові покажчики.

Таким чином, будь то веб-сайт електронної комерції, що виконує мільйони запитів щодня будь-якого типу операційної системи обробки операцій, mysql пропонує унікальну систему зберігання даних, що дозволяє адміністраторам налаштувати сервер mysql без будь-якого недоліку, що забезпечує високу продуктивність.

- Економічно ефективним:

Mysql пропонує підприємствам значну економію коштів, забезпечуючи належне управління та надійність, що може заощадити час на усунення несправностей, які в іншому випадку будуть витрачені на проблеми з роботою та вирішення проблем простоїв.

- Немає простоїв:

Mysql пропонує широкий спектр високодоступних рішень, що забезпечує цілодобовий час роботи за допомогою конфігурацій master / slave реплікації та спеціалізованих серверів кластерів.

- Повний контроль робочого процесу:

Mysql пропонує комплексне рішення, що забезпечує функції самокерування, що автоматизують розширення простору та адміністрування баз даних. Він пропонує мінімальний час налаштування і може бути скоріше використаний, який можна встановити в UNIX, Windows, Linux, Macintosh тощо.

- Рішення з відкритим кодом:

Mysql з відкритим кодом вирішує декілька проблем, таких як налагодження, технічне обслуговування, швидкі оновлення та покращений досвід користувача. Захищена обробка забезпечує ефективну транзакцію для обробки великих наборів даних.

- Транзакційна підтримка:

Mysql є найбільш переважним механізмом транзакційних баз даних на ринку, оскільки він забезпечує повну цілісність даних. Він пропонує необмежене блокування на рівні рядків, ізольовану, послідовну та довговічну - цілком підтримку атомних транзакцій та підтримку транзакцій у кількох версіях.

РОЗДІЛ 2

МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.

2.1 Модель предметної галузі

Останнім часом ми бачили цілу низку ідей щодо архітектури систем. До них належать:

- Гексагональна архітектура (він же Порти та адаптери) Алістером Кокберном і прийнята Стівом Фріманом і Натом Прайсом у їхній чудовій книзі «Вирощування об'єктно-орієнтованого програмного забезпечення».
- Цибуля архітектура Джеффрі Палермо
- DCI від Джеймса Коплієна та Трігве Реєнскауга.
- до нашої ери Івара Джейкобсона з його книги «Об'єктно-орієнтована програмна інженерія: підхід, орієнтований на використання»

Хоча всі ці архітектури дещо відрізняються за своїми деталями, вони дуже схожі. Усі вони мають одну мету, а саме розділення проблем. Усі вони досягають цього поділу шляхом поділу програмного забезпечення на рівні. Кожен має принаймні один рівень для бізнес-правил, а інший — для інтерфейсів.

Кожна з цих архітектур створює системи, які:

Незалежні від фреймворків. Архітектура не залежить від існування якоїсь бібліотеки програмного забезпечення, наповненого функціями. Це дозволяє вам використовувати такі фреймворки як інструменти, а не втискати вашу систему в їх обмежені обмеження.

Перевірені. Бізнес-правила можна перевірити без інтерфейсу користувача, бази даних, веб-сервера чи будь-якого іншого зовнішнього елемента.

Незалежні від UI. Інтерфейс користувача можна легко змінити, не змінюючи решту системи. Наприклад, веб-інтерфейс можна замінити інтерфейсом консолі, не змінюючи бізнес-правила.

Незалежні від бази даних. Ви можете замінити Oracle або SQL Server на Mongo, BigTable, CouchDB або щось інше. Ваші бізнес-правила не прив’язані до бази даних.

Незалежні від будь-яких зовнішніх агентств. Насправді ваші бізнес-правила просто нічого не знають про зовнішній світ.

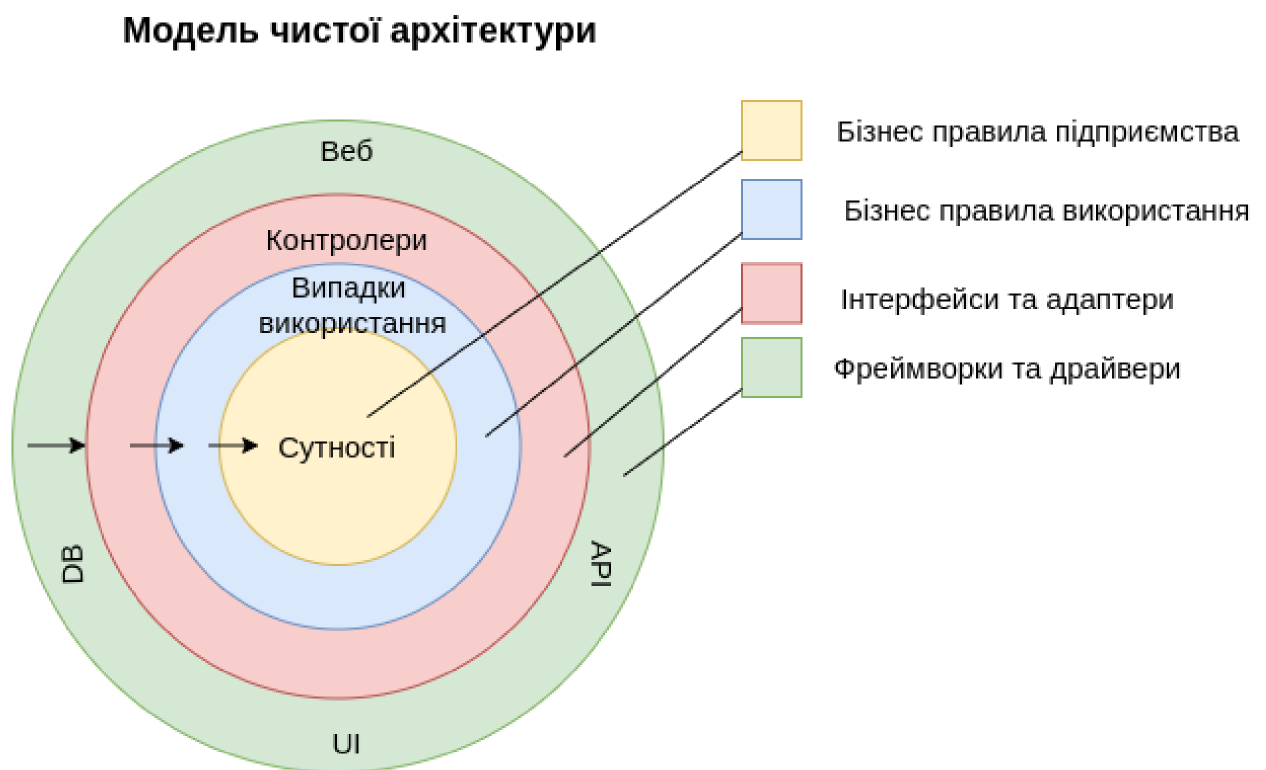


Рисунок 2.1 - Модель чистої архітектури

Правило залежності

Концентричні кола представляють різні області програмного забезпечення. Загалом, чим далі ви йдете, тим вище рівень стає програмне забезпечення. Зовнішні кола є механізмами. Внутрішні кола – політика.

Основне правило, завдяки якому ця архітектура працює, — це правило залежності. Це правило говорить, що залежності вихідного коду можуть вказувати

лише всередину. Ніщо у внутрішньому колі не може взагалі нічого знати про щось у зовнішньому колі. Зокрема, назва чогось, оголошеного у зовнішньому колі, не повинна згадуватися кодом у внутрішньому колі. Це включає в себе функції, класи. змінні або будь-який інший іменований програмний об’єкт.

Таким же чином, формати даних, які використовуються у зовнішньому колі, не повинні використовуватися внутрішнім колом, особливо якщо ці формати генеруються фреймворком у зовнішньому колі. Ми не хочемо, щоб щось у зовнішньому колі впливало на внутрішні кола.

Сутності

Суб'єкти інкапсулюють бізнес - правила підприємства. Сутність може бути об’єктом з методами, або це може бути набір структур даних і функцій. Це не має значення, оскільки сутності можуть використовуватися багатьма різними програмами на підприємстві.

Якщо у вас немає підприємства, а ви просто пишете одну програму, то ці об’єкти є бізнес - об’єктами програми. Вони інкапсулюють найзагальніші правила високого рівня. Найменше вони зміняться, коли щось змінюється зовні. Наприклад, ви не очікуєте, що на ці об’єкти вплинуть зміни в навігації сторінкою чи безпеці. Жодні операційні зміни будь-якої конкретної програми не повинні впливати на рівень сутності.

Випадки використання

Програмне забезпечення на цьому рівні містить конкретні бізнес-правила програми. Він інкапсулює та реалізує всі варіанти використання системи. Ці варіанти використання організовують потік даних до сутностей і від них і спрямовують ці об’єкти використовувати свої бізнес-правила в масштабі підприємства для досягнення цілей варіанта використання.

Ми не очікуємо, що зміни в цьому шарі вплинуть на сутності. Ми також не очікуємо, що на цей рівень вплинуть зміни зовнішніх факторів, таких як база даних, інтерфейс користувача або будь-яка із поширених фреймворків. Цей шар ізольований від подібних занепокоєнь.

Проте ми очікуємо, що зміни в роботі програми вплинуть на варіанти використання, а отже, і на програмне забезпечення на цьому рівні. Якщо деталі варіанту використання зміняться, то певний код у цьому шарі, безумовно, буде вплинутий.

Інтерфейсні адаптери

Програмне забезпечення на цьому рівні являє собою набір адаптерів, які перетворюють дані з формату, найбільш зручного для випадків використання та сутностей, у формат, найбільш зручний для деяких зовнішніх агентств, таких як база даних або Інтернет. Саме цей рівень, наприклад, повністю міститиме архітектуру MVC графічного інтерфейсу. Сюди належать доповідачі, представлення та контролери. Моделі, ймовірно, є просто структурами даних, які передаються від контролерів до варіантів використання, а потім назад від варіантів використання до презентаторів і представлень.

Подібним чином дані перетворюються на цьому рівні з форми, найбільш зручної для сутностей і випадків використання, у форму, найбільш зручну для будь-якої структури збереження. тобто База даних. Жоден код всередині цього кола не повинен знати нічого про базу даних. Якщо база даних є базою даних SQL, то весь SQL повинен бути обмежений цим рівнем, і, зокрема, частинами цього рівня, які мають відношення до бази даних.

Також на цьому рівні є будь-який інший адаптер, необхідний для перетворення даних із зовнішньої форми, наприклад зовнішньої служби, у внутрішню форму, яку використовують варіанти використання та сутності.

Фреймворки та драйвери.

Зовнішній шар, як правило, складається з фреймворків та інструментів, таких як база даних, веб-фреймворк тощо. Як правило, ми не пишемо багато коду на цьому шарі, окрім коду клею, який передається до наступного кола всередину.

На цьому шарі розташовуються всі деталі. Мережа - це деталь. База даних - це деталь. Ми зберігаємо ці речі зовні, щоб вони не могли завдати шкоди.

Тільки чотири кола?

Ні, кола схематичні. Ми можем виявити, що нам потрібно більше, ніж просто ці чотири. Немає правила, яке говорить, що ми завжди повинні мати лише ці чотири. Однак правило залежності діє завжди. Залежності вихідного коду завжди спрямовані всередину. У міру просування всередину рівень абстракції зростає. Зовнішнє коло – це бетонна деталь низького рівня. У міру просування всередину програмне забезпечення стає все більш абстрактним і інкапсулює політику вищого рівня. Внутрішнє коло є найбільш загальним.

2.2 Модель прецедентів (варіантів використання)

Діаграми варіантів використання – це один із видів діаграм UML, призначених для моделювання динамічних аспектів систем. (Інші чотири види з аналогічним призначенням – це діаграми діяльності, станів, послідовності та комунікації). Діаграми варіантів використання – основний вид діаграм при моделюванні поведінки системи, підсистеми або класу. Кожна з них показує набір варіантів використання та дійових осіб у їх взаємодії (рис. 2.2).

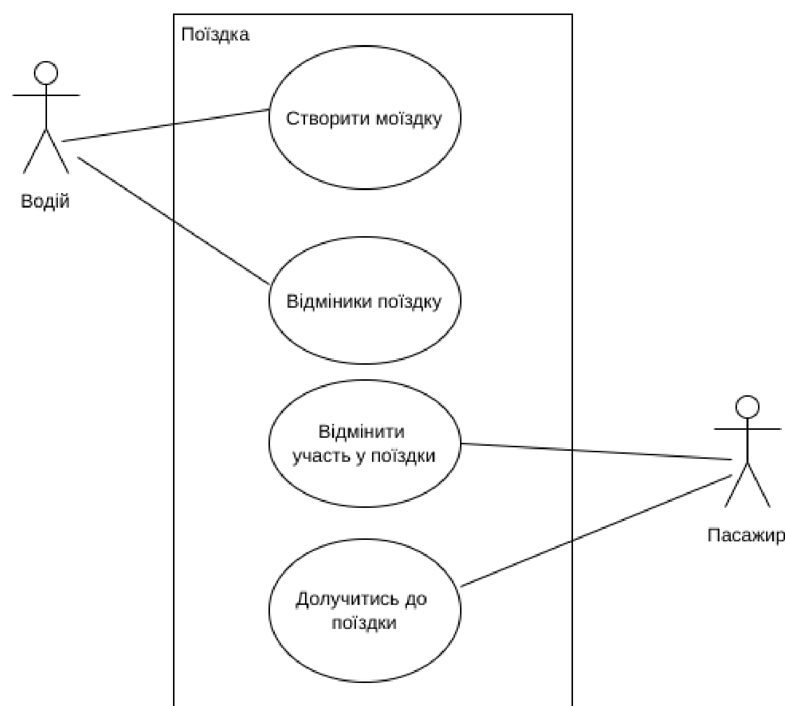


Рисунок 2.2 - Модель прецедентів модулю “поїздка”

Діаграми варіантів використання застосовуються для моделювання уявлення системи з погляду варіантів використання. Здебільшого це передбачає моделювання контексту системи, підсистеми або класу або моделювання вимог до цих елементів.

Діаграми варіантів використання важливі для візуалізації, специфікації та документування поведінки елемента.

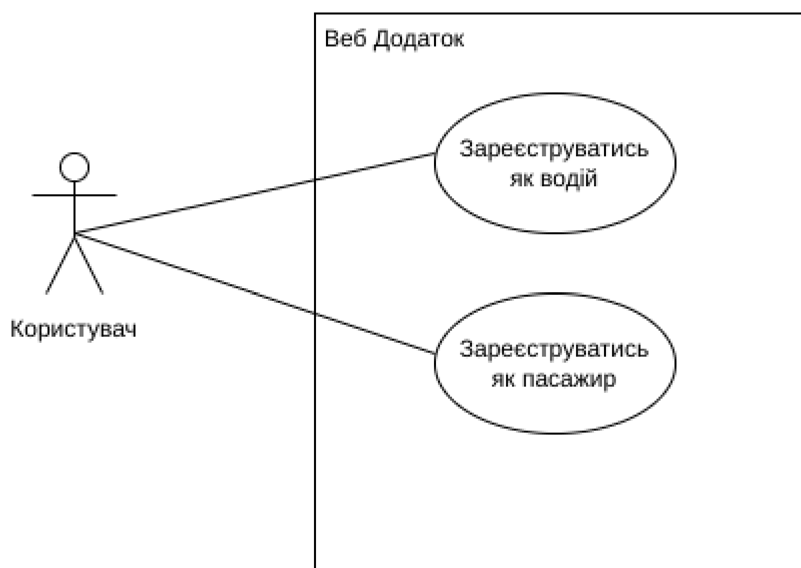


Рисунок 2.2 - Модель прецедентів модулю “реєстрація в додатку”

Вони забезпечують доступність і зрозумілість систем, підсистем та класів за рахунок зовнішнього уявлення того, як ці елементи можуть бути використані в контексті. Крім того, такі діаграми важливі для тестування працюючих систем за допомогою прямого проектування та забезпечення їх розуміння за допомогою зворотного проектування.

Рисунок 2.2 - Модель прецедентів модулю “відгук”

2.3 Нефункціональні вимоги

2.3.1 Додаткова специфікація

Веб-додаток TripSharing повинен допомагати організовувати поїздки між співробітниками до офісу компанії.

В веб-додатку повинно бути реалізовано реєстрацію користувача за ролями: водій або пасажир.

В веб-додатку повинно бути реалізована можливість створення поїздок для водія, вибір дати поїздки (не раніше наступного дня), можливість відмінити поїздку та попередити про це пасажирів.

Для пасажирів повинно бути реалізована можливість пошук поїздки, резерв місця для поїздки та можливість відміни поїздки.

В програмі повинно бути реалізовано gps навігація за водієм в реальному часі, прокладання маршруту, як для водія так і для пасажира.

В програмі повинно бути реалізовано історія минулих поїздок.

2.4 Модель проектування

2.4.1 Проектування діяльності

Діаграми діяльності – це один із п'яти видів діаграм, що застосовуються в UML для моделювання динамічних аспектів систем.

По суті, діаграма діяльності є блок-схемою, яка показує, як потік управління переходить від однієї діяльності до іншої. На відміну від традиційної блок-схеми діаграма діяльності показує паралелізм так само добре, як і розгалуження потоку управління.

Моделювання динамічних аспектів систем за допомогою діаграм діяльності переважно передбачає моделювання послідовних (а іноді і паралельних) кроків обчислювального процесу. З іншого боку, з допомогою діаграм діяльності можна моделювати потік передачі від одного кроку процесу до іншого (рис. 2.5.1).

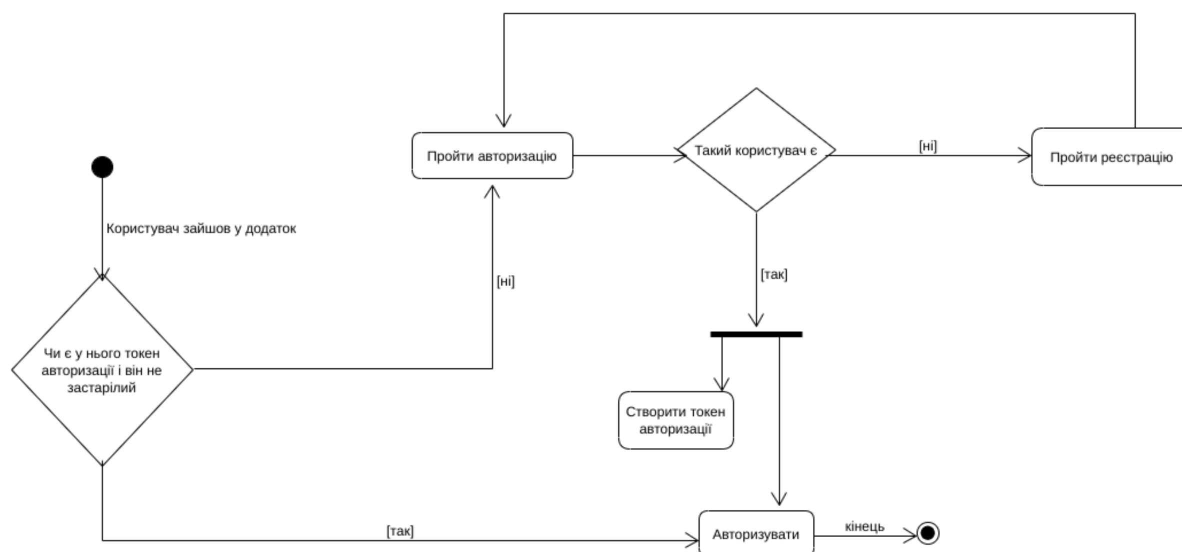


Рисунок 2.4.1 - Діаграма діяльності авторизації в веб додаток

Діаграми діяльності можуть використовуватися окремо для візуалізації, специфікації, конструювання та документування динаміки спільноти об'єктів або для моделювання потоку управління в операції. Якщо в діаграмах взаємодії акцент робиться на переходи потоку управління від одного об'єкта до іншого, то діаграми

діяльності описують переходи потоку управління від одного кроку процесів до іншого. Діяльність – це структурований опис поточної поведінки. Здійснення діяльності зрештою розкривається як виконання окремих дій, кожна з яких може змінювати стан системи або передавати повідомлення.

Діаграми діяльності важливі як для моделювання динамічних аспектів системи, а й у конструювання виконуваних систем за допомогою прямого та зворотного проектування.

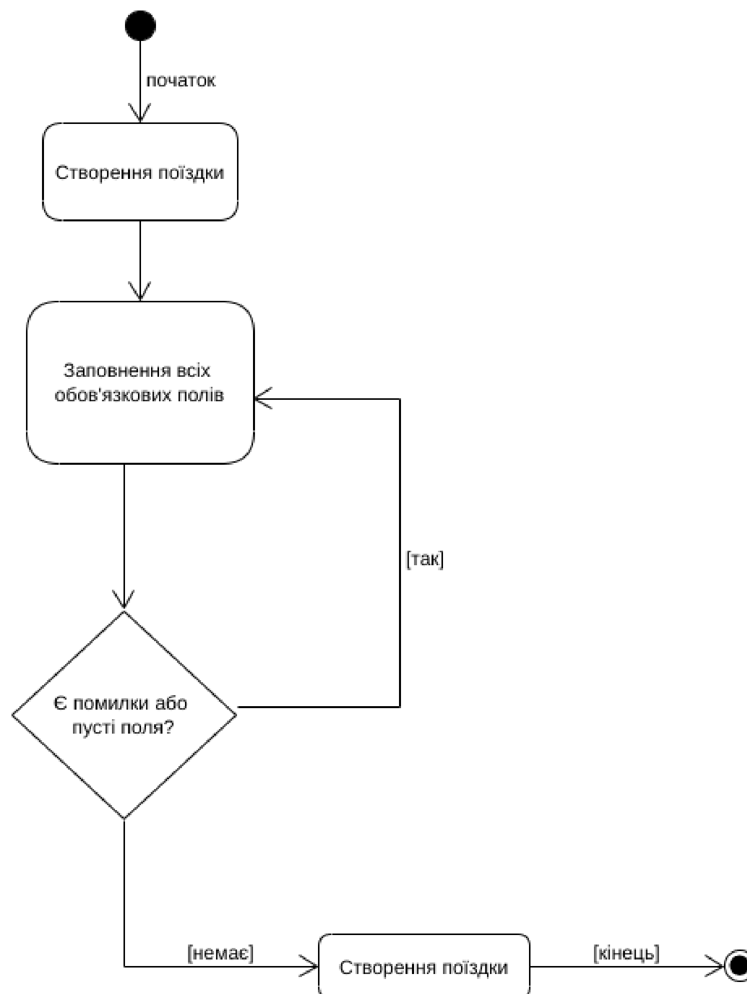


Рисунок 2.4.1 - Діаграма діяльності створення поїздки

2.4.2 Проектування послідовності викликів методів та взаємодії об'єктів

Діаграми взаємодії, у тому числі діаграми послідовності та комунікації, використовуються в UML для моделювання динамічних аспектів систем. У

загальних рисах діаграма взаємодії показує взаємодія ряду об'єктів, а також їх зв'язки та повідомлення, які можуть передаватися між ними.

Діаграми послідовності та комунікації – це діаграми взаємодії, перша з яких відображає тимчасовий порядок повідомлень, а друга – структурну організацію об'єктів, що відправляють та приймають повідомлення.

Здебільшого під моделюванням динамічних аспектів системи стосовно діаграм взаємодії мається на увазі моделювання конкретних або прототипних екземплярів класів, інтерфейсів, компонентів і вузлів поряд з повідомленнями, що передаються між ними – і все це в контексті сценарію, що ілюструє деяку поведінку. Діаграми взаємодії можуть існувати самостійно, щоб візуалізувати, специфікувати, конструювати та документувати динаміку певних спільнот об'єктів, або використовуватися для моделювання одного конкретного потоку управління в межах варіанту використання (рис. 2.4.2).

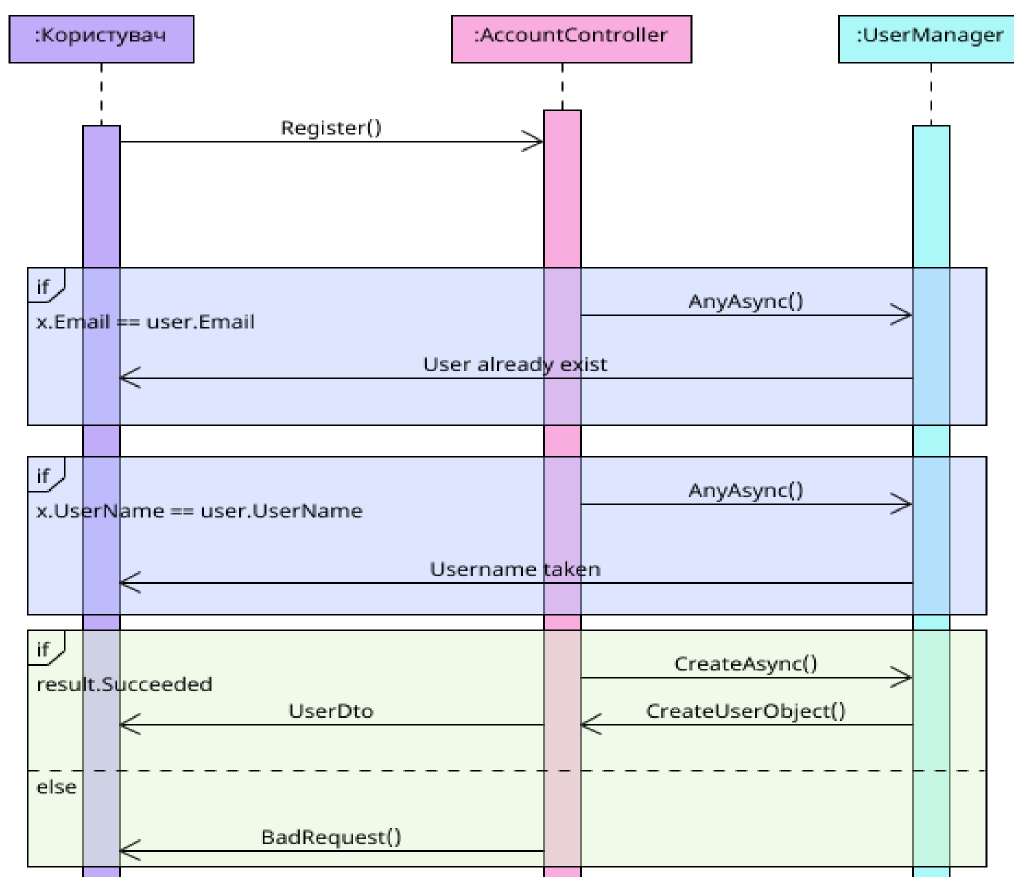


Рисунок 2.4.2 - Діаграма послідовності викликів при реєстрації користувача

Діаграми взаємодії важливі як для моделювання динамічних аспектів систем, але й конструювання виконуваних систем методом прямого і зворотного проектування.

2.4.3 Проектування класів та їх взаємодію

Діаграми класів – це найчастіше використовуваний тип діаграм, що створюються при моделюванні об'єктно-орієнтованих систем. Показують набір класів, інтерфейсів та кооперацій, а також їх зв'язки (рис. 2.4.3).

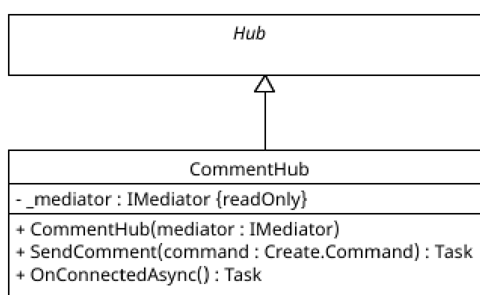


Рисунок 2.4.3 - Діаграма проектування класу CommentHub модулю коментарів

Насправді діаграми класів застосовують для моделювання статичного уявлення системи (переважно це моделювання словника системи, кооперацій чи схем). Крім того, діаграми даного типу є основою для цілої групи взаємопов'язаних діаграм – діаграм компонентів та діаграм розміщення.

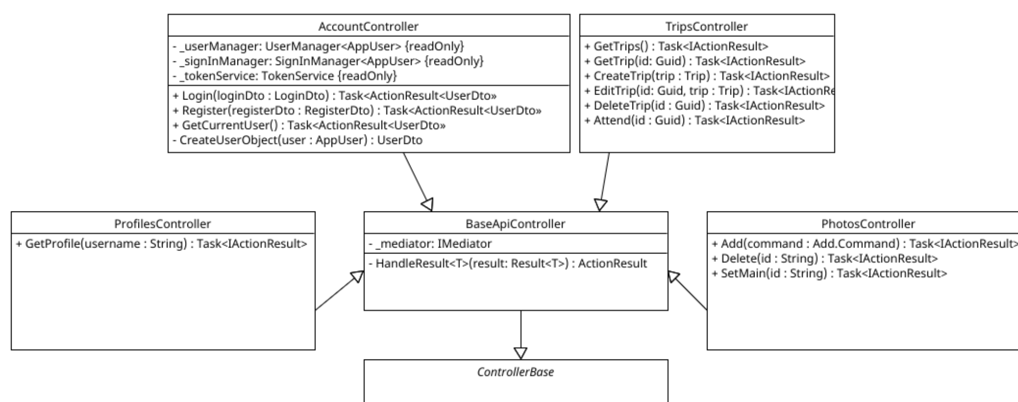


Рисунок 2.4.3 - Діаграма проектування контроллер класів

Діаграми класів важливі не тільки для візуалізації, специфікації та документування структурних моделей, але також для конструювання виконуваних систем за допомогою прямого та зворотного проектування.

2.5 Архітектура (діаграма пакетів, діаграма компонентів)

Пакет (package) є загальним механізмом організації елементів групи. Зображується як папки із закладкою.

Ім'я пакета вказано на папці, якщо вміст останньої не показано, а в іншому випадку – на закладці.

Пакет – це спосіб організації елементів моделі блоки, якими можна розпоряджатися як єдиним цілим. Можна керувати видимістю елементів пакета, так деякі буде видно користувачеві, а інші - приховані. З іншого боку, з допомогою пакетів зображуються різні уявлення архітектури системи.

Добре структурований пакет поєднує семантично близькі елементи, які мають тенденцію змінюватися разом. Такі пакети характеризуються слабкою пов'язаністю та високою узгодженістю, причому доступ до вмісту пакета суворо контролюється (рис. 2.5).

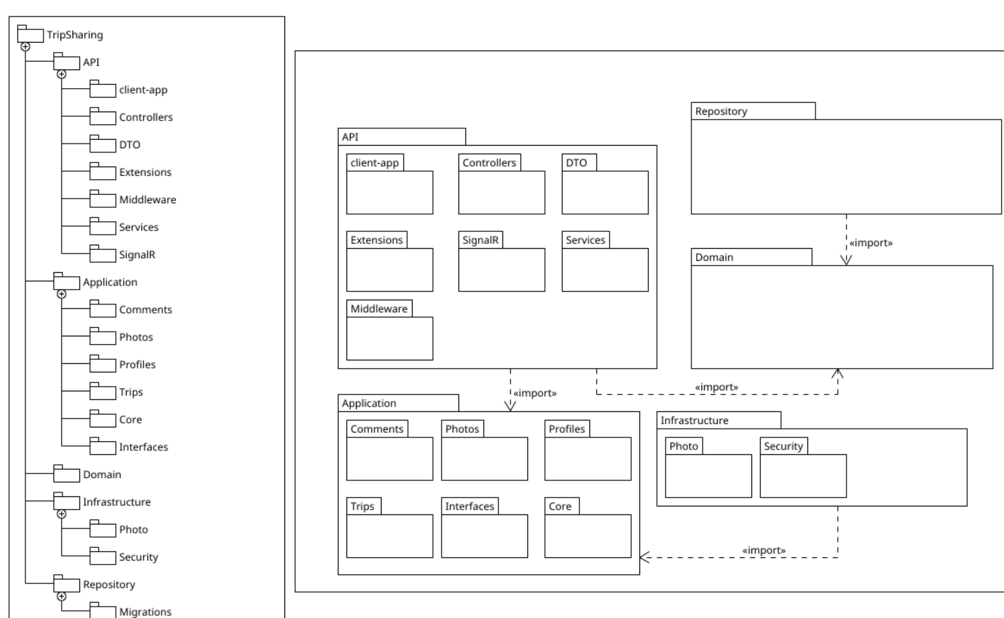


Рисунок 2.5 - Діаграма пакетів додатку

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Для реалізації проекту нам потрібно створити веб-додаток, в якому можна зареєструватися як водій або пасажир. Для пасажира ми повинні передбачити можливість пошуку необхідної поїздки та резервування або відміну поїздки. На сторінці поїздки ми повинні розробити можливість спілкування водія та пасажирів, для узгодження поїздки. Для водія ми повинні реалізувати можливість створення, редагування або відміну поїздки, а також передбачити можливість водію також забронювати поїздку.

Для обох категорій ми повинні передбачити можливість змінювати особисті дані, такі як ім'я, пошта, адреса, зображення та інше.

Для початку створення додатку попередньо було встановлено IDE Rider для роботи з кодом та Docker.7

Rider – це крос-платформна IDE для .NET розробників, заснована на платформі IntelliJ та ReSharper (рис. 3.1).

Rider підтримує .NET Framework, нову платформу .NET Core та проекти на основі Mono. IDE дозволяє розробляти десктопні програми, .NET-сервіси та бібліотеки, ігри на движку Unity, мобільні програми Xamarin, веб-програми ASP.NET і ASP.NET Core.

Rider надає більше 2200 інспекцій коду, сотні контекстних дій та рефакторингів, запозичених із ReSharper, у поєднанні з просунутою функціональністю середовищ розробки на основі платформи IntelliJ. Незважаючи на великий набір функцій, Rider – швидка та чуйна IDE.

Крім уміння запускати та налагоджувати різні програми в різних операційних системах, Rider сам по собі теж підтримує крос-платформність і працює на Windows, macOS та Linux.

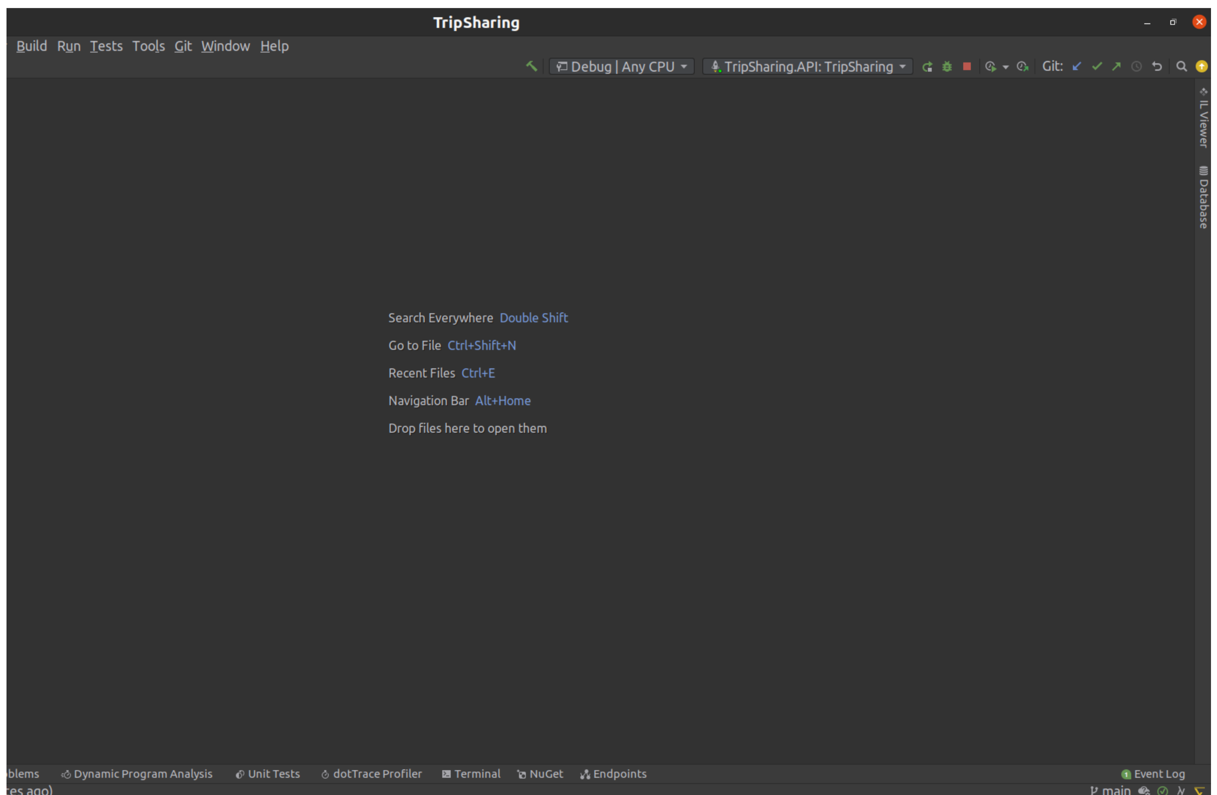


Рисунок 3.1 - Інтерфейс IDE Rider

Docker – це програмне забезпечення з відкритим кодом, найпопулярніша платформа для управління контейнерами.

Коли розробляється додаток, потрібно надати код разом з усіма його складовими, такими як бібліотеки, сервер, бази даних тощо.

Та можна опинитися в ситуації, коли додаток працює на нашому комп'ютері, але відмовляється вмикатися та працювати на пристрої іншого користувача.

Ця проблема вирішується через створення незалежності програмного забезпечення від системи.

Docker розділяє ядро операційної системи на контейнери (Docker container), що працюють, як окремі процеси.

Він вирішує безліч завдань, пов'язаних зі створенням контейнерів, розміщенням в них додатків, управлінням процесами, а також тестуванням ПЗ і його окремих компонентів.

При створенні додатку я вирішив помістити базу даних MySQL в докер контейнер щоб додатково не встановлювати її, якщо на якомусь другому

комп'ютері її не буде. Взагалі докер дуже сильно допомагає при розробці в командах, там як все необхідне програмне забезпечення находиться в докер контейнерах ми будемо впевнені в тому, що у всіх членах команди застосунок буде працювати, незалежно від того під якими вони операційними системами, та які версії мов або пакетів у них локально.

Для створення проекту в IDE Rider вибираємо меню New Solution, далі із списку можна вибрати базовий шаблон проекту, це може бути консольна програма, програма для тестування, бібліотечний клас та вибір різних фреймворків таких як .Net або .Net Core. Із поданого списку вибираємо в розділі .Net Core ASP.NET Core Web Application, в контекстному меню з’являється вікно з налаштуванням, тут ми водим назву проекту, де він буде знаходитись, додаємо підтримку Git, тип проекту - в даному випадку це Web API з підтримкою React.js, мову програмування (C# або F#) C#, фреймворк (якби на комп’ютері було встановлено декілька версій фреймворків то можна було б вибрати зі списку) так як на машині тільки один змінити не можна, Docker Support залишаємо Disabled так як будемо використовувати тільки контейнер для бази даних.

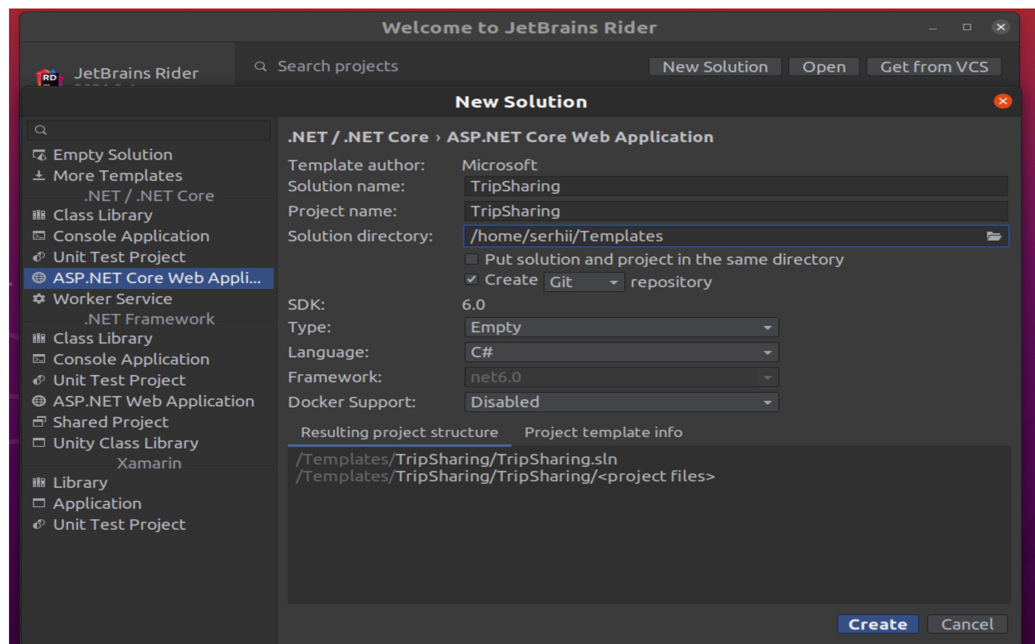


Рисунок 3.2 - Налаштування проекту в IDE Rider

Rider створює базову структуру проекту, але нас вона не влаштовує, згідно підходу чистої архітектури, тому ми прибираємо лишнє, та додаємо потрібну архітектуру.

Точкою входу для проекту буде пакет API, в ньому буде головний клас Program, додаткові сервіси, мідлвейри, ДТО, файли з налаштуванням, докер файл, контролери та фронтенд (рис. 3.3).

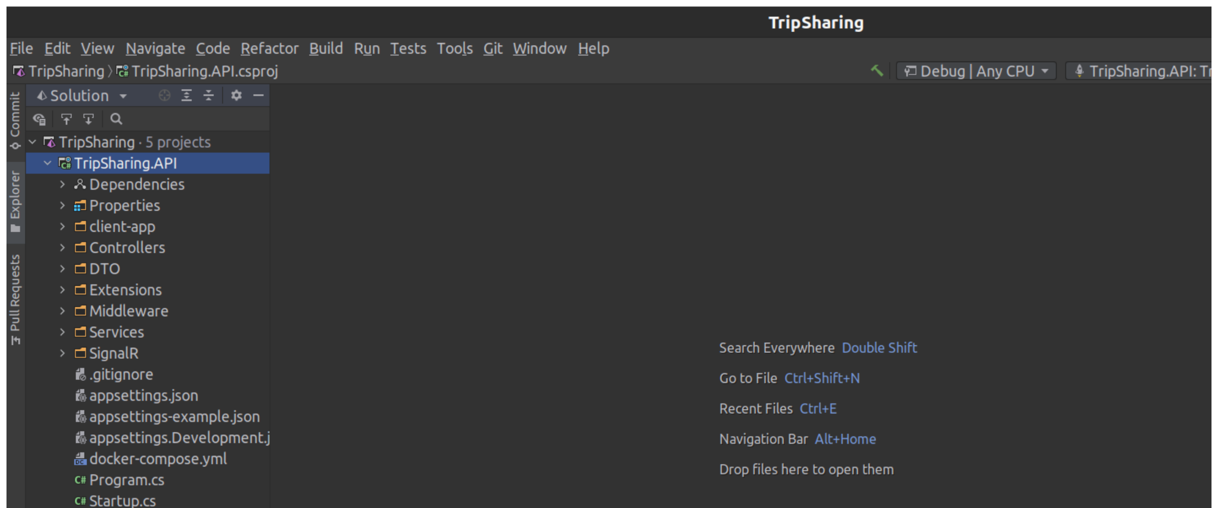


Рисунок 3.3 - Структура пакетів та класів в проекті

Тепер пройдемось по кожньому модулю окремо:

- API
- client-app - тут ми зберігаємо весь візуальний інтерфейс проекту (UI)
- Controllers - тут містяться всі наші контролери які приймають і обробляють запит від користувача
- DTO - класи дто які ми приобразовуємо при отриманні даних від користувачів
- Extensions - допоміжні класи з налаштуванням
- Middleware - класи, через які проходять запити, тут реалізовано клас, який займається обробленням помилок
- Services - сервісні класи, в даному випадку тільки один сервіс для створення токену при реєстрації або авторизації користувача
- SignalR - пакет з налаштуванням SignalR

- Application
- Comments, Photos, Profiles, Trips - це модулі, які відповідають за випадки використання в застосунку
- Core - класи з бізнес логікою
- Interfaces - інтерфейси аксесори
- Domain
- доменні класи в яких описується структура нашої бази даних, кожен клас відповідає таблиці, кожна властивість або поле відповідають за стовпчик в таблиці
- Infrastructure
- додаткові модулі для зовнішньої роботи, такі як загрузка фотографій в хмарний сервіс
- Repository
- тут знаходяться класи, які безпосередньо створюють базу даних застосунку, допоміжні класи для тестових даних,
- Migrations - модуль, який створює entity framework при створенні або редагуванні структури бази даних

Структура проекту готова, далі ми приступаємо до створення докер файлу з налаштування нашого сервісу з базою даних та інтерфейсом доступу до неї через phpmyadmin (рис. 3.4).

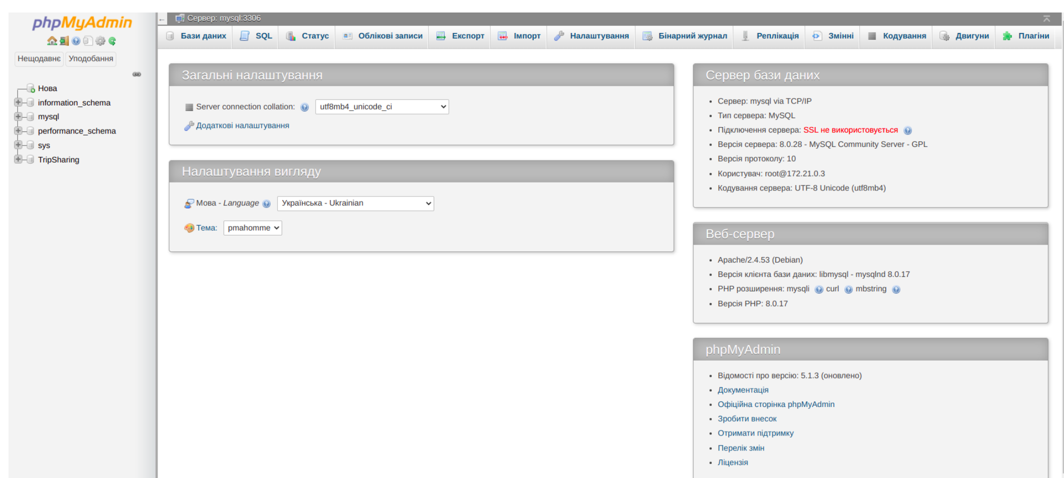


Рисунок 3.4 - Інтерфейс контейнера phpmyadmin

Файл з налаштуванням знаходиться в теці API і виглядає так

```yaml
version: "3"

services:
 mysql:
   image: mysql:8.0
   command: --default-authentication-plugin=mysql_native_password
   restart: unless-stopped
   volumes:
     - db_data:/var/lib/mysql
   ports:
     - 3307:3306
   environment:
     MYSQL_DATABASE: TripSharing
     MYSQL_USER: dev
     MYSQL_PASSWORD: dev
     MYSQL_ROOT_PASSWORD: root
   networks:
     - tripsharing

 phpmyadmin:
   image: phpmyadmin
   environment:
     - PMA_ARBITRARY=1
     - UPLOAD_LIMIT=100M
     - PMA_HOST=mysql
     - PMA_PORT=3306
     - PMA_USER=root
     - PMA_PASSWORD=root
```

```
    restart: unless-stopped
    ports:
      - 8080:80
    depends_on:
      - mysql
    networks:
      - tripsharing

networks:
 tripsharing:
    driver: bridge

volumes:
 db_data:
```

В цьому файлі прописується версія докера, контейнери, версії цих контейнерів (на приклад mysql:8.0) команди при запуску, порти та мережа для з'єднання контейнерів між собою. Для нас тут саме головне це порт для контейнера mysql, по ньому ми будем підключати наш додаток.

В додатку є файли налаштування під назвою appsettings.json, appsettings.Development.json - це налаштування в залежності від середовища, appsettings.Development.json ми використовуємо для локального середовища, appsettings.json - для середовища продакшена. В файлі appsettings.Development.json прописуєм наступні налаштування

```
{
 "Logging": {
   "LogLevel": {
     "Default": "Information",
     "Microsoft": "Information",
```

```
    "Microsoft.Hosting.Lifetime": "Information"
  }
},
"ConnectionStrings": {
    "DefaultConnection": "server=localhost; database=TripSharing; port=3307; user=root; password=root;"
},
"Origins": "http://localhost:5001",
"TokenKey": "temporary token key"
}
```

В налаштуванні ми вказуємо рівень логіювання для консолі, стрічку для з'єднання з базою даних, ключ токен так адресу додатку.

.NET підтримує шаблон проектування програмного забезпечення для впровадження залежностей (DI), який є технікою досягнення інверсії керування (IoC) між класами та їх залежностями. Ін'єкція залежностей у .NET є вбудованою частиною фреймворка разом із конфігурацією, веденням журналу та шаблоном параметрів. В класі Startup описано всі сервіси, які використовує додаток, але щоб він не був дуже “товстий”, ми розбили його на декілька класів екстеншенів.

```
using FluentValidation.AspNetCore;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.AspNetCore.SpaServices.ReactDevelopmentServer;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
using TripSharing.Application.Trips;
using TripSharing.Extensions;
using TripSharing.Middleware;
using TripSharing.SignalR;

namespace TripSharing
{
  public class Startup
  {
    private readonly IConfiguration _config;

    public Startup(IConfiguration config)
    {
      _config = config;
    }

     // This method gets called by the runtime. Use this method to add services to
the container.
    public void ConfigureServices(IServiceCollection services)
    {
      services.AddControllersWithViews(opt =>
        {
          var policy = new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build();

          opt.Filters.Add(new AuthorizeFilter(policy));
        })
        .AddFluentValidation(config =>
      {
```

```
            config.RegisterValidatorsFromAssemblyContaining<Create>();
        });
        services.AddAppServices(_config);
        services.AddIdentityServices(_config);
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseMiddleware<ExceptionMiddleware>();

        if (env.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }

        //app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseSpaStaticFiles();

        app.UseRouting();
```

```
            app.UseCors("CorsPolicy");

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller}/{action=Index}/{id?}");
                endpoints.MapHub<CommentHub>("/comment");
            });

            app.UseSpa(spa =>
            {
                spa.Options.SourcePath = "client-app";

                if (env.IsDevelopment())
                {
                    spa.UseReactDevelopmentServer(npmScript: "start");
                }
            });
        }
    }
}

using MediatR;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.DependencyInjection;
using TripSharing.Application.Core;
using TripSharing.Application.Interfaces;
using TripSharing.Application.Trips;
using TripSharing.Infrastructure.Photo;
using TripSharing.Infrastructure.Security;
using TripSharing.Repository;

namespace TripSharing.Extensions
{
  public static class ApplicationServiceExtensions
  {
        public static IServiceCollection AddAppServices(this IServiceCollection
services, IConfiguration config)
     {
        // In production, the React files will be served from this directory
          services.AddSpaStaticFiles(configuration => { configuration.RootPath =
"client-app/build"; });
        services.AddSwaggerGen();
        services.AddDbContext<DataContext>(options =>
        {

options.UseMySQL(config.GetConnectionString("DefaultConnection"));
        });
        services.AddCors(options =>
        {
           options.AddPolicy("CorsPolicy", policy =>
           {
              policy
                 .AllowAnyMethod()
```

```
                .AllowAnyHeader()
                .AllowCredentials()
                .WithOrigins(config.GetValue<string>("Origins"));
            });
        });
        services.AddMediatR(typeof(List.Handler).Assembly);
        services.AddAutoMapper(typeof(MappingProfiles).Assembly);
        services.AddScoped<IUserAccessor, UserAccessor>();
        services.AddScoped<IPhotoAccessor, PhotoAccessor>();

services.Configure<CloudinarySettings>(config.GetSection("Cloudinary"));
        services.AddSignalR();

        return services;
    }
  }
}

using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using TripSharing.Domain;
using TripSharing.Infrastructure.Security;
using TripSharing.Repository;
using TripSharing.Services;
```

```
namespace TripSharing.Extensions
{
  public static class IdentityServiceExtensions
  {
      public static IServiceCollection AddIdentityServices(this IServiceCollection
services,
        IConfiguration config)
      {
        services.AddIdentityCore<AppUser>(options =>
          {
            options.Password.RequireNonAlphanumeric = false;
          })
          .AddEntityFrameworkStores<DataContext>()
          .AddSignInManager<SignInManager<AppUser>>();

                                                    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]));

        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
          .AddJwtBearer(opt =>
          {
            opt.TokenValidationParameters = new TokenValidationParameters
            {
              ValidateIssuerSigningKey = true,
              IssuerSigningKey = key,
              ValidateIssuer = false,
              ValidateAudience = false
            };
            opt.Events = new JwtBearerEvents
```

```
                {
                    OnMessageReceived = context =>
                    {
                        var accessToken = context.Request.Query["access_token"];
                        var path = context.HttpContext.Request.Path;
                        if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/comment"))
                        {
                            context.Token = accessToken;
                        }

                        return Task.CompletedTask;
                    }
                };
            });
            services.AddAuthorization(opt =>
            {
                opt.AddPolicy("IsTripDriver", policy =>
                {
                    policy.Requirements.Add(new IsDriverRequirement());
                });
            });
            services.AddTransient<IAuthorizationHandler,
IsDriverRequirementHandler>();
            services.AddScoped<TokenService>();

            return services;
        }
    }
}
```

Налаштування проекту готове, тому ми можемо приступати до написання перших модулів.

Тепер ми можемо створити класи сутності для нашої бази даних (рис. 3.5).

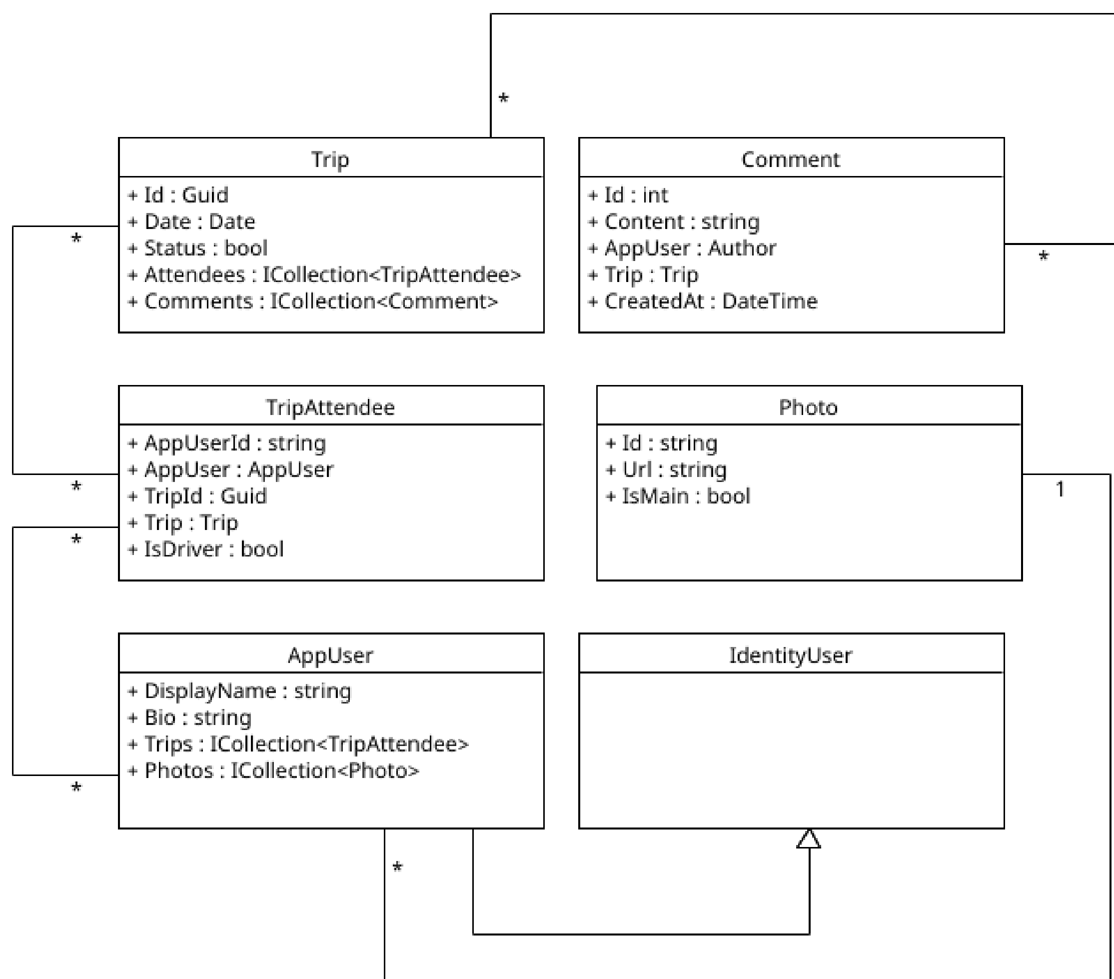


Рисунок 3.5 - Діаграма зв’язків класів сутностей

У всіх додатках мають бути CRUD операції (create, read, update, delete) тому ми почнемо з них. Створимо клас Create для модулю Trips згідно з CQRS патерном - згідно нього у нас є два типи операції command (команда) зробити/створити, та query (запит) це щось получити/знайти. Клас Create відноситься до команди створення, тому він буде виглядати таким чином

```
using System.Threading;
```

```
using System.Threading.Tasks;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using TripSharing.Application.Core;
using TripSharing.Application.Interfaces;
using TripSharing.Domain;
using TripSharing.Repository;

namespace TripSharing.Application.Trips
{
  public class Create
  {
    public class Command : IRequest<Result<Unit>>
    {
      public Trip Trip { get; set; }
    }

    public class CommandValidator : AbstractValidator<Command>
    {
      public CommandValidator()
      {
        RuleFor(x => x.Trip).SetValidator(new TripValidator());
      }
    }

    public class Handler : IRequestHandler<Command, Result<Unit>>
    {
      private readonly DataContext _context;
      private readonly IUserAccessor _userAccessor;
```

```
public Handler(DataContext context, IUserAccessor userAccessor)
{
    _context = context;
    _userAccessor = userAccessor;
}

        public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
{
    var user = await _context.Users.FirstOrDefaultAsync(x =>
        x.UserName == _userAccessor.GetUsername());

    // @todo if user is driver, he could create trip
    var attendee = new TripAttendee
    {
        AppUser = user,
        Trip = request.Trip,
        IsDriver = true
    };

    request.Trip.Attendees.Add(attendee);

    _context.Trips.Add(request.Trip);

    var result = await _context.SaveChangesAsync() > 0;

    return !result
        ? Result<Unit>.Failure("Failed to create trip")
        : Result<Unit>.Success(Unit.Value);
```

```
        }
      }
    }
  }
```

Вкладений клас CommandValidator нам потрібен, щоб провалідувати всі поля, які приходять від користувача, і, на приклад, якщо якесь поле не заповнено або заповнено не коректно повідомити його про це.

У нас є сутність, є випадок використання, тепер ми можемо створити контролер, який буде реагувати на запити користувача, і виконувати відповідні дії. Контролер який приймає запити виглядає так:

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TripSharing.Application.Trips;
using TripSharing.Domain;

namespace TripSharing.Controllers
{
  public class TripsController : BaseApiController
  {
    [HttpGet]
    public async Task<IActionResult> GetTrips()
    {
      return HandleResult(await Mediator.Send(new List.Query()));
    }

    [HttpGet("{id}")]
```

```
public async Task<IActionResult> GetTrip(Guid id)
{
   return HandleResult(await Mediator.Send(new Details.Query{Id = id}));
}

[HttpPost]
public async Task<IActionResult> CreateTrip(Trip trip)
{
     return HandleResult(await Mediator.Send(new Create.Command {Trip =
trip}));
}

[Authorize(Policy = "IsTripDriver")]
[HttpPut("{id}")]
public async Task<IActionResult> EditTrip(Guid id, Trip trip)
{
   trip.Id = id;
      return HandleResult(await Mediator.Send(new Edit.Command {Trip =
trip}));
}

[Authorize(Policy = "IsTripDriver")]
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTrip(Guid id)
{
      return HandleResult(await Mediator.Send(new Delete.Command {Id =
id}));
}

[HttpPost("{id}/attend")]
```

```
public async Task<IActionResult> Attend(Guid id)
{
return HandleResult(await Mediator.Send(new UpdateAttendance.Command {Id = id}));
}
}
}
```

Клас TripsController наслідується від BaseApiController в якому ми описуємо дублюючий код, який використовуємо у всіх контролерах, таким чином ми використали успадкування - один із китів ООП парадигми та принтримались принципу DRY (don't repeat yourself - не повторюйся). Отже функціонал створення поїздок готовий. Так як ми притримуємося патерну CQRS створення всіх наступних класів виглядає однаково.

3.1 Використання узагальнення (Generic)

У перших версіях C# .NET при створенні програм дані описувались строго визначеними типами такими як int, float, double тощо. На базі цих типів можна було створювати базові елементи програми – масиви, структури, класи тощо.

Необхідність створення однакових елементів для різних типів породжувала надлишковість програмного коду. Так, якщо у програмі потрібно було оголосити три масиви даних типів int, float або char, то кожен масив мав бути оголошений окремо. Також окремо потрібно було розробляти коди обробки цих масивів. Якщо кожен масив повинен бути оброблений в однаковий спосіб (наприклад, пошук елементу), то код обробки кожного масиву повторювався. Відмінність була тільки в описі типу даних масиву (int, float, char тощо). У результаті виникла потреба у створенні механізму, який дозволив би обробляти кожен масив єдиним уніфікованим способом незалежно від його базового типу даних.

Іншою причиною стало покращення типової безпеки у випадку використання неузагальнених колекцій. У цьому виді колекцій (CollectionBase, ArrayList, HashTable, SortedList та інші) дані зберігаються у вигляді типу object, який є базовим для всіх типів. При витягуванні даних з колекції потрібно явно вказувати приведення типу (інакше компілятор видасть помилку на етапі компіляції). Якщо при явному приведенні типу вказати не той тип, то помилка виникне на етапі виконання. Для уникнення цього потрібно додавати додаткові блоки перевірок з використанням операторів try-catch, is, as. Такі перевірки додадуть у програму зайвий код, що ускладнить її.

Враховуючи вищенаведені недоліки, починаючи з версії .NET 2.0 у мову C# було введено підтримку так званих узагальнень.

Узагальнення – засіб мови C#, що дозволяє створювати програмний код, що містить єдине (типізоване) рішення задачі для різних типів, з його подальшим застосуванням для будь-якого конкретного типу (int, float, char тощо).

Термін узагальнення асоціюється з поняттям параметризований тип. З допомогою параметризованих типів можна створювати деякі елементи мови (класи, структури тощо) таким чином, що в них оброблювані дані представлені параметром типу.

Використання узагальнень дає наступні переваги:

- спрощення програмного коду. Використання узагальнень дозволяє реалізовувати алгоритм для будь-якого типу елементів. Не потрібно створювати різні варіанти алгоритму для різних типів (int, float, string тощо);
- забезпечення типової безпеки. В узагальнених елементах (класах, методах, колекціях тощо) поміщуються тільки об'єкти визначеного типу, що вказується при їх оголошенні;
- в узагальненнях виключена необхідність явного приведення типу при перетворення об'єкту чи іншого типу оброблюваних даних;
- підвищення продуктивності. При використанні узагальнень структурні типи передаються за значенням. При цьому не виконується упакування (boxing) та розпакування (unboxing), яке уповільнює виконання програми.

Як мовний засіб, узагальнення можуть бути застосовані до:

- класів;
- структур;
- інтерфейсів;
- методів;
- делегатів.

Додатково виділяють застосування узагальнень до колекцій. На базі цього механізму створено ряд узагальнених колекцій: Collection<T>, List<T>, Dictionary<TKey, TValue>, SortedList<TKey, TValue>, Stack<T>, Queue<T>, LinkedList<T>.

Якщо елемент мови (клас, структура, інтерфейс, метод, делегат) оперує параметризованим типом даних (узагальненням), то цей елемент називається узагальнений (узагальнений клас, узагальнений метод тощо).

3.1.1 Параметризовані класи

```
namespace TripSharing.Application.Core
{
  public class Result<T>
  {
     public bool IsSuccess { get; set; }
     public T Value { get; set; }
     public string Error { get; set; }

        public static Result<T> Success(T value) => new Result<T> {IsSuccess = true, Value = value};
       public static Result<T> Failure(string error) => new Result<T> {IsSuccess = false, Error = error};
  }
```

}

Клас Result являється параметризованим, використовується нашипи випадками використання, та в залежності від сценарію приймає тип, заздалегідь який ми не знаємо, і це робить його дуже гнучким, так як від нього нам треба два статуси: успіх або помилка.

3.2 Використання шаблонів проектування

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.

Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Описи патернів зазвичай дуже формальні й найчастіше складаються з таких пунктів:

- проблема, яку вирішує патерн;
- мотивація щодо вирішення проблеми способом, який пропонує патерн;
- структура класів, складових рішення;
- приклад однією з мов програмування;
- особливості реалізації в різних контекстах;
- зв’язки з іншими патернами.

Ми можемо цілком успішно працювати, не знаючи жодного патерна. Більше того, ми могли вже не раз реалізувати який-небудь з патернів, навіть не підозрюючи про це.

Але якраз свідоме володіння інструментом відрізняє професіонала від аматора. Ми можемо забити цвях молотком, а можемо й дрилем, якщо дуже

сильно постараємось. Але професіонал знає, що головна функція дриля зовсім не в цьому. Отже, навіщо ж знати патерни?

- Перевірені рішення. Витрачаємо менше часу, використовуючи готові рішення, замість повторного винаходу велосипеда. До деяких рішень могли б дійти й самотужки, але багато які з них стануть для нас відкриттям.
- Стандартизація коду. Робимо менше прорахунків при проектуванні, використовуючи типові уніфіковані рішення, оскільки всі приховані в них проблеми вже давно знайдено.
- Загальний словник програмістів. Вимовляємо назву патерна, замість того, щоб годину пояснювати іншим програмістам, який крутий дизайн придумали і які класи для цього потрібні.

3.2.1 Використання шаблону DTO

DTO або об’єкти передачі даних — це об’єкти, які переносять дані між процесами, щоб зменшити кількість викликів методів. Вперше шаблон був введений Мартіном Фаулером у своїй книзі EAA (Шаблони архітектури додатків підприємства).

Фаулер пояснив, що головна мета шаблону — скоротити перехід до сервера за рахунок групування декількох параметрів за один виклик. Це зменшує витрати мережі під час таких віддалених операцій.

Іншою перевагою є інкапсуляція логіки серіалізації (механізм, який перекладає структуру об’єкта та дані в певний формат, який можна зберігати та передавати). Він забезпечує єдину точку зміни нюансів серіалізації. Він також відокремлює моделі домену від рівня презентації, дозволяючи обом змінюватися незалежно.

Об’єкти DTO це плоскі структури даних, які не містять бізнес-логіки. Вони містять лише сховище, засоби доступу та методи, пов’язані з серіалізацією або розбором.

Дані відображаються з моделей домену в DTO, як правило, через компонент відображення в шарі презентації або фасаду (рис. 3.2.1).

Зображення нижче ілюструє взаємодію між компонентами:

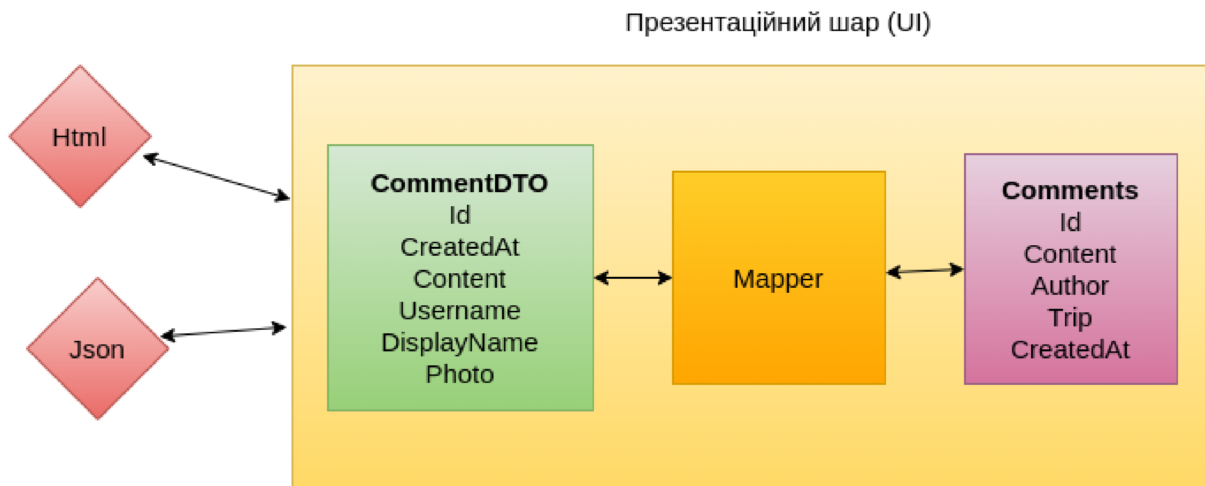


Рисунок 3.2.1 - Приклад використання DTO

DTO стають у нагоді в системах з віддаленими викликами, оскільки допомагають зменшити їх кількість.

DTO також допомагають, коли модель домену складається з багатьох різних об’єктів, а моделі презентації потрібні всі їхні дані одночасно, або вони навіть можуть зменшити зворотний зв’язок між клієнтом і сервером.

За допомогою DTO ми можемо створювати різні уявлення з наших моделей домену, що дозволяє нам створювати інші уявлення того самого домену, але оптимізуючи їх відповідно до потреб клієнтів, не впливаючи на дизайн нашого домену. Така гнучкість є потужним інструментом для вирішення складних проблем.

```
using System;

namespace TripSharing.Application.Comments;

public class CommentDto
{
  public int Id { get; set; }
  public DateTime CreatedAt { get; set; }
  public string Content { get; set; }
```

```
  public string Username { get; set; }
  public string DisplayName { get; set; }
  public string Photo { get; set; }
}

using System;

namespace TripSharing.Domain;

public class Comment
{
  public int Id { get; set; }
  public string Content { get; set; }
  public AppUser Author { get; set; }
  public Trip Trip { get; set; }
  public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
}
```

3.2.2 Використання шаблону CQRS

CQRS (Command and Query Responsibility Segregation) шаблон, який розділяє операції читання та оновлення для сховища даних. Реалізація CQRS у програмі може максимально підвищити її продуктивність, масштабованість та безпеку. Гнучкість, створена в результаті переходу на CQRS, дозволяє системі краще розвиватися з часом і запобігає тому, щоб команди оновлення викликали конфлікти злиття на рівні домену.

У традиційних архітектурах для запитів та оновлення бази даних використовується та сама модель даних. Це просто і добре працює для основних операцій CRUD. Однак у більш складних програмах цей підхід може стати громіздким. Наприклад, на стороні читання програма може виконувати багато різних запитів, повертаючи об’єкти передачі даних (DTO) різної форми.

Відображення об'єктів може ускладнитися. З боку запису модель може реалізувати складну перевірку та бізнес-логіку. В результаті ви можете отримати занадто складну модель, яка робить занадто багато.

CQRS розділяє читання і запис на різні моделі, використовуючи команди для оновлення даних і запити для читання даних.

- Команди повинні бути засновані на завданнях, а не на даних. ("Забронювати готельний номер", а не "встановити ReservationStatus на Reserved").
- Команди можуть розміщуватися в черзі для асинхронної обробки, а не оброблятися синхронно.
- Запити ніколи не змінюють базу даних. Запит повертає DTO, який не інкапсулює будь-які знання домену.

Потім моделі можна відокремити, як показано на наступній діаграмі, хоча це не є абсолютною вимогою (рис. 3.2.2).

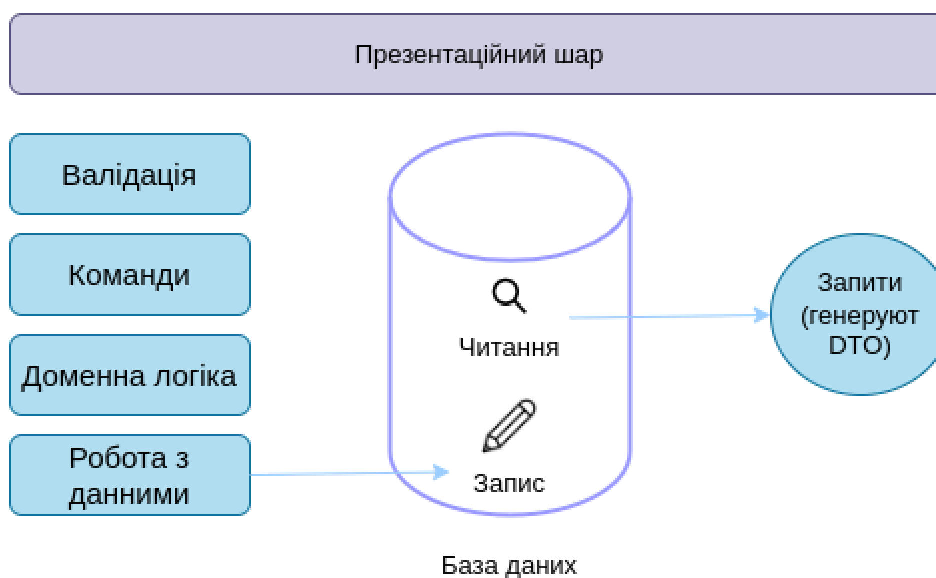


Рисунок 3.2.2 - Приклад взаємодії різних шарів згідно шаблону CQRS

Наявність окремих моделей запитів і оновлення спрощує розробку та реалізацію. Однак одним недоліком є те, що код CQRS не може бути автоматично

згенерований зі схеми бази даних за допомогою механізмів риштувань, таких як інструменти O/RM.

Для більшої ізоляції ми можемо фізично відокремити дані зчитування від даних запису. У цьому випадку читана база даних може використовувати свою власну схему даних, оптимізовану для запитів. Наприклад, він може зберігати матеріалізоване представлення даних, щоб уникнути складних з'єднань або складних зіставлення O/RM. Він навіть може використовувати інший тип сховища даних. Наприклад, база даних запису може бути реляційною, тоді як база даних читання є базою даних документів.

Якщо використовуються окремі бази даних для читання та запису, їх необхідно синхронізувати. Зазвичай це досягається завдяки тому, що модель запису публікує подію щоразу, коли вона оновлює базу даних. Оновлення бази даних і публікація події повинні відбуватися за одну транзакцію (рис. 3.2.2).

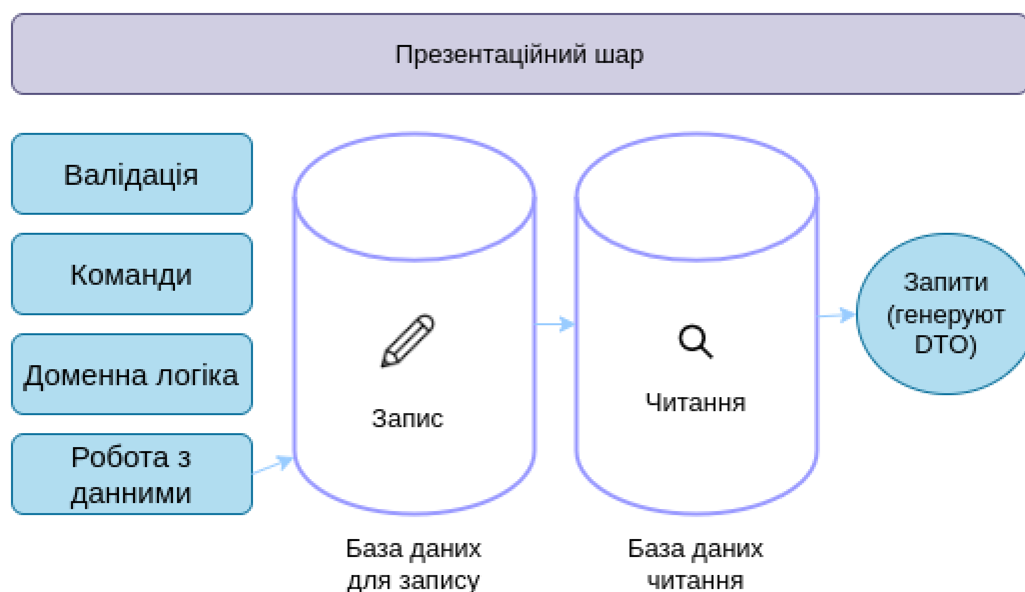


Рисунок 3.2.2 - Приклад взаємодії різних шарів при збільшені серверів бази даних згідно шаблону CQRS

Сховище для читання може бути копією сховища запису лише для читання, або сховища для читання та запису можуть мати зовсім іншу структуру. Використання кількох реплік лише для читання може підвищити продуктивність

запитів, особливо в розподілених сценаріях, де репліки лише для читання розташовані поблизу екземплярів програми.

Розділення сховищ читання і запису також дозволяє відповідним чином масштабувати кожен відповідно до навантаження. Наприклад, сховища читання зазвичай стикаються з набагато більшим навантаженням, ніж сховища запису.

Деякі реалізації CQRS використовують шаблон пошуку подій. За допомогою цього шаблону стан програми зберігається як послідовність подій. Кожна подія являє собою набір змін у даних. Поточний стан створюється шляхом відтворення подій. У контексті CQRS однією з переваг джерела подій є те, що ті самі події можна використовувати для сповіщення інших компонентів, зокрема, для сповіщення моделі зчитування. Модель читання використовує події для створення знімка поточного стану, що є більш ефективним для запитів. Однак пошук подій ускладнює дизайн.

Переваги CQRS включають:

- Незалежне масштабування . CQRS дозволяє робочим навантаженням читання та запису масштабуватися незалежно, і може призвести до меншої кількості суперечок щодо блокування.
- Оптимізовані схеми даних . Сторона читання може використовувати схему, оптимізовану для запитів, а сторона запису використовує схему, оптимізовану для оновлень.
- Безпека . Простіше переконатися, що лише правильні об’єкти домену виконують запис даних.
- Розділення турбот . Розділення сторін читання та запису може призвести до моделей, які є більш зручними та гнучкими. Більшість складної бізнес-логіки вкладається в модель запису. Модель читання може бути відносно простою.
- Простіші запити . Зберігаючи матеріалізоване представлення в прочитаній базі даних, програма може уникнути складних об’єднань під час запитів.

```
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using TripSharing.Application.Core;
using TripSharing.Repository;
```

```
namespace TripSharing.Application.Trips
{
  public class List
  {
    public class Query : IRequest<Result<List<TripDto>>>
    {
    }

    public class Handler : IRequestHandler<Query, Result<List<TripDto>>>
    {
      private readonly DataContext _context;
      private readonly IMapper _mapper;

      public Handler(DataContext context, IMapper mapper)
      {
        _context = context;
        _mapper = mapper;
      }

          public async Task<Result<List<TripDto>>> Handle(Query request,
CancellationToken cancellationToken)
      {
        var trips = await _context.Trips
          .ProjectTo<TripDto>(_mapper.ConfigurationProvider)
          .ToListAsync(cancellationToken);

        return Result<List<TripDto>>.Success(trips);
      }
    }
  }
}

using System.Threading;
using System.Threading.Tasks;
using AutoMapper;
using FluentValidation;
using MediatR;
using TripSharing.Application.Core;
using TripSharing.Domain;
```

```
using TripSharing.Repository;

namespace TripSharing.Application.Trips
{
  public class Edit
  {
    public class Command : IRequest<Result<Unit>>
    {
      public Trip Trip { get; set; }
    }

    public class CommandValidator : AbstractValidator<Command>
    {
      public CommandValidator()
      {
        RuleFor(x => x.Trip).SetValidator(new TripValidator());
      }
    }

    public class Handler : IRequestHandler<Command, Result<Unit>>
    {
      private readonly DataContext _context;
      private readonly IMapper _mapper;

      public Handler(DataContext context, IMapper mapper)
      {
        _context = context;
        _mapper = mapper;
      }

          public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
      {
        var trip = await _context.Trips.FindAsync(request.Trip.Id);

        if (trip == null)
        {
          return null;
        }
```

```
                _mapper.Map(request.Trip, trip);

                var result = await _context.SaveChangesAsync() > 0;

                return !result
                    ? Result<Unit>.Failure("Failed to update the trip")
                    : Result<Unit>.Success(Unit.Value);
            }
        }
    }
}
```

3.2.3 Використання Mediator Pattern (посередник)

У проекті не використовується пряма реалізація цього паттерну, але використовується пакет з цією реалізацією під назвою MediatR.

MediatR полегшує шаблони CQRS які також використовуються в цьому проекті. Це бібліотека з низькими амбіціями, яка намагається вирішити просту проблему — відокремити відправку повідомлень у процесі від обробки повідомлень.

MediatR підтримує запити/відповіді, команди, запити, сповіщення та події, синхронні та асинхронні з інтелектуальною диспетчеризацією через загальну дисперсію C#. Він також підтримує шаблон видавець/передплатник (publisher/subscriber).

Шаблон посередника — це поведінковий шаблон дизайну, який допомагає зменшити хаотичні залежності між об’єктами. Шаблон обмежує прямий зв'язок між об'єктами і змушує їх співпрацювати тільки через об'єкт-посередник. Посередник використовується для зменшення складності зв'язку між кількома об'єктами або класами. Цей шаблон надає клас-посередник, який зазвичай обробляє всі комунікації між різними класами і підтримує легке обслуговування коду за рахунок слабкого зв’язку.

MediatR має два види повідомлень:

1. Повідомлення запиту/відповіді, надіслані одному обробнику.
2. Повідомлення сповіщень, що надсилаються кільком обробникам.

Запити

Запити описують поведінку ваших команд і запитів. Запити - це дуже прості повідомлення в стилі запит-відповідь, де один запит синхронно обробляється одним обробником. Наприклад, повернення чогось із бази даних.

У MediatR є два види запитів — ті, які повертають значення, і ті, які ні. Часто це відповідає читанням/запитам (повертає значення) і запису/командам (зазвичай не повертає значення).

- IRequest<T>- запит повертає значення.
- IRequest- запит не повертає значення.

Обробник

Коли запит буде створено, вам знадобиться обробник для вирішення запиту. Кожен тип запиту має власний інтерфейс обробника, а також деякі допоміжні базові класи/інтерфейси. Вони залежать від двох параметрів:

1. запит.
2. відповідь.

IRequestHandler<T, U> - реалізувати це і повернути Task<U>

RequestHandler<T, U> - успадкувати це і повернути U

Сповіщення

Один запит обробляється одним обробником. Що робити, якщо ми хочемо обробити один запит кількома обробниками? Ось тут і приходять сповіщення. У цих ситуаціях зазвичай кілька незалежних операцій, які мають відбутися після певної події.

Наприклад, коли реєструється новий клієнт, ми хочемо надіслати клієнту електронну пошту та надіслати SMS адміністратору.

Для сповіщень спочатку створюється повідомлення, а потім створюється нуль або більше обробників для сповіщень.

MediatR використовується у контролерах для запитів або команд

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using TripSharing.Application.Trips;
using TripSharing.Domain;

namespace TripSharing.Controllers
{
  public class TripsController : BaseApiController
  {
    [HttpGet]
    public async Task<IActionResult> GetTrips()
    {
      return HandleResult(await Mediator.Send(new List.Query()));
```

```
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> GetTrip(Guid id)
        {
            return HandleResult(await Mediator.Send(new Details.Query{Id = id}));
        }

        [HttpPost]
        public async Task<IActionResult> CreateTrip(Trip trip)
        {
            return HandleResult(await Mediator.Send(new Create.Command {Trip =
trip}));
        }

        [Authorize(Policy = "IsTripDriver")]
        [HttpPut("{id}")]
        public async Task<IActionResult> EditTrip(Guid id, Trip trip)
        {
            trip.Id = id;
            return HandleResult(await Mediator.Send(new Edit.Command {Trip =
trip}));
        }

        [Authorize(Policy = "IsTripDriver")]
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteTrip(Guid id)
        {
            return HandleResult(await Mediator.Send(new Delete.Command {Id =
id}));
        }

        [HttpPost("{id}/attend")]
        public async Task<IActionResult> Attend(Guid id)
        {
            return HandleResult(await Mediator.Send(new
UpdateAttendance.Command {Id = id}));
        }
    }
```

```
}

using System;
using System.Threading;
using System.Threading.Tasks;
using AutoMapper;
using AutoMapper.QueryableExtensions;
using MediatR;
using Microsoft.EntityFrameworkCore;
using TripSharing.Application.Core;
using TripSharing.Repository;

namespace TripSharing.Application.Trips
{
    public class Details
    {
        public class Query : IRequest<Result<TripDto>>
        {
            public Guid Id { get; set; }
        }

        public class Handler : IRequestHandler<Query, Result<TripDto>>
        {
            private readonly DataContext _context;
            private readonly IMapper _mapper;

            public Handler(DataContext context, IMapper mapper)
            {
                _context = context;
                _mapper = mapper;
            }

            public async Task<Result<TripDto>> Handle(Query request,
CancellationToken cancellationToken)
            {
                var trip = await _context.Trips
                    .ProjectTo<TripDto>(_mapper.ConfigurationProvider)
                    .FirstOrDefaultAsync(x => x.Id == request.Id);
```

```
            return Result<TripDto>.Success(trip);
        }
    }
  }
}
```

3.3 Використання Code First

Entity Framework представив підхід Code-First з версії Entity Framework 4.1. Code-First в основному корисний у Domain Driven Design . У підході Code-First ви зосереджуєтесь на домені вашої програми і починаєте створювати класи для вашої сутності домену, а не спочатку проектуєте свою базу даних, а потім створюєте класи, які відповідають дизайну вашої бази даних. Наступний малюнок ілюструє підхід з використанням коду.

Рисунок 3.3 - Створення бази даних з використання підходу Code First

Як ви можете бачити на малюнку вище (рис 3.3), EF API створить базу даних на основі ваших класів домену та конфігурації. Це означає, що вам потрібно спочатку почати кодування на C# а потім EF створить базу даних з вашого коду.

Наступний малюнок ілюструє робочий процес розробки коду в першу чергу.

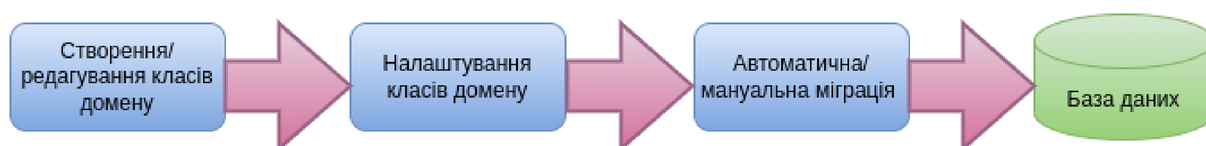


Рисунок 3.3 - Процес створення бази даних при підході Code First

Робочий процес розробки в підході з використанням коду буде таким:

1. Створення або зміна класів домену
2. Налаштування цих класів домену за допомогою Fluent-API або атрибутів анотації даних
3. Створення або оновлення схеми бази даних за допомогою автоматичної міграції або міграції на основі коду.

```
using System;
using System.Collections.Generic;

namespace TripSharing.Domain
{
  public class Trip
  {
    public Guid Id { get; set; }
    public DateTime Date { get; set; }
    public bool Status { get; set; }

        public ICollection<TripAttendee> Attendees { get; set; } = new
List<TripAttendee>();

         public ICollection<Comment> Comments { get; set; } = new
List<Comment>();
  }
}

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using TripSharing.Domain;
```

```
namespace TripSharing.Repository
{
  public class DataContext : IdentityDbContext<AppUser>
  {
    public DataContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Trip> Trips { get; set; }

    public DbSet<TripAttendee> TripAttendees { get; set; }
    public DbSet<Photo> Photos { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
      base.OnModelCreating(builder);

       builder.Entity<TripAttendee>(x => x.HasKey(aa => new {aa.AppUserId,
aa.TripId}));

      builder.Entity<TripAttendee>()
        .HasOne(u => u.AppUser)
        .WithMany(t => t.Trips)
        .HasForeignKey(aa => aa.AppUserId);

      builder.Entity<TripAttendee>()
        .HasOne(u => u.Trip)
        .WithMany(t => t.Attendees)
```

```
        .HasForeignKey(aa => aa.TripId);

    builder.Entity<Comment>()
        .HasOne(a => a.Trip)
        .WithMany(c => c.Comments)
        .OnDelete(DeleteBehavior.Cascade);

    builder.Entity<AppUser>(entity => entity.Property(m => m.NormalizedEmail).HasMaxLength(200));
    builder.Entity<AppUser>(entity => entity.Property(m => m.Id).HasMaxLength(200));
    builder.Entity<AppUser>(entity => entity.Property(m => m.NormalizedUserName).HasMaxLength(200));

    builder.Entity<Photo>(entity => entity.Property(m => m.Id).HasMaxLength(200));

    builder.Entity<IdentityRole>(entity => entity.Property(m => m.NormalizedName).HasMaxLength(200));
    builder.Entity<IdentityRole>(entity => entity.Property(m => m.Id).HasMaxLength(200));

    builder.Entity<IdentityUserLogin<string>>(entity => entity.Property(m => m.UserId).HasMaxLength(200));
    builder.Entity<IdentityUserLogin<string>>(entity => entity.Property(m => m.LoginProvider).HasMaxLength(200));
    builder.Entity<IdentityUserLogin<string>>(entity => entity.Property(m => m.ProviderKey).HasMaxLength(200));
```

```
            builder.Entity<IdentityUserRole<string>>(entity => entity.Property(m =>
m.UserId).HasMaxLength(200));
            builder.Entity<IdentityUserRole<string>>(entity => entity.Property(m =>
m.RoleId).HasMaxLength(200));

            builder.Entity<IdentityUserToken<string>>(entity => entity.Property(m =>
m.UserId).HasMaxLength(200));
            builder.Entity<IdentityUserToken<string>>(entity => entity.Property(m =>
m.LoginProvider).HasMaxLength(200));
            builder.Entity<IdentityUserToken<string>>(entity => entity.Property(m =>
m.Name).HasMaxLength(200));

            builder.Entity<IdentityUserClaim<string>>(entity => entity.Property(m =>
m.UserId).HasMaxLength(200));
            builder.Entity<IdentityRoleClaim<string>>(entity => entity.Property(m =>
m.RoleId).HasMaxLength(200));
        }
    }
}
```

3.4 Використання обробки виключень

Функції обробки винятків мови C# допомагають впоратися з будь-якими несподіваними або винятковими ситуаціями, які виникають під час роботи програми. Обробка винятків використовує ключові слова try, catch і finally ключові слова, щоб спробувати дії, які можуть не увінчатися успіхом, обробити помилки, коли ви вирішите, що це доцільно зробити, а потім очистити ресурси. Винятки можуть створюватися за допомогою загальномовного середовища виконання (CLR), .NET або сторонніх бібліотек або коду програми. Винятки створюються за допомогою throw ключового слова.

У багатьох випадках виняток може бути викликаний не методом, який ваш код викликав безпосередньо, а іншим методом, який знаходиться нижче в стеку викликів. Коли виникає виняток, CLR розгортає стек, шукаючи метод з catch блоком для конкретного типу винятку, і виконує перший такий catch блок, який знайде. Якщо catch в стеку викликів немає відповідного блоку, він завершить процес і відобразить повідомлення користувачеві.

Винятки мають такі властивості:

- Виняток становлять типи, які в кінцевому підсумку походять від System.Exception.
- Використовуйте try блок навколо операторів, які можуть викликати винятки.
- Як тільки в блоці виникає виняток try, потік керування переходить до першого пов’язаного обробника винятків, який присутній у будь-якому місці стеку викликів. У C# catch ключове слово використовується для визначення обробника винятків.
- Якщо для даного винятку немає обробника винятків, програма припиняє виконання з повідомленням про помилку.
- Не ловіть виняток, якщо ви не можете обробити його та залишити програму у відомому стані. Якщо ви зловите System.Exception, перекиньте його за допомогою throw ключового слова в кінці catch блоку.
- Якщо catch блок визначає змінну винятку, ви можете використовувати її для отримання додаткової інформації про тип винятку, що відбувся.
- Винятки можуть бути створені програмою явно за допомогою throw ключового слова.
- Об’єкти винятків містять детальну інформацію про помилку, таку як стан стеку викликів і текстовий опис помилки.
- Код у finally блоці виконується незалежно від того, чи виникне виняток. Використовуйте finally блок, щоб звільнити ресурси, наприклад, щоб закрити будь-які потоки або файли, які були відкриті в try блоці.

- Керовані винятки в .NET реалізовані поверх структурованого механізму обробки винятків Win32

Проміжне програмне забезпечення (middleware) — це програмне забезпечення, яке збирається в конвеєр програми для обробки запитів і відповідей. Кожен компонент:

- Вибирає, чи передавати запит наступному компоненту в конвеєрі.
- Може виконувати роботу до і після наступного компонента в конвеєрі.

Для побудови конвеєра запитів використовуються делегати запитів. Делегати запиту обробляють кожен запит HTTP.

Делегати запитів налаштовуються за допомогою методів розширення Run, Map і Use. Окремий делегат запиту може бути визначений в рядку як анонімний метод (званий вбудованим проміжним програмним забезпеченням) або він може бути визначений у повторно використовуваному класі. Ці повторно використовувані класи та вбудовані анонімні методи є проміжним програмним забезпеченням, яке також називають компонентами проміжного програмного забезпечення. Кожен компонент проміжного програмного забезпечення в конвеєрі запитів відповідає за виклик наступного компонента в конвеєрі або за коротке замикання конвеєра. Коли проміжне програмне забезпечення замикає, воно називається термінальним проміжним програмним забезпеченням, оскільки воно перешкоджає подальшому проміжному програмному забезпеченню обробити запит.

В прикладі нижче приведено код такого делегата, у якому використано блок для обробки виключення з використанням кастомного класу виключень AppException.

```
using System;
using System.Net;
using System.Text.Json;
using System.Threading.Tasks;
```

```
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using TripSharing.Application.Core;

namespace TripSharing.Middleware
{
  public class ExceptionMiddleware
  {
     private readonly RequestDelegate _next;
     private readonly ILogger<ExceptionMiddleware> _logger;
     private readonly IHostEnvironment _env;

     public ExceptionMiddleware(RequestDelegate next,
ILogger<ExceptionMiddleware> logger, IHostEnvironment env)
     {
        _next = next;
        _logger = logger;
        _env = env;
     }

     public async Task InvokeAsync(HttpContext context)
     {
        try
        {
           await _next(context);
        }
        catch (Exception e)
        {
           _logger.LogError(e, e.Message);
```

```
                context.Response.ContentType = "application/json";
                context.Response.StatusCode = (int) HttpStatusCode.InternalServerError;

                var response = _env.IsDevelopment()
                    ? new AppException(context.Response.StatusCode, e.Message, e.StackTrace?.ToString())
                    : new AppException(context.Response.StatusCode, "Server Error");

                var options = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };
                var json = JsonSerializer.Serialize(response, options);

                await context.Response.WriteAsync(json);
            }
        }
    }
}

namespace TripSharing.Application.Core
{
    public class AppException
    {
        public AppException(int statusCode, string message, string details = null)
        {
            StatusCode = statusCode;
            Message = message;
            Details = details;
        }
```

```
    public int StatusCode { get; set; }
    public string Message { get; set; }
    public string Details { get; set; }
  }
}
```

4 ВИСНОВКИ

У ході виконання дипломного проекту були досліджені та випробувані методи створення веб-додатку для організації спільних поїздок працівників компанії. Проведено дослідження ефективності використання різних технологій для реалізації функціонального і готового до експлуатації веб-додатку. Для цього було проаналізовано переваги програмного забезпечення, редакторів вихідного коду, які допомагають полегшити розробку. Розглянуті та використані технології показали, що розумний підбір технологій для розробки проекту може полегшити процес розробки. В результаті дипломної роботи було отримано працездатний веб-додаток. Незаперечно цей проект буде і надалі розвиватись, в планах розвивати готовий функціонал, додавати новий, робити можливість інтегрування проекту у інші, покращувати зовнішній вид та зручність використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. uber.com [Електронний ресурс]. – Режим доступу: https://www.uber.com/uk-UA/blog/ubershuttle-is-in-kyiv/
2. webcase.com.ua [Електронний ресурс]. – Режим доступу: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/
3. training.qatestlab.com [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
4. developer.mozilla.org [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/HTML
5. uk.javascript.info [Електронний ресурс]. – Режим доступу: https://uk.javascript.info/intro
6. uk.peterfeatherstone.com [Електронний ресурс]. – Режим доступу: https://uk.peterfeatherstone.com/311-what-is-typescript-strongly-typed-javascript
7. simplilearn.com [Електронний ресурс]. – Режим доступу: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs
8. docs.microsoft.com [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
9. uk.education-wiki.com [Електронний ресурс]. – Режим доступу: https://uk.education-wiki.com/2686691-what-is-mysql-database
10. blog.cleancoder.com [Електронний ресурс]. – Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
11. docs.microsoft.com [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/uk-ua/dotnet/csharp/fundamentals/exceptions/
12. medium.com [Електронний ресурс]. – Режим доступу: https://medium.com/dotnet-hub/use-mediatr-in-asp-net-or-asp-net-core-cqrs-and-mediator-in-dotnet-how-to-use-mediatr-cqrs-aspnetcore-5076e2f2880c
13. entityframeworktutorial.net [Електронний ресурс]. – Режим доступу: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx

14. www.baeldung.com [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/java-dto-pattern

15. martinfowler.com [Електронний ресурс]. – Режим доступу: https://martinfowler.com/books/eaa.html

16. docs.microsoft.com [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/uk-ua/azure/architecture/patterns/cqrs

17. bestprog.net [Електронний ресурс]. – Режим доступу: https://www.bestprog.net/uk/2021/06/30/c-generics-basic-concepts-generic-classes-and-structures-ua/

18. qagroup.com.ua [Електронний ресурс]. – Режим доступу: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/

19. docs.microsoft.com [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/uk-ua/dotnet/core/extensions/dependency-injection

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка додатку для спільних поїздок працівників компанії, з використанням Net Core та Entity Framework

Виконав (ла) студент(ка) 5 курсу
групи ППЗ-51
Пелюховський С. С.
Керівник роботи
Гаманюк І. М.

Київ – 2022

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - автоматизація створення, бронювання та відслідковування поїздок через додаток, створенний з використанням Net Core та Entity Framework, написаний на мові C#
- **Об’єкт дослідження -** процес створення, резервування та відслідковування спільних поїздок працівників компанії
- **Предмет дослідження -** програмне забезпечення для спільних поїздок працівників компанії

АНАЛОГИ

Орієнтовна карта маршрутів Shuttle

Троєщина ← → Кудряшова, Печерськ, КПІ, Чернігівська

Позняки ← → Центр, Поділ, КПІ, Дорогожицька, Протасів Яр

Виноградар ← → Центр, КПІ

Коцюбинське ← → Поділ

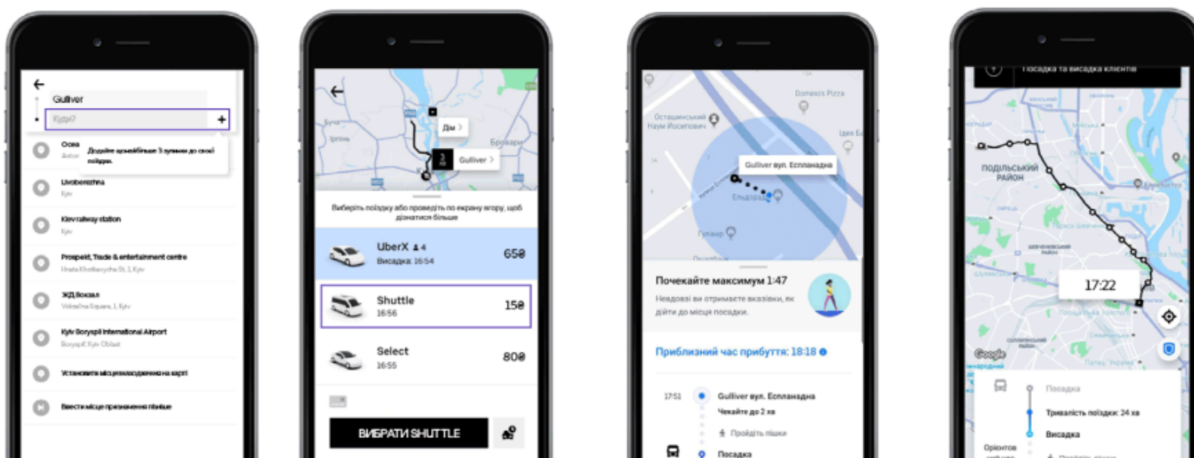
Борщагівка ← → Печерськ, Поділ

Петропавлівська Борщагівка ← → Поділ

Мінський масив ← → Протасів Яр

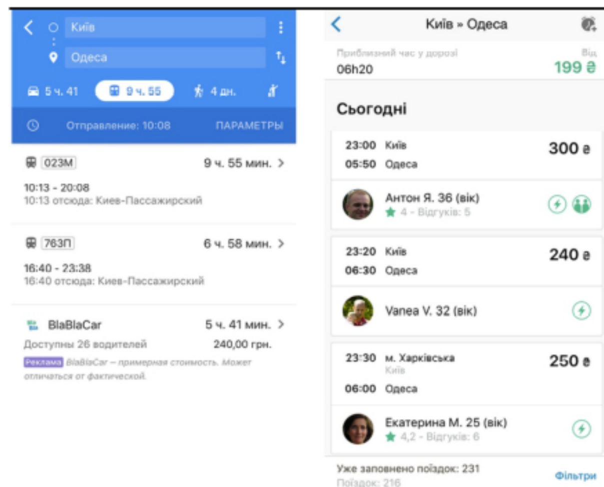
Приклад маршруту Uber Shuttle

АНАЛОГИ



Приклад інтерфейсу Uber Shuttle

АНАЛОГИ



Приклад інтерфейсу Blablacar

ТЕХНІЧНІ ЗАВДАННЯ

- *1. Створити та підключити до проекту базу даних*
- *2. Створити з шаблону .Net Web API веб-додатку базову структуру проекту*
- *3. У застосунку повинна бути можливість реєстрації нових користувачів як водіїв так і пасажирів*
- *4. У застосунку повинні бути екрани для входу або виходу із додатка*
- *5. У користувача з роллю водія повинна бути можливість створювати, змінювати або відміняти поїздки*

ТЕХНІЧНІ ЗАВДАННЯ

- *6. У користувача з роллю пасажира повинна бути можливість*
- *7. В додатку повинен бути присутнім чат на сторінці поїздки для комунікації з водієм*
- *8. Створити сторінку профайла з інформації про користувача*

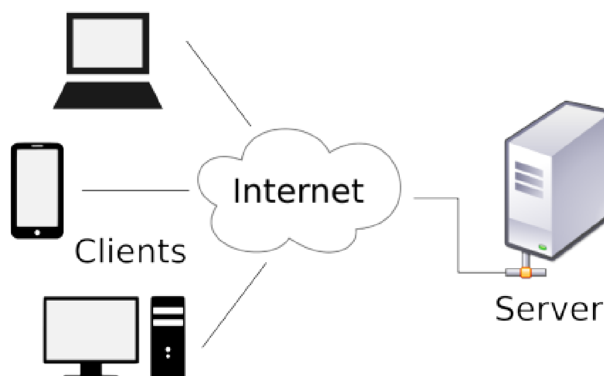
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Для реалізації поставленої задачі нам знадобиться перш завсе редактор коду або IDE (integrated development environment).Також нам знадобиться база даних MySQL для збереження даних про користувачів та поїздок. І заключне, нам потрібно сховище для зберігання та відслідковування змін у коді, я використовую ресурс Github.com так як він безкоштовний, популярний, функціональний.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Класи, які будуть приймати запит від користувача, називаються контролерам. В залежності від типу запиту (Get, Post, Put, Delete) та роуту (або сторінки) буде Виконуватись відповідний метод контролера



Приклад запитів від користувачів

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using TripSharing.Application.Profiles;

namespace TripSharing.Controllers
{
    public class ProfilesController : BaseApiController
    {
        [HttpGet("{username}")]

        public async Task<IActionResult> GetProfile(string username)
        {
            return HandleResult(await Mediator.Send(new Details.Query {UserName = username}));
        }
    }
}
```

Приклад класа контролера профайлів

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

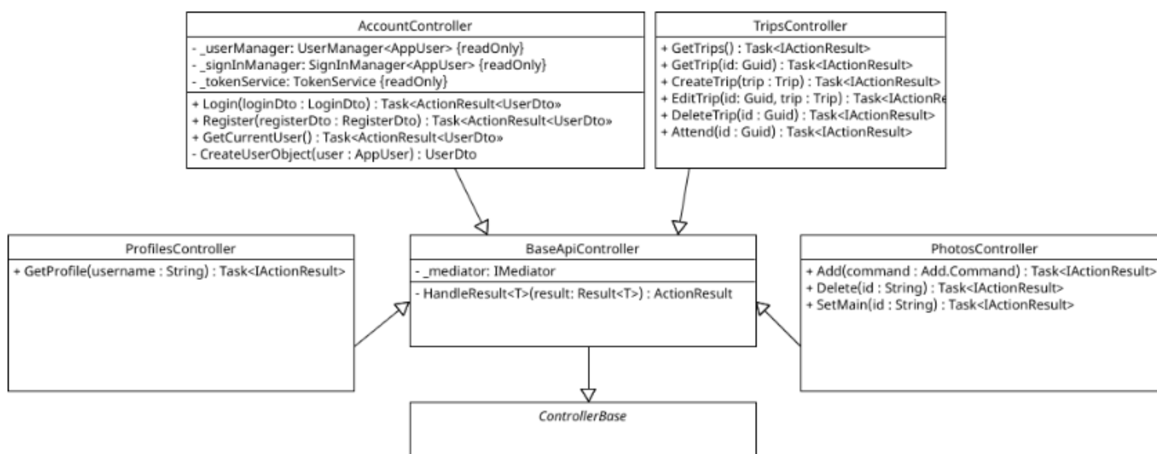
```
using ...

namespace TripSharing.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class BaseApiController : ControllerBase
    {
        private IMediator _mediator;

        protected IMediator Mediator => _mediator ??= HttpContext
            .RequestServices
            .GetService<IMediator>();

        protected ActionResult HandleResult<T>(Result<T> result)
        {
            if (result == null)
            {
                return NotFound();
            }

            if (result.IsSuccess && result.Value != null)
            {
                return Ok(result.Value);
            }

            if (result.IsSuccess && result.Value == null)
            {
                return NotFound();
            }

            return BadRequest(result.Error);
        }
    }
}
```

Базовий клас контролера BaseApiController

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Як ми бачимо на прикладі контролера ProfileController, клас ProfileController наслідує базовий клас BaseApiController в якому описана базова логіка для обробки запитів і відповідей на них. Перед методом ми вказуємо атрибут HttpGet – це означає, що даний метод буде обробляти тільки Get запити, при цьому в гет запиті повинний бути присутній стріка username. Метод GetProfile приймає стрічку username, та викликає метод HandleResult, який реалізований в базовому класі. Далі через ланцюг визовів викликаємо метод Query у класа Details та передаємо йому стрічку username. Якщо у нас є в базі даних такий користувач, то повертаємо результат з даними цього користувача, якщо не має – повідомлення, що користувача не знайдено

Діаграма класів контролерів, їх методів та залежностей



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Пелюховський С.С Розробка програмного забезпечення щодо додатку для спільних поїздок працівників компанії з використанням хмарної технології / Пелюховський С.С. // Розробка програмного забезпечення щодо додатку для спільних поїздок працівників компанії на мові програмування c# та javascript. Збірник тез. 20.04.2022, ДУТ, м. Київ — К.: ДУТ, 2022

ВИСНОВКИ

Під час проходження переддипломної практики на кафедрі інженерії програмного забезпечення в Державному університеті телекомунікацій було виконано всі поставлені задачі індивідуального завдання.

Підібрано та проаналізовано 5 джерел, з них 3 іншомовних.

Проведено аналіз можливості організації поїздок для працівників компанії. Ключові проблеми, які можна виділити, пов’язані з відсутності потрібного за функціонуванням додатку. Це призводить до того, що підлаштовуватись під багато факторів, при організації схожих поїздок.

ВИСНОВКИ

Обрано мову програмування C# та JavaScript, що забезпечує стабільну та швидку роботу додатку, кросрплатформленість та гнучкість для розширення додатку.
Визначено об’єкт, предмет та мету бакалаврської роботи.
Робота пройшла апробацію на науково-технічна конференція "Застосування програмного забезпечення в інфокомунікаційних технологіях". За результатами апробації опубліковано тези доповідей:

Розробка програмного забезпечення щодо додатку для спільних поїздок працівників компанії з використанням хмарної технології.
Розробка програмного забезпечення щодо додатку для спільних поїздок працівників компанії на мові програмування c# та javascript

ВИСНОВКИ

Отримані під час переддипломної практики результати будуть використані в кваліфікаційній роботі бакалавра.

ДЯКУЮ ЗА УВАГУ!