**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра програмного забезпечення**

**Звіт**  
про переддипломну практику за темою бакалаврської кваліфікаційної роботи

**Веб-система моніторингу та планування пасажирських авіаперевезень**

тема

студента (ки) IV курсу групи ПІ-41

Гаврилюк А.М.

(прізвище, ініціали) 

**база практики ТзОВ «Едвантіс»**\_\_\_\_\_\_\_\_\_\_\_\_\_

**\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва підприємства)

**термін практики** з «03.04.2017» до «29.04.2017»   
 

**Керівники практики**

від кафедри\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_доцент Левус Є.В.

(підпис)     (посада, ПІП)

від бази практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)     (посада, ПІП)   
 

**Керівник бакалаврської** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ д.т.н., доцент Коротеєва Т.О.

(підпис)     (наук.ст., вч.звання, ПІП)

**кваліфікаційної роботи**

**Оцінка**\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Дата\_**\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів 2017

ЗМІСТ

[1. Опис бази практики 3](#_Toc481706204)

[2. Завдання отримане на базі практики 3](#_Toc481706205)

[3. Результат виконання завдання на базі практики 3](#_Toc481706206)

[3.1 Ознайомлення з фреймворком AngularJS та мовою C# 3](#_Toc481706207)

[3.2 Демо-застосунок, використовуючий описану вище технологію 6](#_Toc481706208)

[4. Завдання стосовно бакалаврської кваліфікаційної роботи 9](#_Toc481706209)

[**Розділ 1. Огляд програмного забезпечення для моніторингу та планування пасажирських авіаперевезень** 9](#_Toc481706210)

[**Розділ 2. Постановка завдання для розробки програмного забезпечення для моніторингу та планування пасажирських авіаперевезень** 16](#_Toc481706214)

[**Розділ 3. Архітектура і проектування програмного забезпечення для моніторингу та планування пасажирських авіаперевезень** 2](#_Toc481706218)7

[5. Висновки про отримані під час практики результати 3](#_Toc481706219)7

[СПИСОК ОПРАЦЬОВАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 3](#_Toc481706220)8

[Додаток А 3](#_Toc481706221)9

## Опис бази практики

Edvantis – компанія з створення програмного забезпечення. Заснована у Львові, також розташовані відділи у Польщі, Німеччині, Канаді та США. Переважно займається розробкою веб-сайтів або додатків для мобільних платформ, таких як Android та iOS. Переважно компанія розробляє серверну частину своїх продуктів на технологіях Java та C#. Компанія обслуговує різні корпорації та інші компанії, і разом з тим обслуговує індивідуальних клієнтів та стартапи. Основними принципами компанії є: прозорість процесу розробки, гнучкість, прагнення допомогти клієнту. Компанія працює за наступною схемою: є декілька команд які розробляють певні проекти, вони підпорядковуються менеджеру проекту, проектні менеджери підпорядковуються головному менеджеру офісу. Більшість команд працює за методологією Scrum.

## Завдання отримане на базі практики

Згідно переддипломної практики на базі практики ТзОВ «Едвантіс», було отримано наступні завдання:

1. Ознайомитися з фреймворками AngularJS та вдосконалити знання мови C#.
2. Розробити застосунок, використовуючи вказані вище технології.

## Результат виконання завдання на базі практики

### Ознайомлення з фреймворком AngularJS та мовою C#

AngularJS це JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки.

Фреймворк працює зі сторінкою HTML, що містить додаткові атрибути і пов'язує області вводу або виводу сторінки з моделлю, яка є звичайними змінними JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних.

Фреймворк адаптує та розширює традиційний HTML, щоб забезпечити двосторонню прив'язку даних для динамічного контенту, що дозволяє автоматично синхронізувати модель та вид. У результаті AngularJS зменшує роль DOM-маніпуляцій з метою підвищення продуктивності і спрощення тестування.

Директиви AngularJS дозволяють розробнику модифікувати поведінку деяких елементів, чи описати власні елементи. Основними директивами є:

* **ng-app**

Оголошує елемент кореневим елементом застосунку, дозволяючи змінювати поведінку за допомогою спеціальних тегів.

* **ng-bind**

Змінює текст елемента на значення виразу. <span ng-bind="name"></span> відобразить значення змінної name всередині тегу span. Будь-які зміни змінної будуть миттєво відображені в DOM, де б змінна не використовувалась.

* **ng-model**

Подібна до ng-bind, але дозволяє двостороннє зв`язування даних між (зміни в DOM будуть змінювати змінну).

* **ng-class**

Дозволяє динамічно додавати та забирати класи елемента.

* **ng-controller**

Вказує клас JavaScript контролера.

* **ng-repeat**

Створює кілька екземплярів елемента, для кожного об'єкта колекції.

* **ng-show & ng-hide**

Показують чи ховають елемент залежно від значення булевого виразу. Це досягається за допомогою задання в CSS атрибуту display.[1]

Мова C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників — мов С++, Delphi, Модула і Smalltalk — С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++).

C# є дуже близьким родичем мови [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [Java](https://uk.wikipedia.org/wiki/Java). Мова Java була створена компанією Sun Microsystems, коли глобальний розвиток інтернету поставив задачу роззосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу вказівників без контролю виходу за межі). Для роззосереджених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програм і апаратними інструкціями комп'ютера чи іншого інтелектуального пристрою.

C# розроблялась як мова програмування прикладного рівня для [CLR](https://uk.wikipedia.org/wiki/CLR) і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо.

В червні 2004 року [Андерс Гейлсберг](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81_%D0%93%D0%B5%D0%B9%D0%BB%D1%81%D0%B1%D0%B5%D1%80%D0%B3) вперше розповів на сайті Microsoft про плановані розширення мови в C#3.0.[[1]](https://uk.wikipedia.org/wiki/C_Sharp#cite_note-1). У вересні 2005 року було випущено проект [специфікації](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F) C# 3.0 і бета-версія C# 3.0, що встановлюється у вигляді доповнення до існуючих [Visual Studio](https://uk.wikipedia.org/wiki/Visual_Studio) 2005 і [.NET](https://uk.wikipedia.org/wiki/.NET) 2.0 [[2]](https://uk.wikipedia.org/wiki/C_Sharp#cite_note-2). Офіційно версія C# 3.0 побачила світ [19 листопада](https://uk.wikipedia.org/wiki/19_%D0%BB%D0%B8%D1%81%D1%82%D0%BE%D0%BF%D0%B0%D0%B4%D0%B0) [2007 року](https://uk.wikipedia.org/wiki/2007) у складі [.NET Framework 3.5](https://uk.wikipedia.org/w/index.php?title=.NET_Framework_3.5&action=edit&redlink=1).

В C# 3.0 з'явилися такі радикальні доповнення та зміни:

Ключові слова select, from, where, що дозволяють робити запити з [SQL](https://uk.wikipedia.org/wiki/SQL), [XML](https://uk.wikipedia.org/wiki/XML), колекції тощо (запит, інтегрований в мову, [англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) [Language Integrated Query](https://uk.wikipedia.org/w/index.php?title=Language_Integrated_Query&action=edit&redlink=1), або LINQ)

Методи-розширення — додавання методу в існуючий клас за допомогою ключового слова this при першому параметрі статичної функції.

C# 3.0 буде сумісний з C# 2.0 за генерованим [MSIL](https://uk.wikipedia.org/w/index.php?title=MSIL&action=edit&redlink=1)-кодом; поліпшення в мові — чисто синтаксичні і реалізуються на етапі [компіляції](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80). Наприклад, багато з інтегрованих запитів LINQ можна реалізувати в поточних версіях використовуючи безіменні делегати в поєднанні з предикативними методами над контейнерами, на кшталт List.FindAll і List.RemoveAll.

### Демо-застосунок, що використовує фреймворк AngularJS та мову C#

Для демонстрації переваг та можливостей цих фреймворків, було вирішено розробити сайт для моніторингу та планування пасажирських авіаперевезень для авіакомпанії.

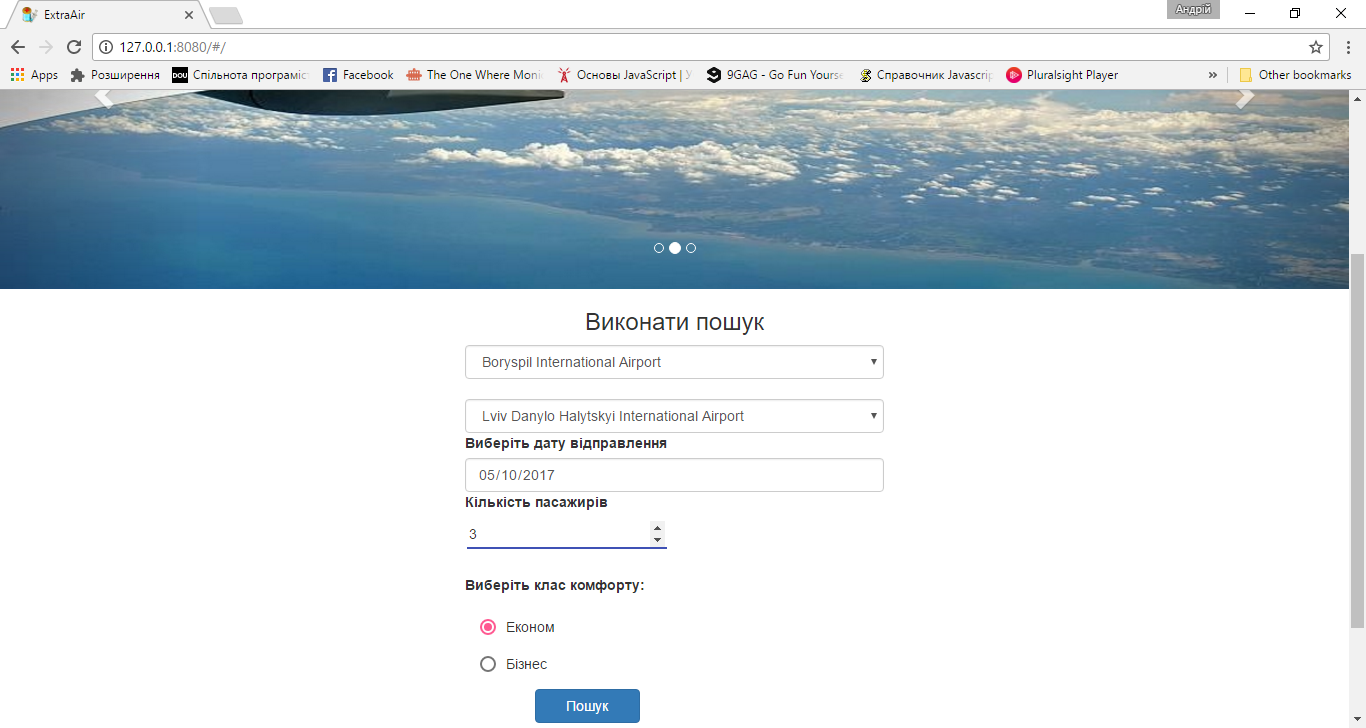
Результати виконання завдання відображені на рис. 3.2.1 - 3.2.3 Вихідний код сторінки можна знайти у додатку А. 

Рис. 3.2.1 Розроблена сторінка для пошуку рейсів(з використанням вище вказаних технологій).

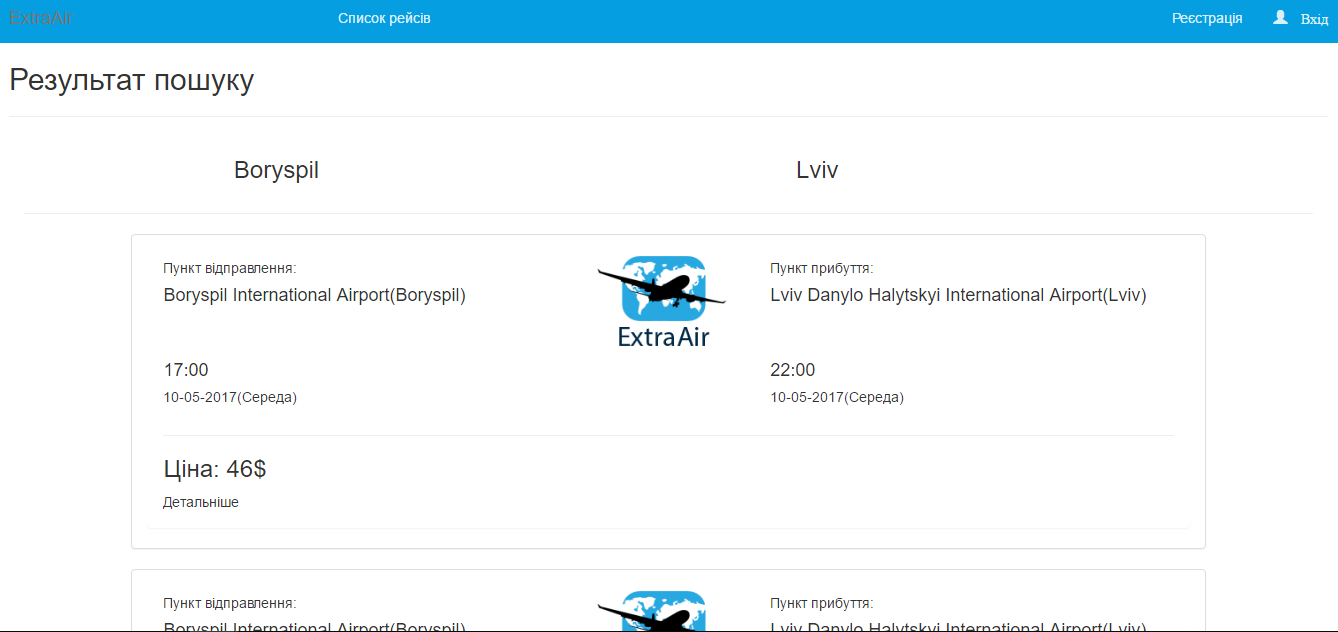


Рис. 3.2.2 Сторінка із рузультатом пошуку рейсів.

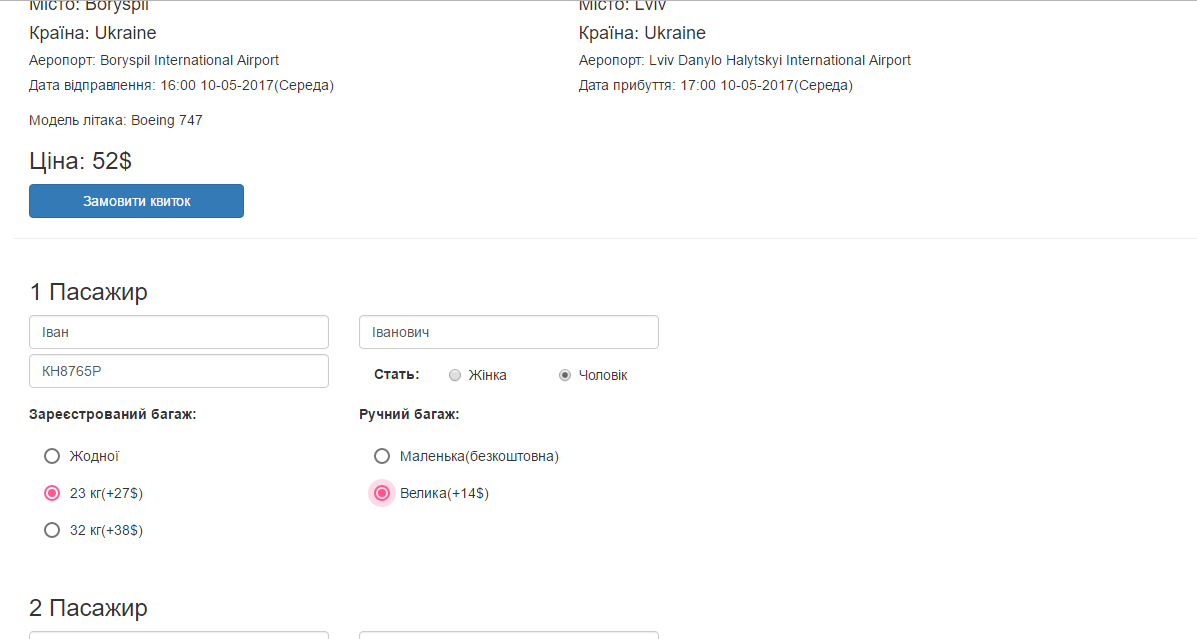


Рис. 3.2.3 Сторінка із детальною інформацією про рейс та подальшим замовленням квитків.

## Завдання стосовно бакалаврської кваліфікаційної роботи

Стосовно бакалаврської кваліфікаційної роботи було винесено наступні завдання:

1. Провести аналіз програмного забезпечення для моніторингу та планування пасажирських авіаперевезень.
2. Скласти специфікацію вимог до ПЗ.
3. Вибрати інструментальні засоби для розробки ПЗ.
4. Розробити прототип інтерфейсу користувача.
5. Змоделювати діаграми варіантів використання та класів.

В результаті виконання цих завдань було сформовано 2 розділи дипломної роботи та частково 3-й розділ.

# **Розділ 1. Огляд програмного забезпечення для моніторингу та планування пасажирських авіаперевезень**

## Опис предметної області

Подорож — переміщення якоюсь певною територією з метою її вивчення, а також із загальноосвітньою, пізнавальною, спортивною цілями. Подорожі відіграють велику роль у житті людини, а з появою авіалайнерів вони стали набагато швидшими.

Загалом подорож літаком має такі переваги:

— Найважливішою перевагою подорожей повітряним транспортом є його висока швидкість пересування. Всього за кілька годин можна перетнути тисячі кілометрів і опинитися в потрібному вам місці. Особливо це зручно при обмеженому часі.

— Подорож літаком не здається такою нудною, як, наприклад, на поїзді або автобусі. Адже тут не потрібно декілька днів витрачати на поїздку.

— Літаки відрізняються високим сервісом. Пасажиру під час польоту надається їжа, напої. До того ж тут пропонуються всілякі способи, що дозволяють скоротати час, наприклад, подивитися фільм, ознайомитися з пресою або погортати журнал.

— Під час польоту пасажир відчуває себе максимально комфортно і зручно, що забезпечується зручними сидіннями, що дозволяють чудово відпочити і навіть поспати.

— Дуже вражає вид з ілюмінатора, красиві пейзажі, розташовані внизу або білосніжні хмари, що огортають літак.

— Сьогодні існує унікальна можливість забронювати квиток по телефону або Інтернету і забути про незліченні черги. Причому, доступна така послуга у будь-який час, найбільш зручний для вас.

— Літаком можна відправитися в будь-яку країну, куди не завжди можна дістатися залізничним або автомобільним транспортом.

Дивлячись на всі ці переваги нe дивно, що сьогодні люди в основному подорожують на літаку. Він поєднує в собі комфорт і швидкість, ви доберетеся до місця призначення дуже швидко. Без сумніву, подорож літаком – найзручніше і комфортний засіб пересування.

## 1.2. Огляд аналогів

Переглянувши декілька аналогів мною було вибрано три найяскравіші приклади:

WizzAir —  [угорсько-польська бюджетна авіакомпанія, із вільни](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)м серверним [веб-застосунок](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA)ом для користування послугами компанії.

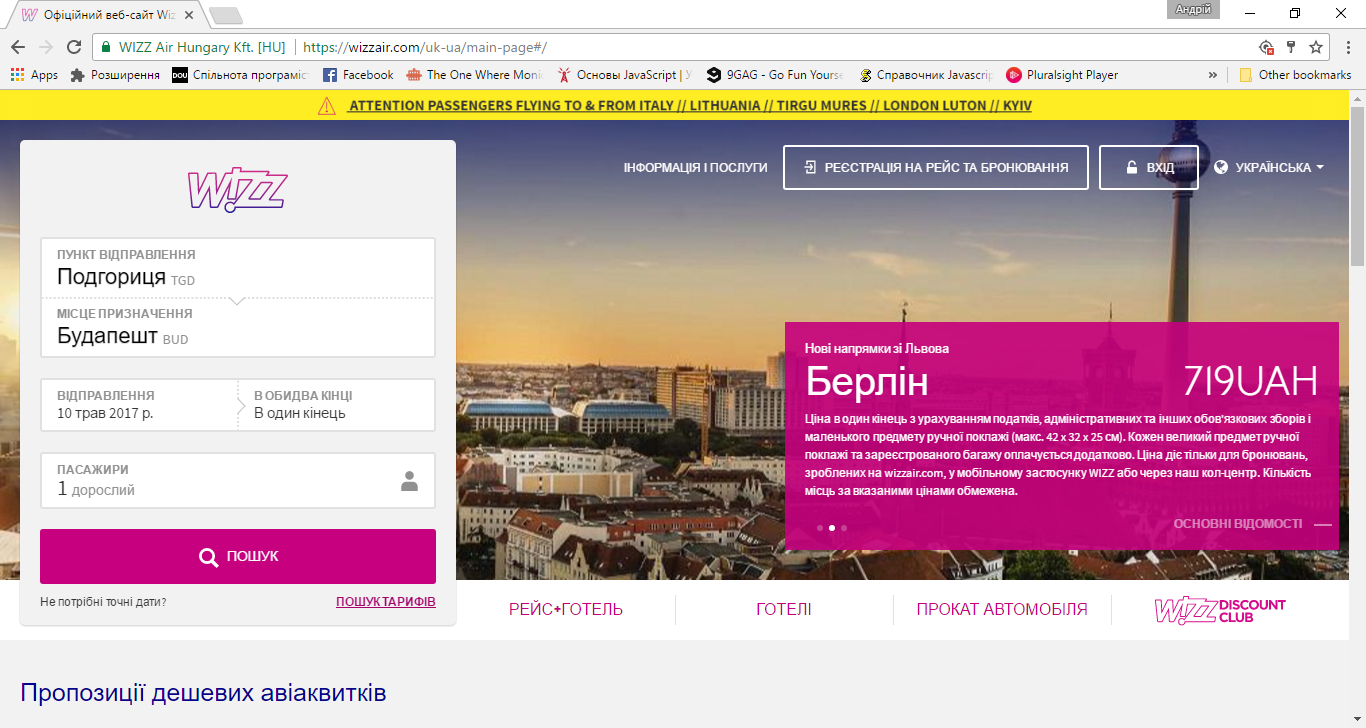


Рис. 4.1.2.1 Головна сторінка сервісу WizzAir



Рис. 4.1.2.2 Сторінка результату пошуку рейсів та подальшим замовленням сервісу WizzAir

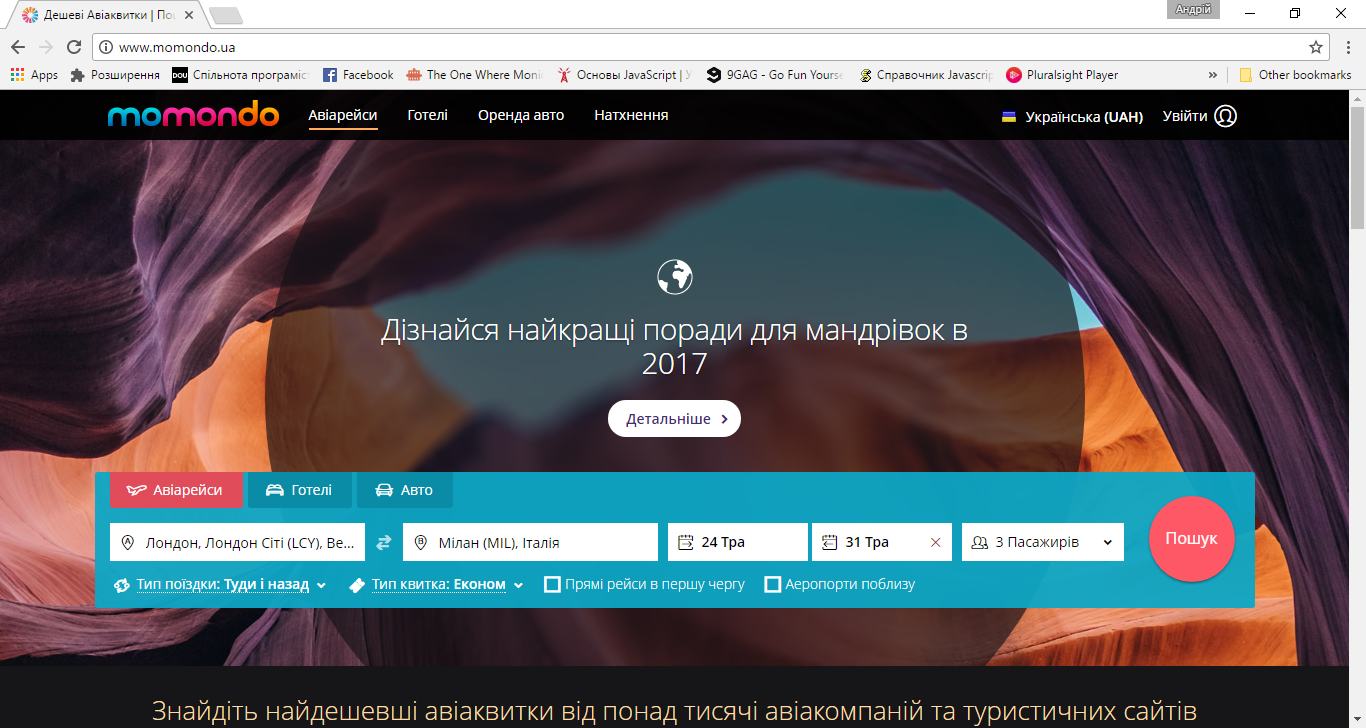
Переваги:

* Можливість додаткових послуг, таких як: бронювання готелів, виклик таксі та перегляд найблищих закладів харчування.
* Наявна українська локалізація
* Синхронізація із соціальними мережами
* Величезна кількість інформації про компанію та авіаперевезення.

Недоліки:

* Складний інтерфейс.
* Потребує багато кроків для купівлі або бронювання квитка.[

Momondo — сервіс для пошуку та замовлення авіаквитків

Рис. 4.1.2.3 Головна сервісу Momondo

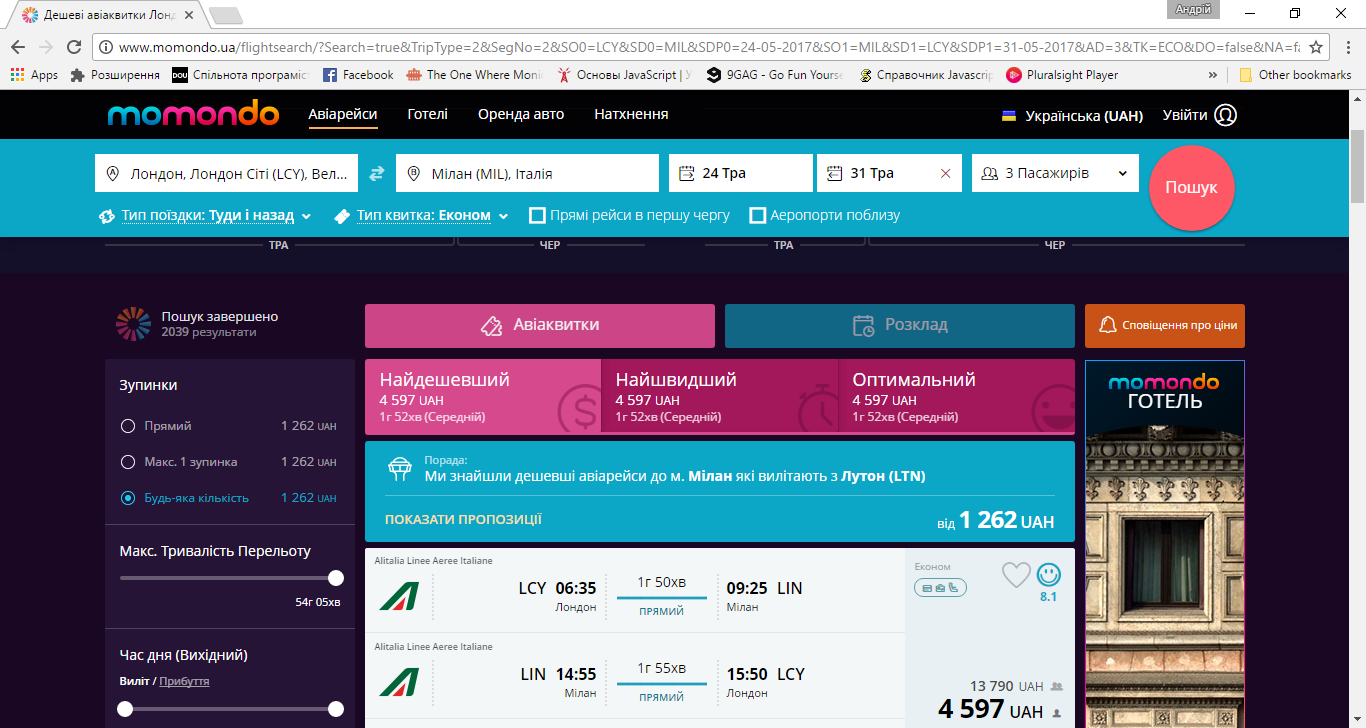


Рис. 4.1.2.4 Сторінка результату пошуку рейсів та подальшим замовленням сервісу Momondo

Переваги:

* Простота і зручність інтерфейсу
* Пропозиції найпопулярніших рейсів.
* Величезна кількість інформації про компанію та авіаперевезення.

Недоліки:

* Відслідковування статусу рейса.
* Присутня реклама

British Airways — найбільша авіакомпанія та національний авіаперевізник Великої Британії, одна з найбільших в Європі, [із вільни](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)м серверним [веб-застосунок](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA)ом для користування послугами компанії.

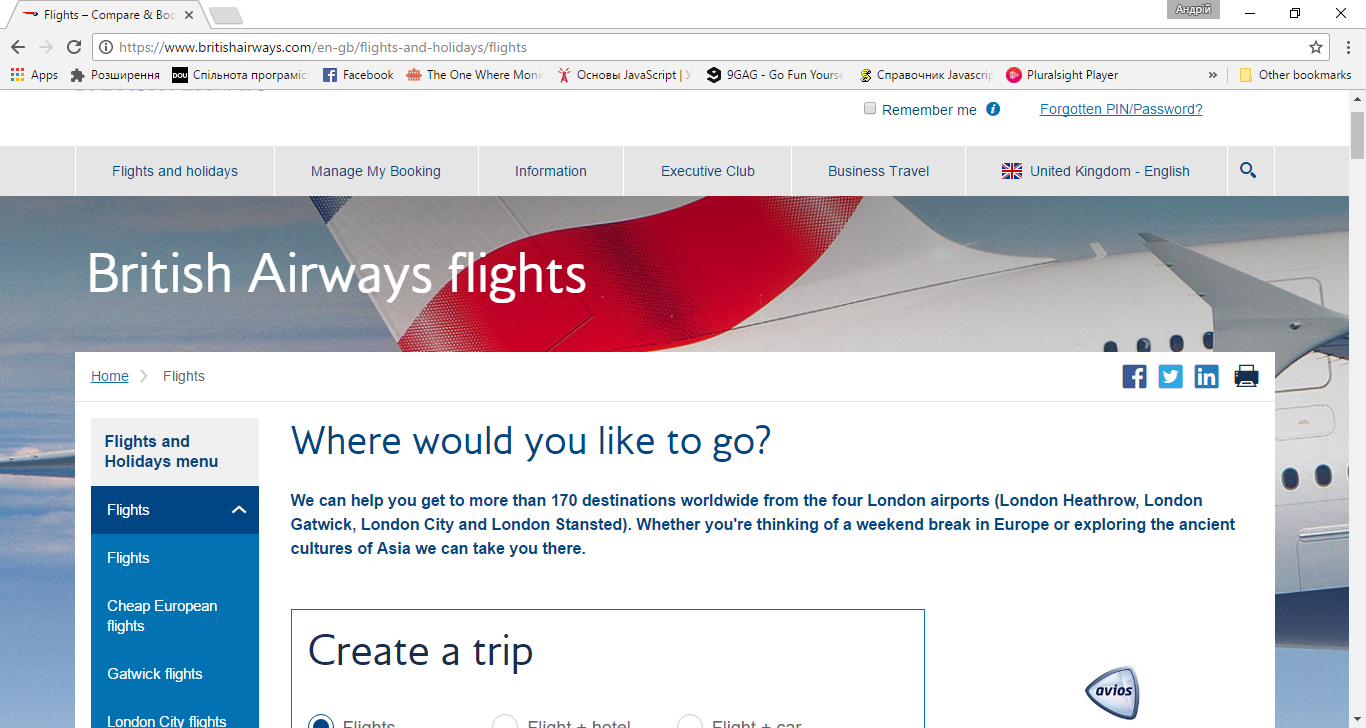


Рис. 4.1.2.5 Сторінка пошуку рейсів сервісу British Airways

Переваги:

* Простота і зручність інтерфейсу.
* Пропозиції найдешевших квитків.
* Відслідковування статусу рейса.
* Величезна кількість інформації про компанію та авіаперевезення.

Недоліки:

* Немає укрїнської локалізації

## 1.3. Висновок

В результаті огляду програмного забезпечення моніторингу та планування пасажирських авіаперевезень було проаналізовано предметну область. Описано головні тенденції та потреби. Також проаналізовано декілька продуктів аналогів. В результаті огляду аналогів було виявлено переваги та недоліки, які будуть братись до уваги при розробці програмного забезпечення.

# **Розділ 2. Постановка завдання для розробки програмного забезпечення для моніторингу та планування пасажирських авіаперевезень**

## 2.1. Загальна постановка задачі

Завданням бакалаврської кваліфікаційної роботи є створення програмного забезпечення для моніторингу та планування пасажирських авіаперевезень для авіакомпанії у вигляді веб сайту.

Цей сайт суттєво зекономить час користувачів, так як їм не прийдеться стояти у величезних чергах для замовлення квитків, а також не потрібно дзвонити у службу підтримки для отримання докладної інформації.

Основними можливостями веб сайту будуть:

* Пошук авіарейсів.
* Перегляд детальної інформації про авіарейс.
* Бронювання або замовлення квитка на вибраний рейс.
* Відслідковування статусу рейсу.
* Реєстрація та авторизація в системі.

Програмне забезпечення складатиметься з front-end(веб-браузер) та back-end(RESTfull API) частин.

Front-end частина буде використовувати основні веб-браузери такі як: Google Chrome, Mozilla Firefox, та наступні технології: HTML5\CSS3, Java Script та фреймворк AngularJS[1].

Back-end частина буде реалізована у вигляді RESTfull API, яке буде написано на об’єктно-орієнтованій мові C#. Для збереження даних слугуватиме MS SQL база даних.

## 2.2. Вибір архітектури

Для написання продукту буде використовуватись архітектура клієнт-сурвер.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

* набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
* набір клієнтів, які використовують сервіси, що надаються серверами;
* мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

## 2.3. Вибір програмних засобів

Для створення програмного продукту на серверній частині було обрано технологію .Net, середовище розробки Microsoft Visual Studio, мова C# та фреймворк ASP.NET Web API. Також для маніпуляцією з базою даних використовуватиметься Entity Framework. Для збереження даних слугуватиме MS SQL база даних.

На клієнтській було обрано такі технології як: HTML5\CSS3, JavaScript, Bootstrap[2], фреймворк AngularJS та середовище розробки Web Storm.

 .NET — програмна технологія, запропонована фірмою [Microsoft](https://uk.wikipedia.org/wiki/Microsoft) як платформа для створення як звичайних програм, так і [веб-застосунків](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA). Багато в чому є продовженням ідей та принципів, покладених в технологію [Java](https://uk.wikipedia.org/wiki/Java). Одною з ідей .NET є сумісність служб, написаних різними [мовами](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Хоча ця можливість рекламується Microsoft як перевага .NET, платформа Java має таку саму можливість.

Кожна [бібліотека](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC) (збірка) в .NET має свідчення про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок.

.NET — [крос-платформова](https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81-%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%BE%D0%B2%D1%96%D1%81%D1%82%D1%8C) технологія, в цей час існує реалізація для платформи [Microsoft Windows](https://uk.wikipedia.org/wiki/Microsoft_Windows), [FreeBSD](https://uk.wikipedia.org/wiki/FreeBSD) (від Microsoft) і варіант технології для [ОС](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [Linux](https://uk.wikipedia.org/wiki/GNU/Linux) в проекті [Mono](https://uk.wikipedia.org/wiki/Mono) (в рамках угоди між Microsoft з [Novell](https://uk.wikipedia.org/wiki/Novell)), DotGNU .

Захист авторських прав відноситься до створення середовищ виконання (CLR — [Common Language Runtime](https://uk.wikipedia.org/wiki/Common_Language_Runtime)) для програм .NET. Компілятори для .NET випускаються багатьма фірмами для різних мов вільно. [3-6]

JavaScript (JS) — динамічна, [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Реалізація стандарту [ECMAScript](https://uk.wikipedia.org/wiki/ECMAScript). Найчастіше використовується як частина [браузера](https://uk.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80), що надає можливість коду на стороні [клієнта](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, [асинхронно](https://uk.wikipedia.org/wiki/AJAX) обмінюватися даними з [сервером](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0), змінювати [структуру](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0) та [зовнішній вигляд](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD) [веб-сторінки](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0). Мова JavaScript також використовується для [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) на стороні сервера (подібно до таких мов програмування, як [Java](https://uk.wikipedia.org/wiki/Java) і [C#](https://uk.wikipedia.org/wiki/C_Sharp)), розробки [ігор](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%96_%D1%96%D0%B3%D1%80%D0%B8), стаціонарних та мобільних додатків, сценаріїв в прикладному [ПЗ](https://uk.wikipedia.org/wiki/%D0%9F%D0%97) (наприклад, в програмах зі складу [Adobe Creative Suite](https://uk.wikipedia.org/wiki/Adobe_Creative_Suite)), всередині [PDF](https://uk.wikipedia.org/wiki/PDF)-документів тощо.

JavaScript класифікують як [прототипну](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (підмножина [об'єктно-орієнтованої](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [скриптову мову](https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування ([імперативну](https://uk.wikipedia.org/wiki/%D0%86%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) та частково [функціональну](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і деякі відповідні архітектурні властивості, зокрема: [динамічна](https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F) та [слабка](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D0%BB%D0%B0%D0%B1%D0%BA%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F&action=edit&redlink=1) [типізація](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97), автоматичне керування пам'яттю, прототипне [наслідування](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D1%96%D0%B4%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [функції](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) як [об'єкти першого класу](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D1%88%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%83).[7]

## 2.4. Специфікація вимог до програмного продукту

* + 1. **Вступ**
       1. **Призначення, мета**

Призначення даного продукту – дати можливість користувачам моніторити та планувати авіарейси, тобто замовляти або бронювати квитки, а також відслідковувати статус рейсу.

Мета – надати користувачеві інструмент для пошуку та замовляти або бронювати квитків, а також відслідковувати статус рейсу.

.

* + 1. **Загальний опис**
       1. **Характеристики продукту**

Функції, що будуть реалізовані у програмі:

* Пошук авіарейсів.
* Перегляд детальної інформації про авіарейс.
* Бронювання або замовлення квитка на вибраний рейс.
* Редагування даних в особистому кабінеті.
* Відслідковування статусу рейсу.
* Реєстрація та авторизація в системі.



Рис. 2.4.1 UML діаграма варіантів використання

* + - 1. **Класи користувачів та їх характеристики**

|  |  |
| --- | --- |
| 1. Зареєстрований користувач – це користувач, який є зареєстрованим в системі. | * Пошук рейсів. * Перегляд інформації про рейс. * Замовлення або бронювання квитків. * Відслідковування статусу рейсів. * Зміна персональної інформації. |
| 1. Адміністратор – це користувач, який є зареєстрованим в системі та має найбільші права в системі | * Пошук рейсів. * Перегляд інформації про рейс. * Замовлення або бронювання квитків. * Відслідковування статусу рейсів. * Зміна персональної інформації. * Відслідковування та зміна статусу рейсів. * Редагування рейсів. |
| 1. Незареєстрований користувач – це користувач, який не є зареєстрованим в системі. | * Реєстрація в системі. * Пошук рейсів. * Перегляд інформації про рейс. * Відслідковування статусу рейсів. |

* + - 1. **Середовище функціонування**

Програмний продукт передбачає такі апаратні та програмні вимоги до пристрою:

* Браузери: Google Chrome, Mozilla Firefox;
* Наявність підключення до мережі Internet.
  + 1. **Характеристики системи**
       1. **Пошук рейсів**
          1. Опис і пріоритет

Відображення списку за вибраними параметрами(дата, пункт відправлення, пункт прибуття і т.д.) для подальшого замовлення квитка.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук
  1. Відкриття головної сторінки.
  2. Користувач вводить дані для пошуку рейсів та натискає кнопку «Пошук».
  3. Перехід на сторінку з завантаженим списком усіх можливих рейсів для заданих параметрів.
     + - 1. Функціональні вимоги

REQ-1: Завантаження списку доступних рейсів.

REQ-2: Усі поля повинні бути заповнені.

* + - 1. **Пререгляд детальної інформації про рейс**
         1. Опис і пріоритет

Після пошуку рейсів користувач вибирає конкретний рейс для перегляду додаткової інформації.

Пріоритет середній.

* + - * 1. Послідовності дія/відгук

1. Після пошуку рейсів кориситувач вибирає бажаний рейс клікнувши на посилання «Детальніше».
2. Користувач переглядає детальну інформацію
   * + 1. **Реєстрація в системі**
          1. Опис і пріоритет

Користувач реєструється у системі для отримання додаткової функціональності.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Перехід на сторінку реєстрації клікнувши на посилання «Реєстрація» у верхній панелі.
2. Користувач заповнює усі поля.
3. Програма перевіряє введені дані.
4. У разі неправильних даних виводиться відповідне повідомлення.
5. Система відображає результат реєстрації.
   * + - 1. Функціональні вимоги

REQ-1: Усі обов’язкові поля повинні бути заповненні.

* + - 1. **Вхід в систему**
         1. Опис і пріоритет

Вхід користувача в систему та надання відповідних прав.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Перехід на сторінку реєстрації клікнувши на посилання «Вхід» у верхній панелі.
2. Користувач вводить email та пароль.
3. Користувач натискає кнопку «Вхід»
4. Система повідомляє про результат входу.
   * + - 1. Функціональні вимоги

REQ-1: Усі поля повинні бути заповнені.

* + - 1. **Редагування особистого кабінету**
         1. Опис і пріоритет

Користувач має можливість редагувати особисті дані.

Пріоритет низький.

* + - * 1. Послідовності дія/відгук

1. Користувач повинен увійти в систему та перейти в особистий кабінет клікнувши на посилання «Особистий кабінет»
2. На сторінці особистого кабінету натиснути на кнопку «Налаштування».
3. Зявляється додаткове вікно, де користувач змінює дані та натискає на кнопку «Змінити».
4. Зявляється повідомлення про те що дані змінено успішно.
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен бути авторизований.

REQ-2. Введені дані мають бути конкретного типу.

REQ-3. Можливість скасування операції.

* + - 1. **Замовлення або бронювання квитка**
         1. Опис і пріоритет

Користувач замовляє або бронює квиток.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Здійснивши пошук та вибравши бажаний рейс користувач натискає на кнопку замовити.
2. Користувач заповнює усі поля.
3. Також вибирає місця натиснувши на кнопку «Обрати місце».
4. Вибравши місце напискає кнопку «Гаразд».
5. Користувач натискає на кнопку «Продовжити».
6. Зявляється повторна інформація про замовлення та в залежності від вибраного методу реєстрації(Онлайн/Аеропорт) зявляється або форма заповнення кредитної картки(Онлайн метод) або кнопка «Забронювати»
7. Користувач заповняє форму з кредитною карткою і натискає кнопку «Замовити»(для онлайн режиму) або натискає кнопку «Забронювати»(для аеропорт режиму).
8. Зявляється повідомлення про успішно виконане замовлення.
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен бути авторизований.

REQ-2. Введені дані мають бути конкретного типу.

REQ-3. Можливість скасування операції.

* + - 1. **Статус рейсу**
         1. Опис і пріоритет

Користувач має можливість переглянути статус рейсу

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Користувач переходжить на сторінку «Статус рейсу», натиснувши на посилання «Статус рейсу» у верхній панелі
2. Користувач вибирає пункт відправлення/прибуття та дату і натискає на кнопку «Пошук».
3. Користувач відслідковує статус рейсу.
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен вибрати/заповнити всі поля.

* + - 1. **Список рейсів**
         1. Опис і пріоритет

Користувач може переглядати список усіх рейсів, а також фільтрувати їх.

Пріоритет середній.

* + - * 1. Послідовності дія/відгук

1. Користувач переходжить на сторінку «Список рейсів», натиснувши на посилання «Список рейсів» у верхній панелі.
2. Користувач переглядає список усіх можливих рейсів.
3. Користувач використовує фільтри та пошук для знаходження потрібного рейсу.
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен вибрати/заповнити всі поля.

* + - 1. **Список замовлень**
         1. Опис і пріоритет

Користувач може переглядати список своїх замовлень, а також фільтрувати їх.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Користувач переходжить на сторінку «Список замовлень», натиснувши на посилання «Список замовлень» у верхній панелі.
2. Користувач переглядає список своїх замовлень.
3. Користувач використовує фільтри та пошук для знаходження потрібного замовлення.
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен бути авторизованим у системі.

* + - 1. **Пререгляд детальної інформації про замовлення**
         1. Опис і пріоритет

Після перегляду замовлень користувач вибирає бажане замовлення для перегляду детальної інформації про замовлення.

Пріоритет високий.

* + - * 1. Послідовності дія/відгук

1. Після пошуку рейсів кориситувач вибирає бажаний замовлення клікнувши на посилання «Детальніше».
2. Користувач переглядає детальну інформацію
   * + - 1. Функціональні вимоги

REQ-1. Користувач повинен бути авторизованим у системі.

* + 1. **Вимоги зовнішніх інтерфейсів**
       1. **Користувацькі інтерфейси**

Інтерфейс програми повинен бути простим для розуміння, забезпечувати швидкодію відповідно до вимог та дозволяти зручно виконувати необхідні операції.

* + - 1. **Апаратні інтерфейси**

Апаратні інтерфейси не будуть використовуватися цією програмною системою.

* + - 1. **Програмні інтерфейси**
* Visual Studio
* Entity Framework
* Web Storm
* MS SQL Server
  + - 1. **Комунікаційні інтерфейси**

Клієнтська частина системи буде спілкуватися з серверною за допомогою протоколу HTTP.

* + 1. **Інші нефункціональні вимоги**
       1. **Вимоги продуктивності**

Швидкість завантаження та опрацювання даних буде прямо залежати від швидкості передачі даних в мережі та від кількості даних, які отримуються з мережі.

* + - 1. **Вимоги безпеки**

Програмний продукт немає вимог безпеки.

* + - 1. **Атрибути якості програмного продукту**
* Зручність використання.
* Надійність.
* Супроводжуваність.
  + - 1. **Інші вимоги**

Українська локалізація.

# **Розділ 3. Архітектура і проектування програмного забезпечення для моніторингу та планування пасажирських авіаперевезень**

**3.1 Архітектура програмного продукту. Опис впровадження клієнт серверної архітектури, а також MVC шаблону в програмному продукті**

Програмний продукт побудований на використанні клієнт серверної архітектури, тобто існує окремо серверна частина(RESTfull API) та клієнтська частина.

Архітектура клієнт – сервер (client-server architecture) – це концепція інформаційної мережі, в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів. Розглянута архітектура визначає два типи компонентів: сервери і клієнти.

 Сервер – це об’єкт, що дає сервіс іншим об’єктам мережі за їх запитами. Сервіс – це процес обслуговування клієнтів. Сервер працює за завданнями клієнтів і управляє виконанням їх завдань. Після виконання кожного завдання сервер посилає отримані результати клієнту, який послав це завдання. Сервісна функція в архітектурі клієнт – сервер описується комплексом прикладних програм, відповідно до якого виконуються різноманітні прикладні процеси.

Процес, який викликає сервісну функцію за допомогою певних операцій, називається клієнтом. Ним може бути програма або користувач. Клієнти – це робочі станції, які використовують ресурси сервера і надають зручні інтерфейси користувача. Інтерфейси користувача це процедури взаємодії користувача з системою або мережею.

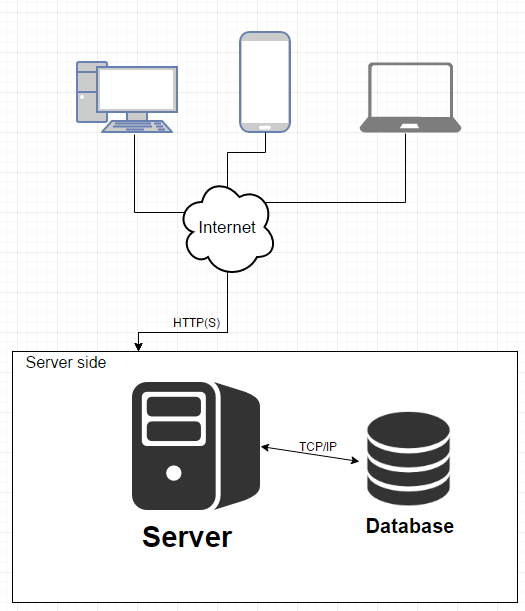


Рис. 3.1.1 Приклад клієнт серверної архітектури

Северна та клієнтська (фреймворк AngularJS) частини спроектовані за шаблоном MVC (Model-View-Controller).

Model-View-Controller (MVC) шаблон відокремлює [модель даних](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), вигляд ([інтерфейс користувача](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0)) та модуль керування, застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Model - представляє дані та реагує на команди контролера, змінюючи свій стан.

View - керує відображенням інформації.

Controller - інтерпретує дії користувача, повідомляючи модель про необхідність змінитись.

Контролер Web Api не працює з відображеннями, замість цього він приймає запити від клієнта та відсилає відповідь через HTTP протокол.

**3.2. Проектування компонент**

При проектуванні серверної частини продукту було вирішено створити три проекти у Visual Studio які взаємодіють між собою:

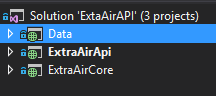


Рис. 3.2.1 Список пакетів на серверній частині

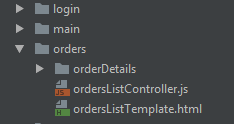
Data – у цьому проекті буде зосереджено роботу на маніпулювання з базою даних, усі запити до бази даних будуть відбуватись у цьому проекті.

ExtraAirCore – цей проект містить усі моделі та інтерфейси за якими відбувається маніпуляції з базою даних.

ExtraAirApi – це основний проект який об’єднює в собі два попередні та обробляє запити від клієнта, містить в собі всі додаткові збірки, бібліотеки та утиліти.

Цей підхід дозволяє зберегти код більш чистішим та зрозумілішим при зростанню основного проекта, також застосовується IoC-контейнер.

На клієнтській частині було вирішено кожен новий модуль створювати як новий пакет який містить: Controller, View та допоміжні файли.

  
Рис. 3.2.2 Вигляд пакетів на клієнтській частині

**3.3 Проектування класів**



Рис. 3.3.1 Діаграма класів

В результаті аналізу предметної області, визначено наступні класи:

User – клас в якому міститься інформація основна інформація користувача: імя, прізвище, email, пфроль, дата народження, видалений, адреса та роль.



Рис. 3.3.2 Клас User

Client – клас який наслідується від User та містить: список замовлень, список відгуків, спосок кредитних карт, спосок історій пошуків.



Рис. 3.3.3 Клас Client

Address – клас в якому міститься інформація про адресу(користувача та аеропорта) та мітить такі поля: країна, місто, вулиця, номер будинку, поштовий індекс



Рис. 3.3.4 Клас Address

Airport – клас який містить інформацію про аеропорт, а саме: імя, адресу та посилання на проміжний клас tourToAiprort(зв’язок багато до багатьох).



Рис. 3.3.5 Клас Airport

TourToAirport – проміжний клас який об’єднює Tour та Airpor та містить: час віправлення, час прибуття, чи це проміжний рейс, та посилання на рейс та аеропорт.



Рис. 3.3.6 Клас TourToAirport

Tour – клас який містить інформація про рейс, а саме: час віправлення, час прибуття, ціна, літак, поточна кількість пасажирів, список пасажирів, список замовлень та посилання на tourToAirport



Рис. 3.3.7 клас Tour

Order – клас який містить інформацію про замовлення, а саме: дату замовлення, користувача, список рейсів, список пасажирів, ціна, чи замовлення оплачено, дата відправлення, дата прибуття.



Рис. 3.3.8 Клас Order

Plane – клас який несе інформацію про літак, назва, максимальна кількість пасажирів, рівні кофморту



Рис. 3.3.9 Клас Plane

В даному проекті використовується підхід Code First отже за цими класами було створено базуданих і звязки між сутностями.

## Прототипування інтерфесу користувача

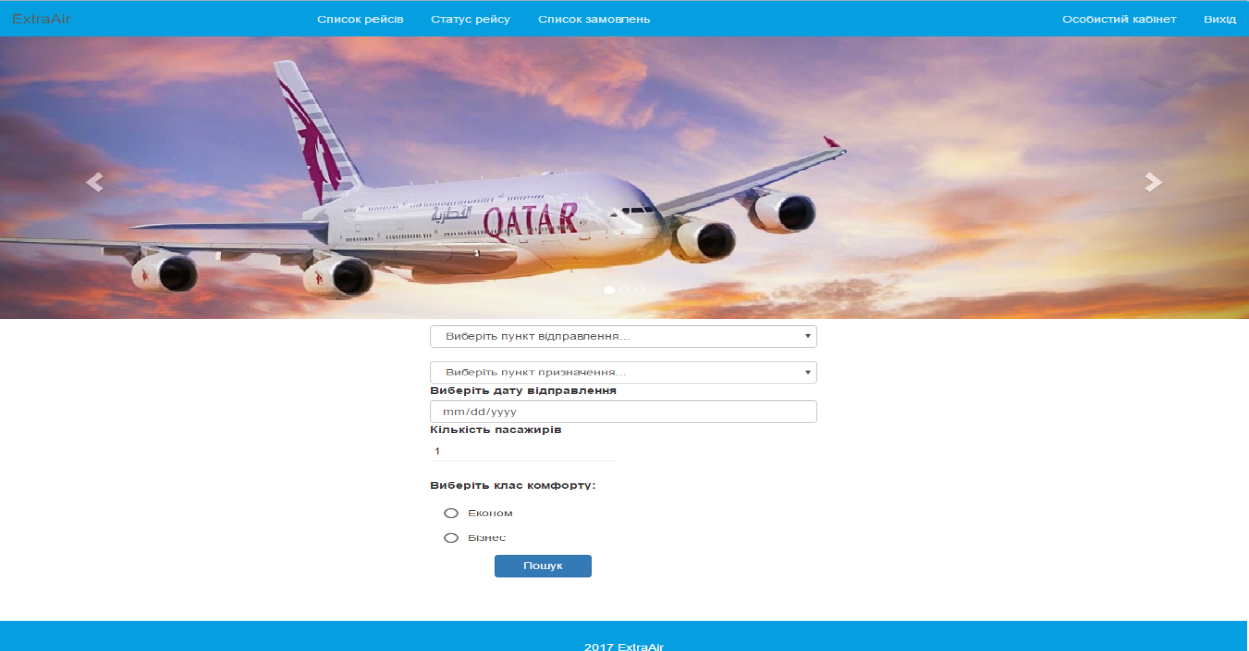


Рис. 3.4.1 Домашня сторінка додатку

**Опис:** при першому запуску програми відображається головне вікно, на якому користувач може здійснити пошук рейсів, а також перейти на інші вкладки програми.

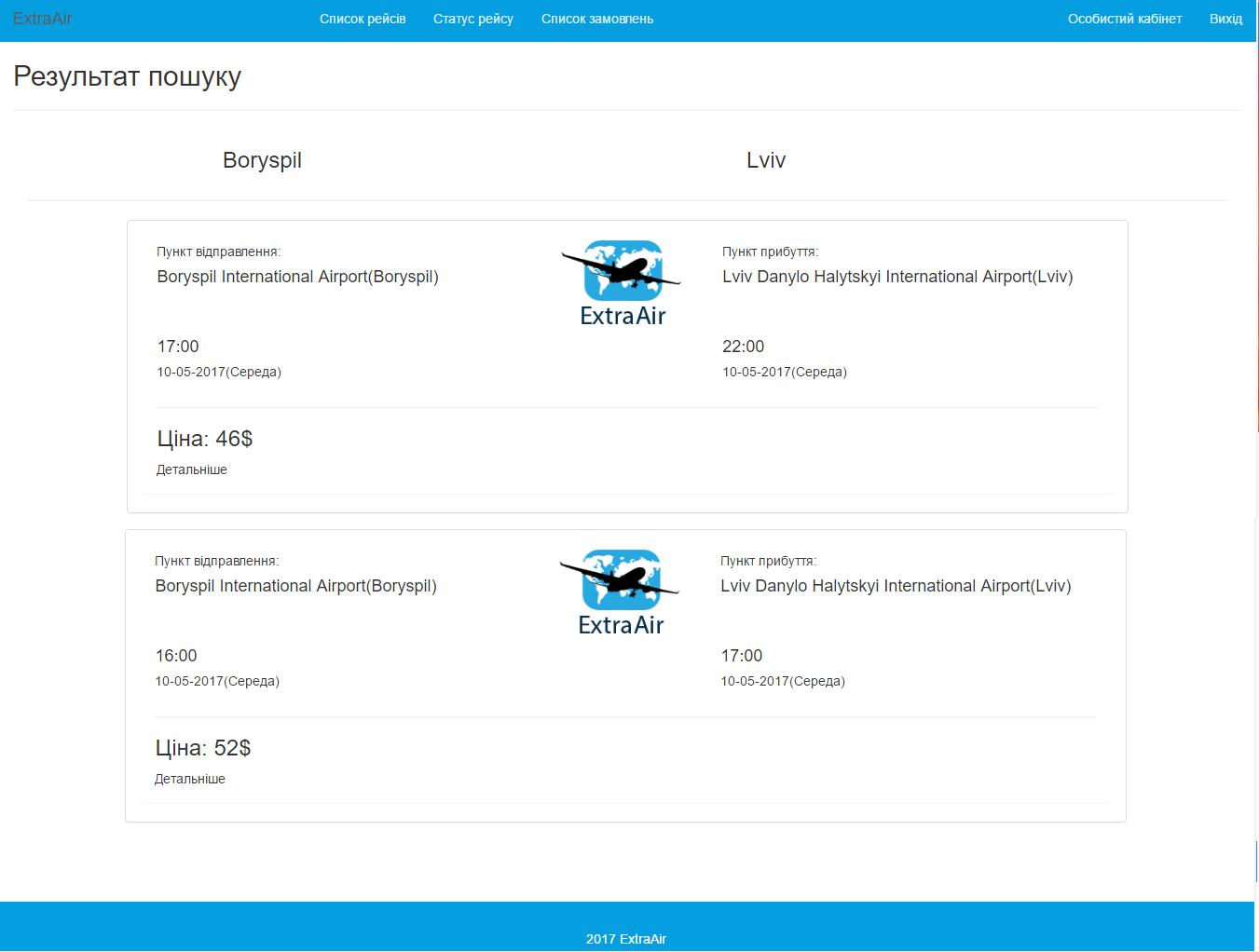
****

Рис. 3.4.2 Сторінка результату пошуку рейсів

**Опис:** на цій сторінці можна вибрати рейс для детальної інформації та подальшого замовлення.

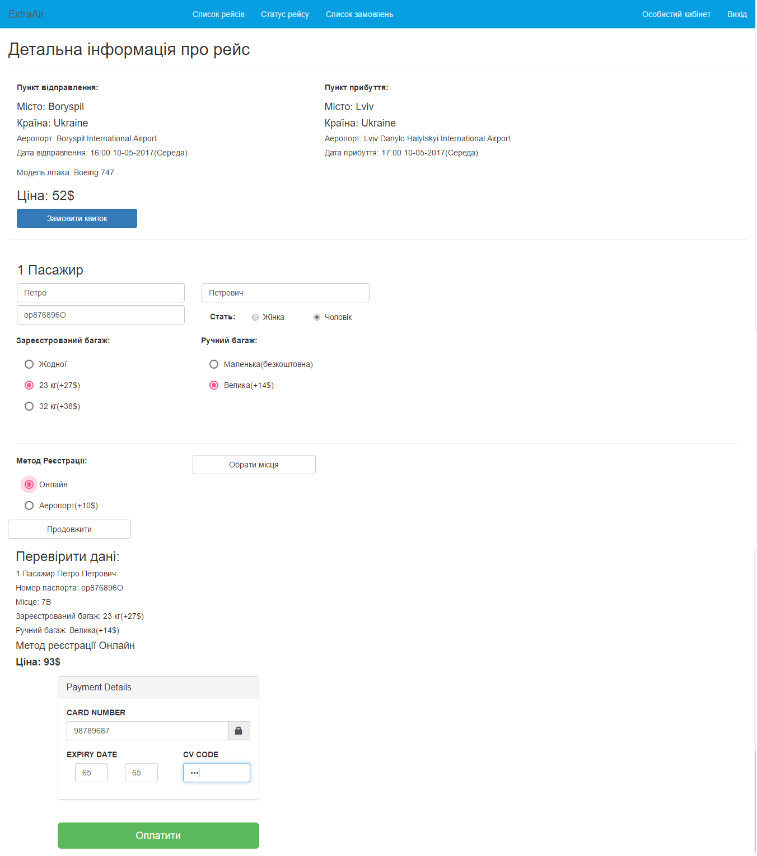


Рис. 3.4.3 Сторінка замовлення квитка

**Опис:** на сторінці замовлення квитка потрібно заповнити усі поля вводу та натискати відповідні клавіші.

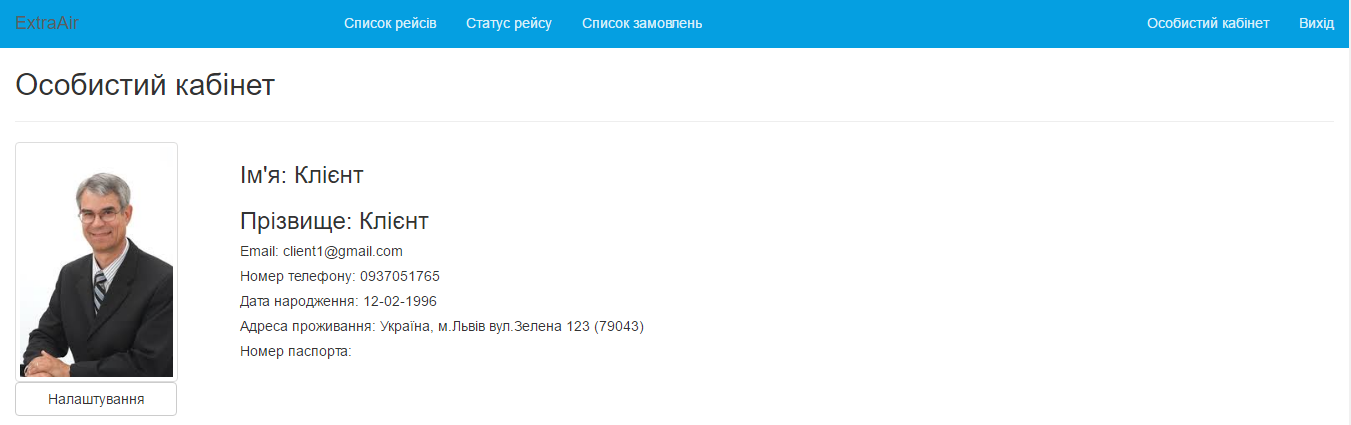


Рис. 3.4.4 Сторінка особистого кабінету.

**Опис:** на цій сторінці користувач може перегляну особисту інформацію, а також редагувати її.



Рис. 3.4.5 Модальне вікно редагування особистого кабінету.

**Опис:** на цьому вікні користувач може редагувати особисту інформацію.

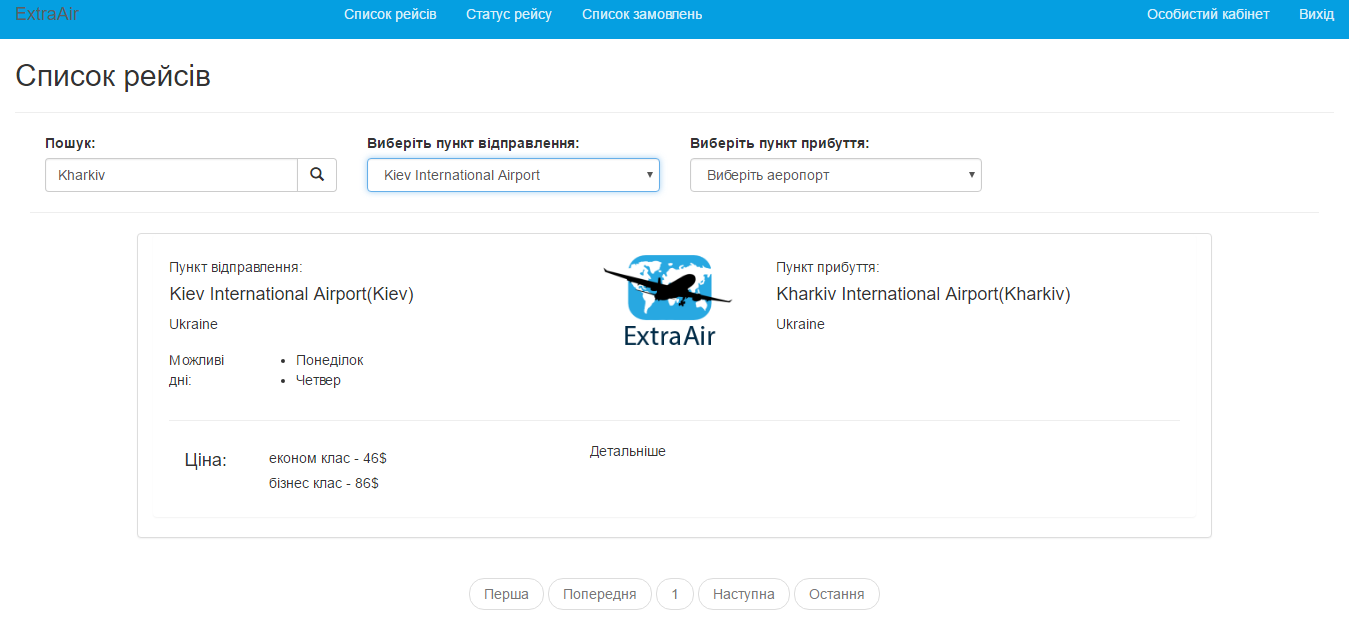


Рис. 3.4.6 Сторінка «Список рейсів»

**Опис:** на даній сторінці буде можливість переглянути повний список рейсів, а таож їх фільтрація та пошук.

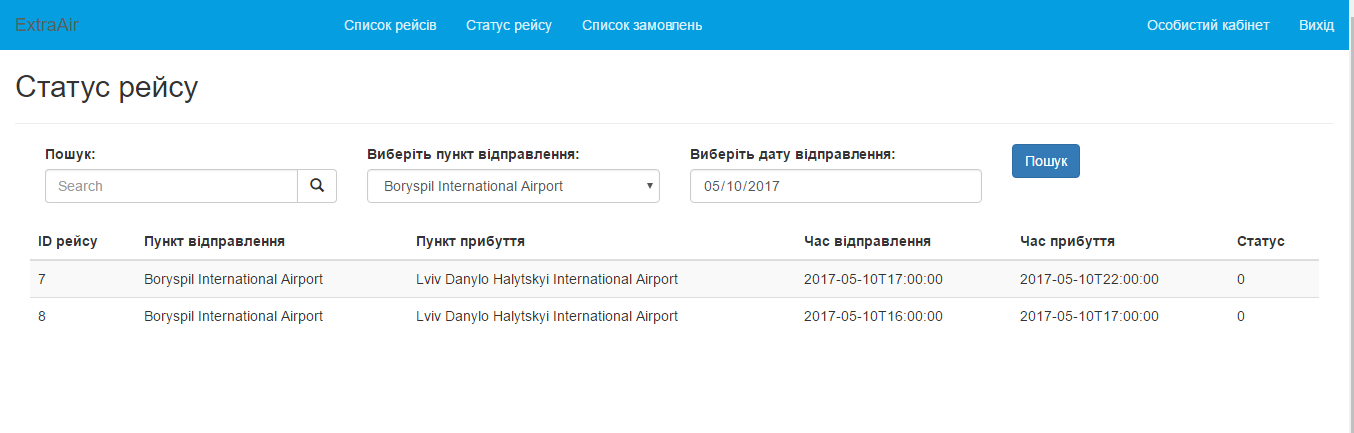


Рис. 3.4.7 Сторінка «Статус рейсу»

**Опис:** на цій сторінці користувач може відслідковувати старус рейсів

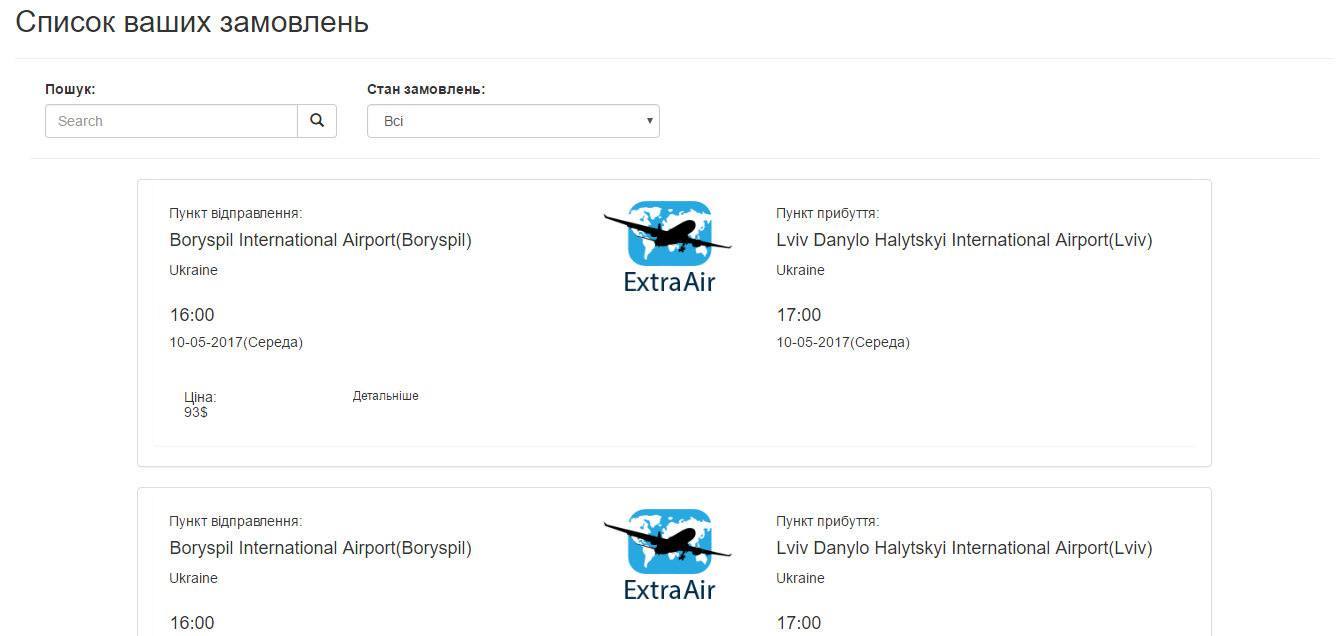


Рис. 3.4.8 Сторінка «Список ваших замовлень»

**Опис:** користувач буде мати змогу переглядати список усіх своїх замовлень.

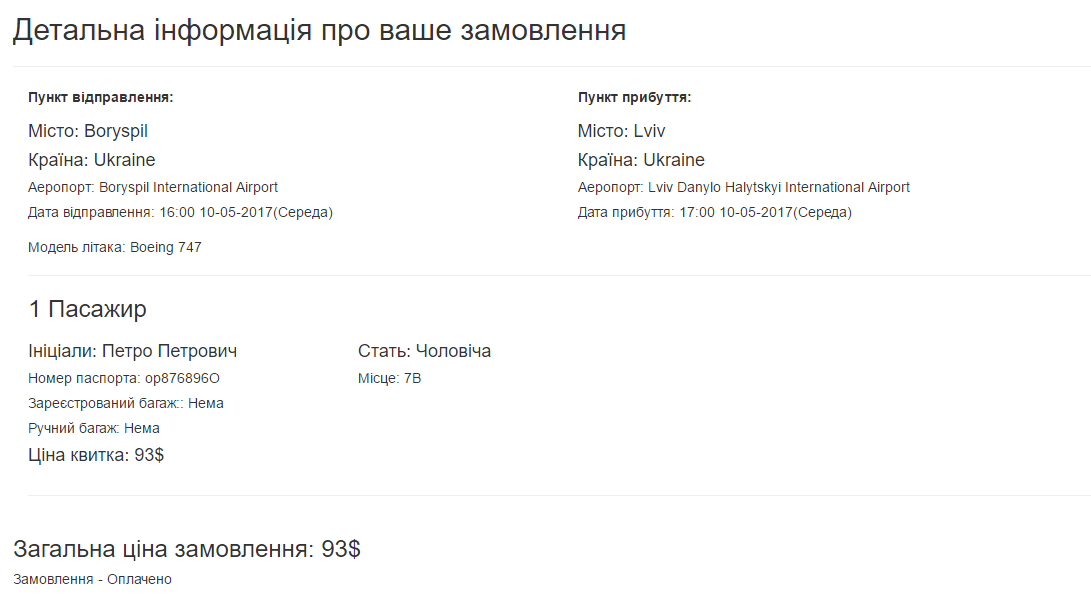


Рис. 3.4.9 Сторінка детальної інформації про конкретне замовлення

**Опис:** на сторінці користувач буде мати змогу переглянути детальну інформацію про своє замовлення.

## Висновки про отримані під час практики результати

Пройшовши практику на Edvantis я отримав безцінний досвід роботи в команді, побачив життя компанії зсередини. Я отримав багато практичних та теоретичних знань. Покращив знання мови C# та фреймворка AngularJS. Ці здобуті знання я втілив у життя розробивши частину програмного забезпечення для моніторингу та планування пасажирських авіаперевезень.

Також була проведена робота на тему моєї бакалаврської кваліфікаційної роботи – моніторинг та планування пасажирських авіаперевезень. В результаті цієї роботи було сформовано 2 розділи (огляд предметної області та продуктів-аналогів, специфікація вимог та характеристика системи) дипломної роботи та частина 3-го розділу (UML-діаграми).

# СПИСОК ОПРАЦЬОВАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. AngularJS [Електронний ресурс] : – Режим доступу: <https://docs.angularjs.org/api>
2. WebApi [Електронний ресурс] : – Режим доступу: <https://www.asp.net/web-api>
3. Entity Framework [Електронний ресурс] : – Режим доступу : <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro>
4. Документація по мові C# [Електронний ресурс] : – <https://docs.microsoft.com/en-us/dotnet/articles/csharp/csharp>
5. Документація по середовишу розробки Visual Studio [Електронний ресурс] : – <https://www.visualstudio.com/en-us/docs/vs/overview>
6. Документація по JavaScript [Електронний ресурс] : – Режим доступу : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
7. Документація по Bootstrap [Електронний ресурс] : – Режим доступу : <https://www.w3schools.com/bootstrap>
8. Документація по HTML5 [Електронний ресурс] : – Режим доступу : <https://www.w3schools.com/html/default.asp>
9. Документація по середовишу розробки Web Storm [Електронний ресурс] : – Режим доступу : <https://www.jetbrains.com/webstorm/documentation/>
10. Документація по CSS3 [Електронний ресурс] : – Режим доступу : <https://www.w3schools.com/css/default.asp>
11. Сайт аналог авіакомпанії WizzAir [Електронний ресурс] : – Режим доступу : <https://wizzair.com/uk-ua/main-page#/>
12. Сайт аналог авіакомпанії Momondo [Електронний ресурс] : – Режим доступу : <http://www.momondo.ua/>

Додаток А. Вихідний код розробленого програмного продукту для моніторингу та планування пасажирських авіаперевезень.

'use strict';  
  
**var** app = angular.*module*('extraAir');  
app.controller('toursListController', **function** ($rootScope, $scope, $location, $window, $filter, getService, paginationArrayService, paginationService, airportsService) {  
   
 $scope.isLoading = **true**;  
  
 initPaginationParams();  
   
 airportsService.getAirports().then(**function**(data){  
 $scope.airports = data;  
 });  
  
 **var** utils = **new** CommonUtils();  
  
 $scope.$on('$locationChangeStart', **function**(e) {  
 $scope.isLoading = **true**;  
 initPaginationParams();  
 // paginationService.ChangeURL($scope.loadList, $scope.tours, $rootScope.preArray, '/toursList', $rootScope.pagingInfo);  
 });  
  
   
 $scope.tours = [];  
 $scope.loadList = **function**(){  
 $scope.isLoading = **true**;  
 **var** URL = "api/tours?page=" + $rootScope.pagingInfo.page + "&itemsPerPage=" + $rootScope.pagingInfo.itemsPerPage +  
 "&search=" + $rootScope.pagingInfo.search + "&airportFromId=" + $rootScope.pagingInfo.airportFromId +  
 "&airportToId=" + $rootScope.pagingInfo.airportToId + "&day=" + $rootScope.pagingInfo.day;  
   
 getService.GetObjects(URL).then(**function** (data) {  
 $scope.tours = data.data.list;  
 $scope.AnyElementOfList = data.data.count == 0;  
 $rootScope.pagingInfo.totalItems = data.data.count;  
 $scope.pages = Math.ceil(data.data.count / $rootScope.pagingInfo.itemsPerPage);  
 $scope.paginArray = paginationArrayService.Array($scope.pages, $rootScope.pagingInfo.page);  
 correctData();  
   
 $scope.paginationScope = {  
 data: !!$rootScope.pagingInfo ? {} : $rootScope.pagingInfo,  
 list: $scope.paginArray  
 };  
  
 **if** (!$scope.$$phase) {  
 $scope.$apply();  
 }  
   
 $scope.isLoading = **false** }, **function** (error) { }).finally(**function**(){  
 $scope.isLoading = **false**;  
 });  
  
 $location.search('search', !!$rootScope.pagingInfo.search ? $rootScope.pagingInfo.search : **null**);  
 $location.search('page', $rootScope.pagingInfo.page);  
 $location.search('airportFromId', $rootScope.pagingInfo.airportFromId);  
 $location.search('airportToId', $rootScope.pagingInfo.airportToId);  
 $location.search('day', !!$rootScope.pagingInfo.day ? $rootScope.pagingInfo.day : **null**);  
  
  
 $rootScope.preArray = $scope.tours;  
 };  
  
 $scope.loadList();  
  
  
 **function** initPaginationParams(){  
 $rootScope.pagingInfo = {  
 page: $location.search().page !== undefined ? $location.search().page : 1,  
 itemsPerPage: 5,  
 search: $location.search().search !== undefined ? $location.search().search : '',  
 airportFromId: $location.search().airportFromId !== undefined ? $location.search().airportFromId : **null**,  
 airportToId: $location.search().airportToId !== undefined ? $location.search().airportToId : **null**,  
 day: $location.search().day !== undefined ? $location.search().day : ''  
 };  
 }  
   
 **function** correctData() {  
 **for** (**var** i = 0; i < $scope.tours.length; i++) {  
 $scope.tours[i].timeStart = $filter('date')($scope.tours[i].DateStart, 'HH:mm');  
 $scope.tours[i].timeFinish = $filter('date')($scope.tours[i].DateFinish, 'HH:mm');  
 $scope.tours[i].economyPrice = ($scope.tours[i].Price \* Constants.PRICE\_COEF.ECONOMY).toFixed(0);  
 $scope.tours[i].businessPrice = ($scope.tours[i].Price \* Constants.PRICE\_COEF.BUSSINESS).toFixed(0);  
 **for** (**var** j = 0; j < $scope.tours[i].PossibleDays.length; j++)  
 {  
 $scope.tours[i].PossibleDays[j] = utils.translateDays($scope.tours[i].PossibleDays[j]);  
 }  
 }  
 }  
});

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

using System.Linq;

using System.Net;

using System.Web.Http;

using System.Web.Http.Description;

using ExtraAirApi.Utils.Ninject;

using ExtraAirCore.API\_DTOs.Helper\_DTOs;

using ExtraAirCore.Command.Tour;

using ExtraAirCore.Models.EFContex;

using ExtraAirCore.Models.EFModels;

namespace ExtraAirApi.Controllers

{

[RoutePrefix("api/tours")]

public class ToursController : ApiController

{

private ExtraAirContext db = new ExtraAirContext();

// GET: api/Tours

[Route("")]

public object GetTours([FromUri]int page = 1, [FromUri]int itemsPerPage = 15, [FromUri]string search = null, [FromUri]int? airportFromId = null, [FromUri]int? airportToId = null,

[FromUri]string day = null)

{

var list = IoC.Get<IGetTours>().GetAllTours();

return IoC.Get<IGetTours>().GetToursWithPaginFiltering(new PaginFilteringHelper

{

Day = day,

AirportToId = airportToId,

AirportFromId = airportFromId,

ItemsPerPage = itemsPerPage,

Page = page,

Search = search

}, list);

}

[HttpGet]

[Route("bysearch")]

public object GetToursBySearch([FromUri]int airportFromId, [FromUri]int airportToId, [FromUri]string dayStart)

{

return IoC.Get<IGetTours>().GetToursBySearch(new TourSearchHelperDto

{

AirportFormId = airportFromId,

AirportToId = airportToId,

DayStart = dayStart

});

}

// GET: api/Tours/5

[ResponseType(typeof(Tour))]

public IHttpActionResult GetTour(int id)

{

var tour = IoC.Get<IGetTours>().GetTourById(id);

return Ok(tour);

}

// PUT: api/Tours/5

[ResponseType(typeof(void))]

public IHttpActionResult PutTour(int id, Tour tour)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != tour.TourId)

{

return BadRequest();

}

db.Entry(tour).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!TourExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return StatusCode(HttpStatusCode.NoContent);

}

[ResponseType(typeof(Tour))]

public IHttpActionResult PostTour(Tour tour)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

db.Tours.Add(tour);

db.SaveChanges();

return CreatedAtRoute("DefaultApi", new { id = tour.TourId }, tour);

}

[ResponseType(typeof(Tour))]

public IHttpActionResult DeleteTour(int id)

{

Tour tour = db.Tours.Find(id);

if (tour == null)

{

return NotFound();

}

db.Tours.Remove(tour);

db.SaveChanges();

return Ok(tour);

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

private bool TourExists(int id)

{

return db.Tours.Count(e => e.TourId == id) > 0;

}

}

}