НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматики та управляння в технічних системах

«Облік польотів»

Курсова робота

З дисципліни «Об’єктно-орієнтовне програмування»

|  |  |  |
| --- | --- | --- |
| Керівник  Хмелюк M.С  «Допущений до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Особистий підпис керівника)  « » \_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.  Захищений з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оцінка)  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Особистий підпис)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Особистий підпис) |  | Виконавець  ст. Овсеєнко В.К.  зал. Книжка № ЗПІ-зп6123  гр. ЗПІ-зп 61  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Особистий підпис виконавця)  « » \_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­\_\_\_\_\_\_\_\_\_  (Розшифровка підпису)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Розшифровка підпису) |

Київ - 2018

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра АВТОМАТИКИ ТА УПРАВЛЯННЯ В ТЕХНІЧНИХ СИСТЕМАХ

Дисципліна «Об’єктно-орієнтоване програмування»

Курс 3 Група ЗПІ-зп61 Семестр 1

**ЗАВДАННЯ**

**на курсову роботу студента**

|  |
| --- |
| Овсеєнко В.К. |
| (прізвище, ім’я, по батькові) |

|  |  |
| --- | --- |
| 1. Тема роботи | «Облік польотів» |
|  | |
|  | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи |  |

|  |
| --- |
| 3. Вихідні дані до роботи: |
| Реалізація програми в програмі Visual Studio 2017 з використанням фреймворку ASP.NET MVC 5 |
|  |
|  |

|  |
| --- |
| 4. Зміст розрахунково-пояснювальної записку (перелік питань, які підлягають розробці) |
| *1.Вступ. 2. Постановка задачі. 3. Огляд існуючих рішень та опис моделі.* |
| *4. Режим роботи та опис діалогу. 5. Опис програми. 6. Інструкція програміста.* |
| *7. Керівництво користувача 8. Висновки.* |
| *9. Список використаних джерел. 10. Додаток* |

|  |
| --- |
| 5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) |
| Діаграма класів, діаграма зв’язки структури |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 6. Дата видачі завдання | 14 10 2017 |

**Анотація**

ЗМІСТ

ВСТУП

Дана курсова робота описує структуру та методи програми, яка є частиною веб-додатку. Веб-додаток являє собою невеликий веб-сайт, на якому користувач має змогу здійснити пошук рейсових польотів між різними країнами. На сайті користувачеві немає необхідності здійснювати реєстрації, він одразу має змогу знайти інформацію щодо того чи іншого польоту, а також здійснити реєстрацію на бажаний рейс.

Серед основних функцій програми – це пошук у базі даних інформації щодо країн, міст, рейсів, а також пасажирів та екіпажу того чи іншого рейсу. Програма написана за допомогою сучасного фреймворку ASP.NET MVC 5 на мові програмування C#. Даний фреймворк дуже популярний та широко використовується у світі для створення веб-додатків та мобільних додатків. Для написання та компіляції програми ми використовували програмне забезпечення від компанії Microsoft Visual Studio 2017.

Станом на сьогодні C# визначено флагманською мовою корпорації Microsoft, бо вона найповніше використовує нові можливості .NET. Решта мов програмування, хоч і підтримуються, але визнані такими, що мають спадкові прогалини щодо використання .NET. Символ # у назві мови можна інтерпретувати і як дві пари плюсів ++, що натякають на новий крок у розвитку мови порівняно з C++ (подібно до кроку від C до C++), і як музичний символ дієз, разом з буквою C, що становить в англійській мові назву ноти до-дієз. Останнє й дало назву мові. Попри те, що символ # (октоторп) насправді є символом для позначення номера на більшості клавіатур і відрізняється від символу дієз ♯ (Unicode U+266F), Microsoft, як автор мови, неодноразово зверталася до своїх клієнтів з проханням прийняти таку стилізацію. C# розроблялась як мова програмування прикладного рівня для 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.

Титульним компілятором C# є Microsoft Visual C#. Існують інші компілятори C#, часто вони включають реалізації Common Language Infrastructure і бібліотеки класів .NET.

*1 Постановка* *задачі*

Задача полягає у створенні солюшена з 2 взаємозв'язаними проектами, а також створенні 5 видів (views) та навігації між ними та головного меню. Необхідно також здійснити написання класів згідно теми з властивостями на методами. Серед вимог до роботи також: організація об'єктної моделі, реалізація логіки заведення та відображення даних, трьохрівневе наслідування, інкапсуляція, поліморфізм, підключення баз даних, реалізація клієнт-серверної архітектури.

# МАТЕМАТИЧНА МОДЕЛЬ ТА СТРУКТУРА ПРОГРАМИ

Проект, який ми розглядаємо у цій роботі, являє собою веб-додаток, веб-сайт, на якому користувач має змогу переглянути список польотів на поточну дату, ознайомитись з деталями конкретного рейсу, а також здійснити пошук рейсу. Для реалізації проекту ми використали фреймворк, який реалізовує архітектуру mvc.

Моде́ль-вигляд-контро́лер (або Модель-вид-контролер, англ. Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення. Цей шаблон поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відокремлення даних (модель) 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. Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності. Архітектурний шаблон Модель-Вид-Контролер (MVC) поділяє програму на три частини. У тріаді до обов'язків компоненту Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

MVC поділяє цю частину системи на три самостійні частини: введення даних, компонент обробки даних і виведення інформації. Модель, як вже було відмічено, інкапсулює ядро даних і основний функціонал з їх обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу Вигляд може мати декілька взаємопов'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. У функції Контролера входить моніторинг за подіями, що виникають в результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле).

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

ASP.NET MVC являє собою платформу для створення сайтів і веб-додатків з використанням патерну (або шаблону) MVC (model - view - controller). Робота над новою платформою була розпочата в 2007 році, а в 2009 році з'явилася перша версія. В результаті до теперішнього моменту (2018 рік) вже було випущено 5 версій платформи, а сам фреймворк знайшов велику популярність по всьому світу завдяки своїй гнучкості та адаптивності. Шаблон MVC, що лежить в основі нової платформи, має на увазі взаємодію трьох компонентів: контролера (controller), моделі (model) і вигляд (view).

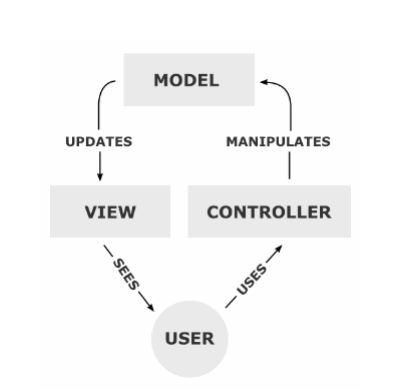


Рисунок 5.3 – Модель MVC

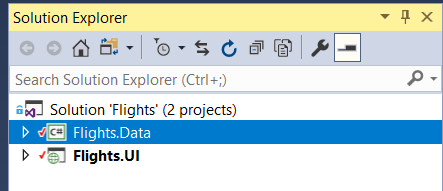
Контролер (controller) представляє клас, з якого власне і починається робота програми. Цей клас забезпечує зв'язок між моделлю і представленням. Отримуючи вводяться користувачем дані, контролер виходячи з внутрішньої логіки при необхідності звертається до моделі і генерує відповідне подання.

Вигляд (view) - це власне візуальна частина або призначений для користувача інтерфейс програми - наприклад, html-сторінка, через яку користувач, що зайшов на сайт, взаємодіє з веб-додатком.

Модель (model) представляє набір класів, що описують логіку використовуваних даних.

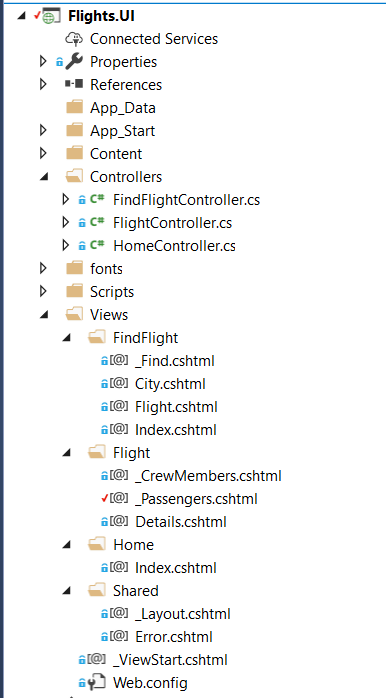
Платформа ASP.NET MVC надає багато можливостей. Перш за все справа у поділі завдань програми (логіка введення, бізнес-логіка і логіка призначеного для користувача інтерфейсу). Також програми, написані за mvc-архітектурою досить зручно тестувати. Всі основні контракти платформи MVC засновані на інтерфейсі і підлягають тестуванню за допомогою макетів об'єкта, які імітують поведінку реальних об'єктів програми. Додаток можна піддавати модульному тестування без запуску контролерів в процесі ASP.NET, що прискорює тестування і робить його більш гнучким. Для тестування можливе використання будь-якої платформи модульного тестування, сумісної з .NET Framework. Наступною перевагою є можливість розширення і доповнення платформи. Компоненти платформи ASP.NET MVC можна легко замінити або налаштувати. Розробник може підключати власний механізм уявлень, політику маршрутизації URL-адрес, сериализацию параметрів методів дій і інші компоненти. Платформа ASP.NET MVC також підтримує використання моделей контейнера впровадження залежності (DI) і інверсії елемента управління (IOC). Модель впровадження залежності дозволяє впроваджувати об'єкти в клас, а не очікувати створення об'єкта самим класом. Модель інверсії елемента управління вказує на те, що якщо один об'єкт вимагає інший об'єкт, то перші об'єкти повинні отримати другий об'єкт із зовнішнього джерела (наприклад, з файлу конфігурації). Це полегшує тестування. Безперечно перевагою є розширена підтримка маршрутизації ASP.NET. Цей потужний компонент зіставлення URL-адрес дозволяє створювати додатки з зрозумілими URL-адресами, які можна використовувати в пошуку. URL-адреси не повинні містити розширення імен файлів і призначені для підтримки шаблонів іменування URL-адрес, які забезпечують адресацію, оптимізовану для пошукових систем (SEO) і для передачі репрезентативного стану (REST).

Підтримка існуючих функцій ASP.NET. ASP.NET MVC дозволяє використовувати такі функції, як перевірка справжності за допомогою форм і Windows, перевірка достовірності за URL-адресою, членство і ролі, кешування виведення і даних, управління станом сеансу і профілю, спостереження за працездатністю, система конфігурації і архітектура постачальника.

Проект, написаний за допомогою фреймворку ASP.NET MVC 5 на мові програмування C# у Visual Studio 2017 має робочу назву Flights та складається з двох проектів. Така структура зумовлена доцільністю розділення моделей (базових класів з усіма їх властивостями) від контролерів та представлень. Відображення структури проекту у Visual Studio наведено нижче:

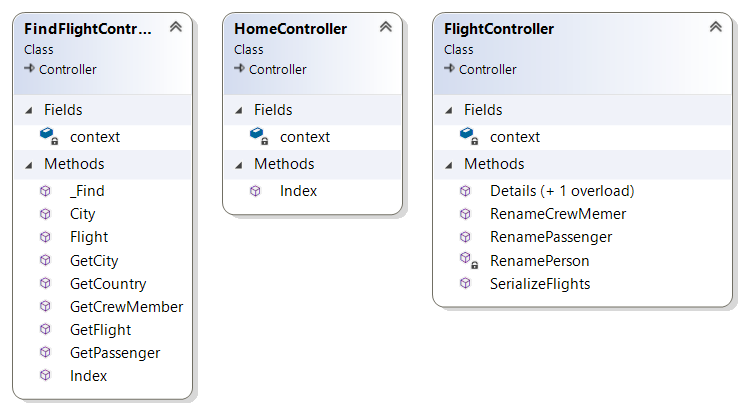
Flights.UI – це та частина проекту, у якій знаходиться основна логіка проекту, ті файли, які відповідають за її відображення та інтеракцію з користувачем, а також посилання на Flights.Data для зв’язку з базою даних. Така структура допомагає зробити проект більш гнучким на менш залежним від ймовірних наслідків при зміні бази даних.

Оскільки для реалізації проекту ми використовувати популярний фреймворк ASP.NET MVC 5, який є MVC-фреймворком, тобто реалізує архітектуру Модель-Представлення-Контроллер, то основними частинами нашого проекту являються контроллери (Controllers), моделі (ViewModels) та представлення (View). Нагадаємо, що Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку. Представлення – це те, що ми отримуємо на виході виконання програми, тобто те, з чим безпосередньо взаємодіє користувач. А Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду. Структура проекту Flights.UI наступна:



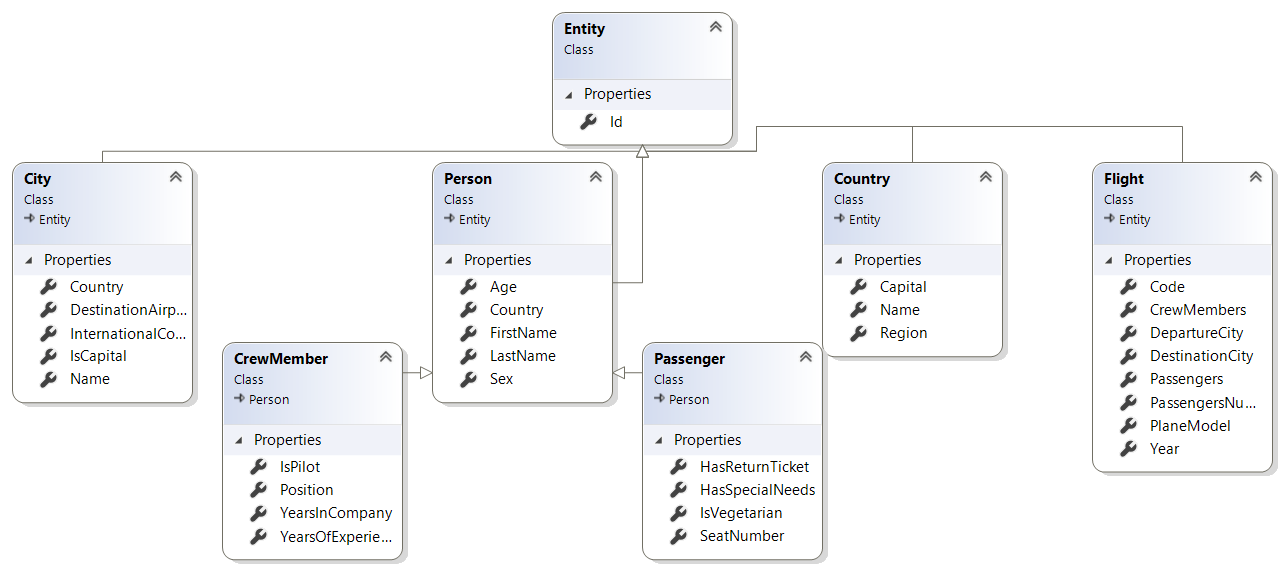
Основними частинами проекту являється контролери та представлення. Також у проекті UI є папка Scripts, у якій знаходяться скріпти, написані на Javascript, які використовуються для обробки дій користувача у браузері – таких як кліки чи наведення.

Що стосується котроллерів, їхніх властивостей та методів, то схематично їх можна зобразити так (діаграми зроблені за допомогою спеціального функціоналу Visual Studio):



У наступних розділах ми більш детально розглянемо призначення методів та властивостей класів FindFlightController, FlightController та HomeController.

У проекті Flights.Data знаходяться моделі, основні класи проекту. Також саме у цій частині загально проекту визначено зв'язок кожної окремої моделі з таблицями баз даних. Властивості та методи класів ми розглянемо пізніше. Загальна структура та взаємозв’язок класів зображено на наступному рисунку:



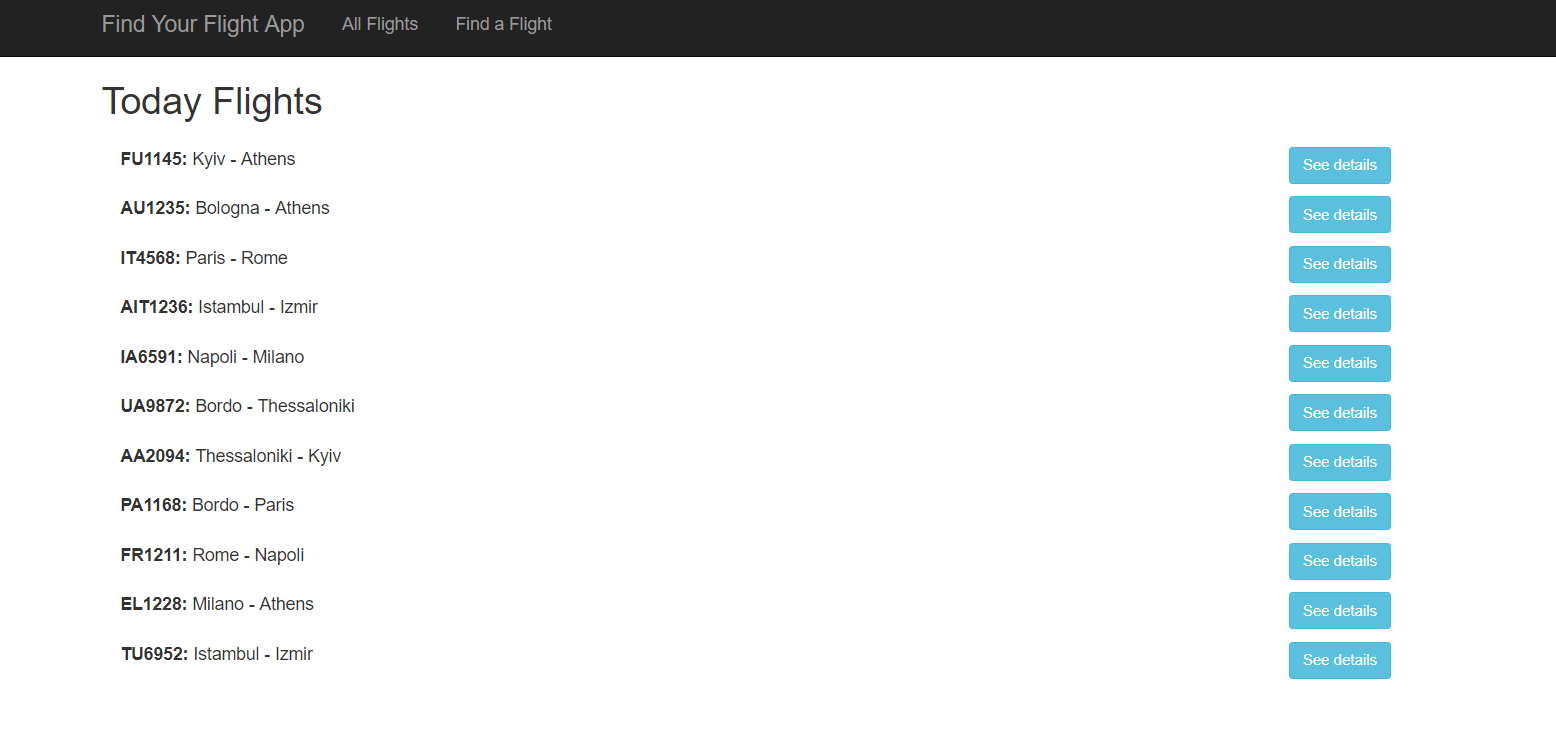
Для підключення баз даних ми використовували Microsoft SQL Server. Бази даних містять п’ять таблиць згідно з п’ятьма моделями City, Passenger, CrewMember, Country та Flight. Структуру таблиць бази даних зображено на рисунку нижче:



Як бачимо, деякі таблиці пов’язані між собою та містять посилання на пов’язані дані. Так, наприклад, таблиця польотів пов’язана з таблицею міст, а таблиці, які містять інформацію про пасажирів та екіпаж, посилаються на таблицю з польотами через поле Flight\_Id.

# ПЕРЕЛІК ТА ПРИЗНАЧЕННЯ РЕЖИМІВ ТА СТУКТУРА ДІАЛОГУ

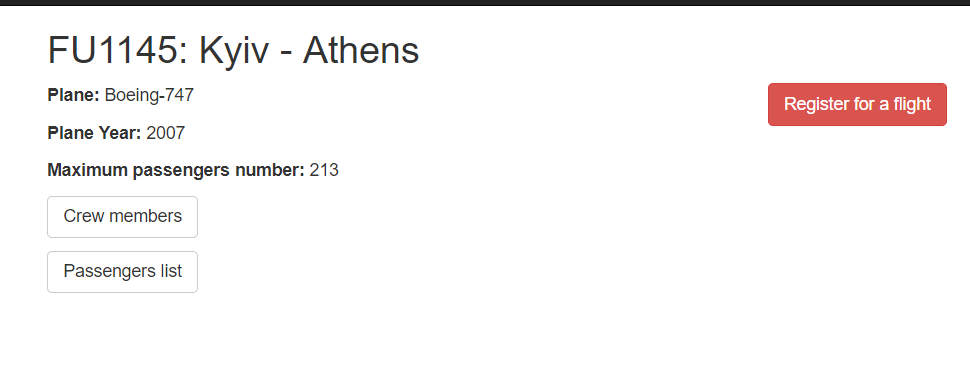
Користувач знайомиться з проектом через інтерфейс головної сторінки, який виглядає наступним чином:



Інтерфейс головної сторінки складається з кількох елементів:

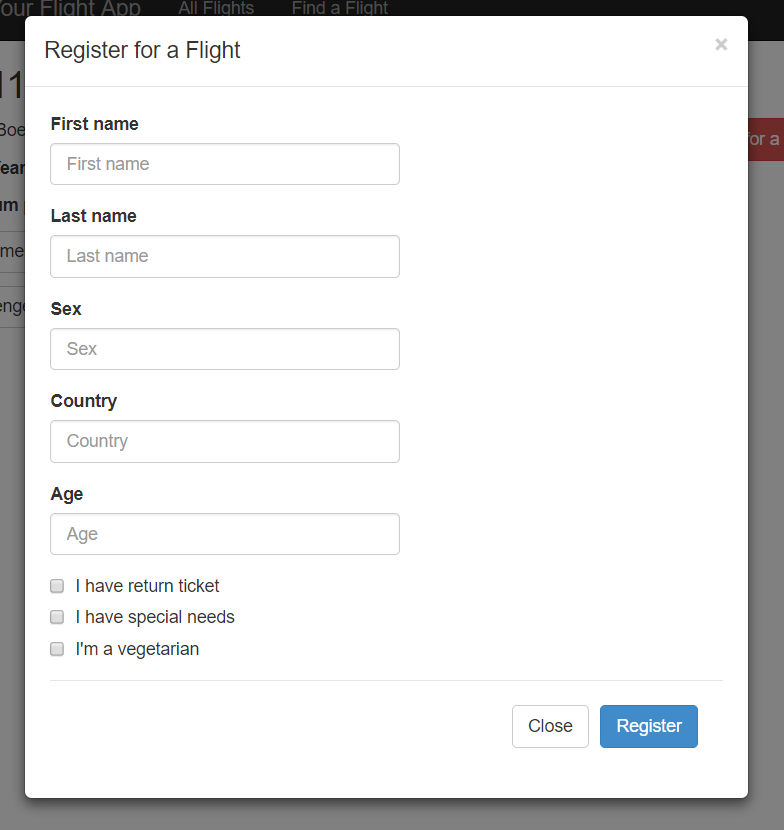
* Головне меню, яке знаходиться зверху та містить назву проекту, посилання на сторінку пошуку рейсів та список усіх доступних польотів.
* Список рейсів на поточну дату.

При натисканні на кнопку See details користувач перейде на сторінку деталей польоту, наприклад:



На цій сторінці користувач має змогу переглянути список пасажирів та екіпажу. Списки згортаються на розгортаються за допомогою ui-фреймворку Bootstrap, який призначено для модифікації та стилізації веб-інтерфейсів.

У разі, якщо користувач натискає кнопку Register for a flight, перед ним у модальному вікні з’являється форма, яка після заповнення усіх необхідних полів та після натискання кнопки Register за допомогою метода post відправляє дані про нового пасажира до бази даних та змінює таблицю Passenger.



СТРУКТУРА ДАНИХ ТА РЕСУРСІВ ПРОГРАМИ

Оскільки наш проект має архітектуру, властиву MVC моделі, класи належать або до контролерів, або до моделей. У розділі Математична модель та структура програми наведено загальну діаграму класів. У цьому розділі ми більш детально розглянемо властивості на методи всіх класів.

Спочатку розглянемо основні моделі. Для того, щоб реалізувати трьохрівневе наслідування, ми створили базовий клас Entity, який має єдину властивість – Id, яка унаслідується усіма класами:

public class Entity

{

public int Id { get; set; }

}

Моделі проекту:

* Клас Country унаслідується від Entity. Крім властивості Id він також має поля Name, Region та Capital. Усі вони є строками, тобто мають тип даних string. Клас Country:

public class Country : Entity

{

public string Name { get; set; }

public string Region { get; set; }

public string Capital { get; set; }

}

* Клас City має три властивості типу string (Name, DestinationAirport та InternationalCode), властивість IsCapital булевого типу даних, а також властивість, яка вказує на зв'язок даного класу з іншим класом – Country. У відповідній таблиці бази даних поле Country буде позначатися відповідним до країни Id. Клас City з його властивостями:

public class City:Entity

{

public string Name { get; set; }

public virtual Country Country { get; set; }

public bool IsCapital { get; set; }

public string DestinationAirport { get; set; }

public string InternationalCode { get; set; }

}

* Клас Flight:

public class Flight : Entity

{

public string Code { get; set; }

public virtual City DepartureCity { get; set; }

public virtual City DestinationCity { get; set; }

public string PlaneModel { get; set; }

public int PassengersNumber { get; set; }

public int Year { get; set; }

public virtual ICollection<CrewMember> CrewMembers { get; set; }

public virtual ICollection<Passenger> Passengers { get; set; }

}

* Класи CrewMember та Passenger унаслідуються від класу Person. Таким чином досягається трьохрівневе наслідування:

public class Person : Entity

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string Sex { get; set; }

public int Age { get; set; }

public string Country { get; set; }

}

Таким чином Passenger унаслідується від Person та Entity

public class Passenger : Person

{

public int SeatNumber { get; set; }

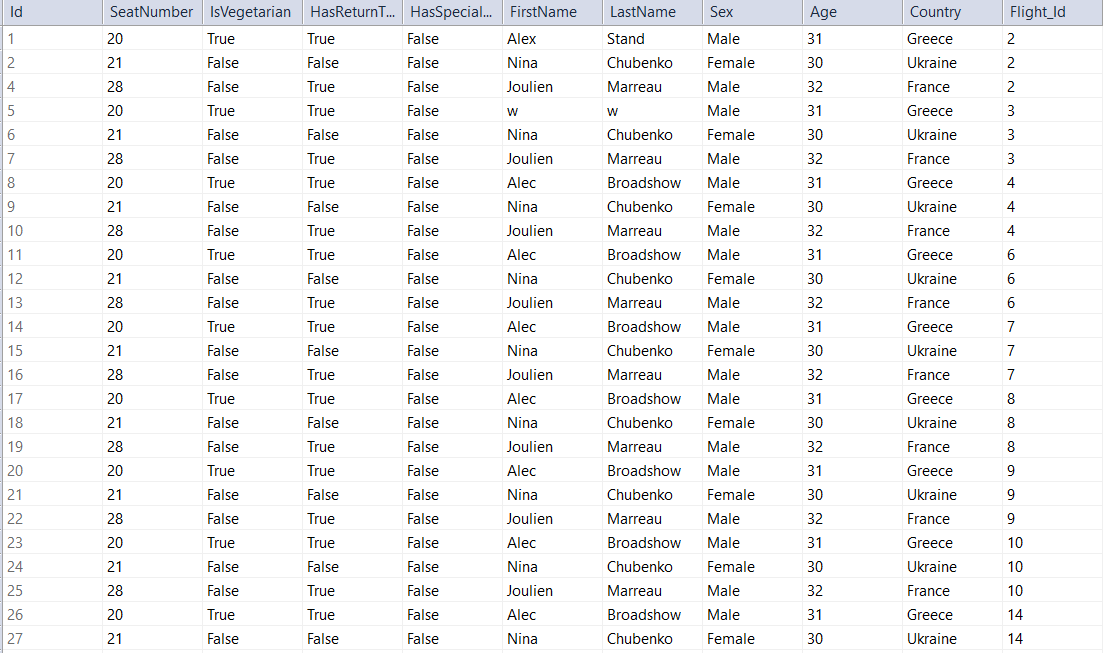
public bool IsVegetarian { get; set; }

public bool HasReturnTicket { get; set; }

public bool HasSpecialNeeds { get; set; }

}

Властивості моделей відповідають полям таблицях баз даних. Розглянемо, наприклад, таблицю Passenger:



Також до моделей належить допоміжний клас Step, який не пов'язаний з іншими моделями, але використовується для логіки відображення представлень на етапі пошуку польоту.

public class Step

{

public string CurrentStepName { get; set; }

public string NextStepLink { get; set; }

}

Як уже зазначалось вище, уся логіка щодо відображення та взаємодії з користувачем знаходиться у проекті Flights.UI. Тут знаходяться контролери, представлення, а також допоміжні скрипти, написані на Javascript, за допомогою яких відбувається обробка кліків на клієнті, тобто у браузері.

Клас HomeController має лише один метод, який відповідає за зв'язок моделі польотів, перетворення даних про всі польоти у список та їхнє відображення в представленні Index. HomeController виглядає наступним чином:

public class HomeController : Controller

{

private DbContext context = new DbContext();

public ActionResult Index()

{

var flights = context.Flight.ToList();

return View(flights);

}

}

Клас FlightController має дещо більше методів:

* Метод Details() повертає інформацію про політ, який було знайдено у таблиці бази даних за кодом, вказаним у якості параметра:

[HttpGet]

public ActionResult Details(string code)

{

var flight = context.Flight.FirstOrDefault(f => f.Code == code);

return View(flight);

}

* Метод Details, який здійснюється за допомогою HttpPost запиту дає змогу додавати пасажира до таблиці бази даних.

[HttpPost]

public ActionResult Details(Passenger passenger, int flightId)

{

var flight = context.Flight.FirstOrDefault(f => f.Id == flightId);

flight.Passengers.Add(passenger);

context.SaveChanges();

return RedirectToAction("Index", "Home");

}

Внесення змін до бази даних відбувається за допомогою методу SaveChanges(). Після цього користувача автоматично редиректить на головну сторінку, і тепер під час перегляду деталей рейсу щойно доданого пасажира буде видно у списку усіх пасажирів рейсу.

* Методи RenameCrewMemer(), RenamePassenger() та RenamePerson() демонструють один з основних принципів поліморфізм, який полягає у роботі з різними об’єктами однаковим чином. Так, у нашому випадку, методи RenameCrewMemer() та RenamePassenger() однаково використовують метод RenamePerson().

Клас FindFlightController містить основну логіку пошуку рейсу. Серед його методів:

* \_Find() – повертає часткове предтавлення – тобто таку частину html, яка може бути підключена та використана у інших views.
* Index() – який повертає представлення – тобто те, що бачить користувач. Це представлення у нашому проекті містить посилання на часткове представлення \_Find(). На цьому етапі користувач виконує пошук країни, здійснюючи запит до бази даних.
* City() – також включає в себе часткове представлення \_Find(). Для користувача має вигляд поля для пошуку та кнопки, після натискання якої відбувається get запит до бази даних.
* Flight() – повертає представлення, яке являє собою поле форми пошуку рейсу.

Наведені вище методи стосуються формування представлень. Проте цей контролер також містить методи, які формують відповіді для get-запитів, зроблених на клієнті, тобто у браузері, у форматі json. Серед таких методів даного контролера:

* GetCountry(string key) – повертає json, який містить масив усіх країн. Цей метод приймає один аргумент типу строка, за яким іде пошук у таблицях бази даних. Якщо ж аргумент – пуста строка, метод повертає масив з усіх країн.

public ActionResult GetCountry(string key)

{

var countries = context.Country.ToArray();

if (key.Length > 0) countries = countries.Where(c => c.Name.ToLower().IndexOf(key.ToLower()) > -1).ToArray();

return Json(countries, JsonRequestBehavior.AllowGet);

}

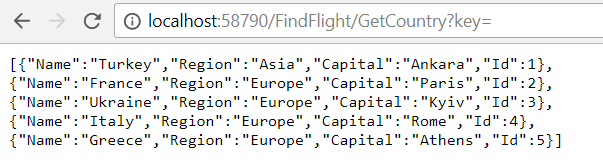
* GetCity(string key, string filterParam) – подібний до попереднього методу, проте він, на відміну, приймає два параметри. Перший – строка, за якою йде пошук серед назв міст, другий – назва країни, Id яка повинна відповідати полю Country\_Id. Фільтрація за другим параметром необхідна, оскільки цей get запит здійснюється на етапі, коли користувач уже вибрав країну, тобто було б неправильним відповідати масивом з містами, які знаходяться в країнах, які не цікавлять користувача.
* GetFlight(string key, string filterParam) – повертає список усіх польотів у форматі json.
* GetCrewMember(string key) – повертає масив з іменами усіх членів екіпажу, якщо їхнє ім’я та прізвище містить строку-аргумент.
* GetPassenger(string key) – повертає масив з іменами усіх пасажирів, якщо їхнє ім’я та прізвище містить строку-аргумент.

Розглянемо більш детально етапи відправлення get запиту та отримання і обробки відповіді сервера на прикладі запиту на отримання обраних країн:

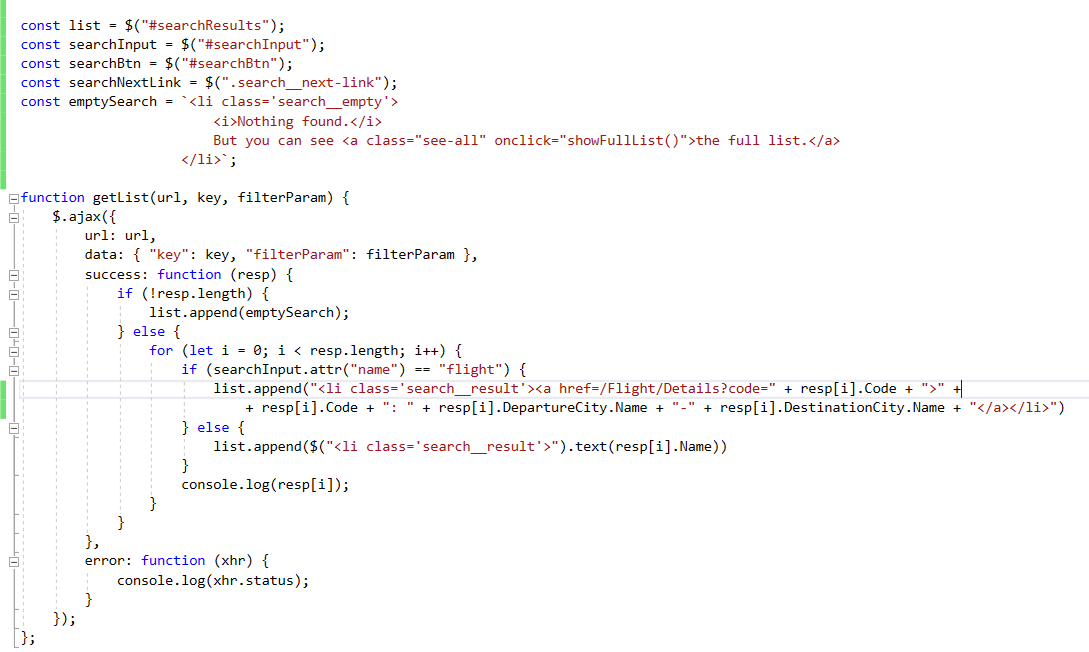
1. Першочергово в FindFlightController контроллері ми формуємо відповідь сервера на запит, вказавши дозволений метод:

return Json(countries, **JsonRequestBehavior.AllowGet**);

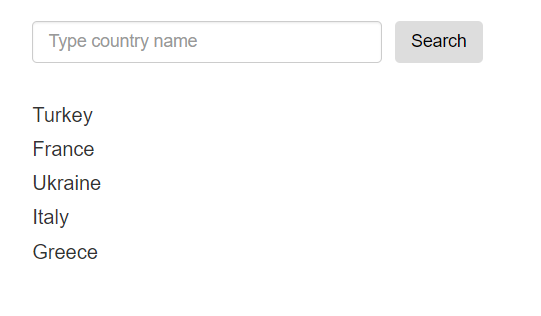
Тобто якщо ми у браузері звернемось за адресою “/FindFlight/GetCountry?key=”, то побачимо відповідь сервера у форматі json, яка виглядає наступним чином:



1. Для того, щоб реалізувати клієнт-серверну архітектуру, запит ініціалізуватиметься з браузера, тобто з клієнта. Натискання кнопки Search оброблятиметься скриптом Javascript, а точніше за допомогою фреймфорка JQuery і ajax запиту.



Як бачимо, у разі успішної відповіді сервера, ми заповнюємо DOM елемент списку отриманою інформацією. У наслідок цих дій на сторінці браузера користувач бачить такий список:



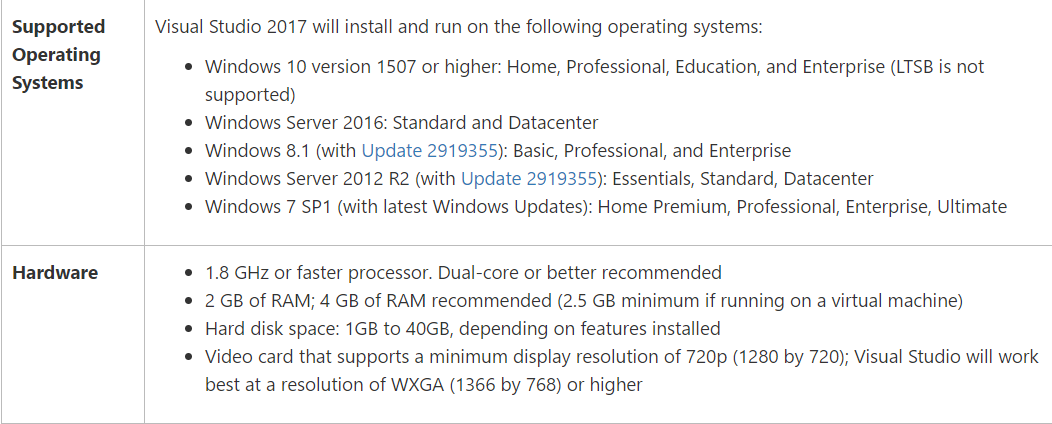
Для того, що здійснити сериалізацію у формат XML, яку було зазначено як одну з вимог до даної роботи, ми використали методу SerializeFlights():

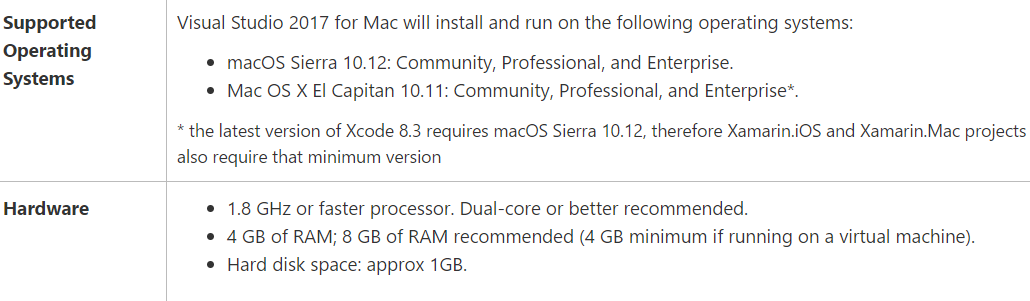


Цей метод здійснює сериалізацію у формат XML. В результаті його виконання ми можемо отримати XML документ, який містить список усіх польотів. Він доступний за адресою: /Flight/SerializeFlights

# ІНСТРУКЦІЯ АДМІНІСРАТОРА

Програма під робочою назвою “Flights” написана на мові C# у програмі Visual Studio 2017. Це програмне забезпечення має все необхідне для розробки, збирання та запуску проекту. Visual Studio 2017 може бути встановлена на таких операційних системах, як Windows, Linux та Mac OS. Вимоги до операціних систем для встановлення Visual Studio 2017:





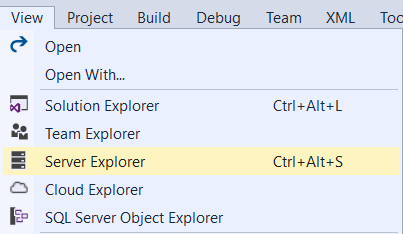
Для баз даних ми використовуємо Microsoft SQL Server. Завантажити дане програмне забезпечення офіційно можна за посиланням: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017-editions. Зв'язок з базою даних необхідно вказати у конфігураційному файлі Web.config:

<connectionStrings>

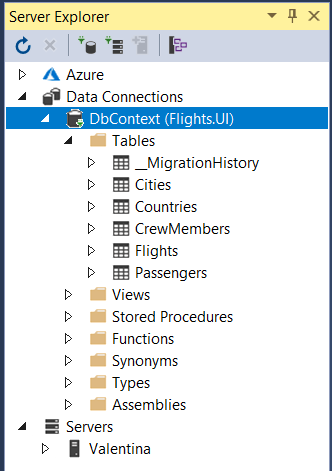
<add name="DbContext" connectionString="Server=.;Initial Catalog=Flights;Persist Security Info=False; Integrated security = true;" providerName="System.Data.SqlClient" />

</connectionStrings>

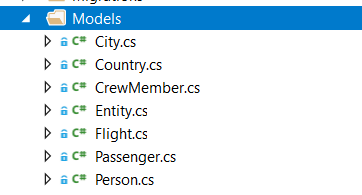
Щоб подивитись вміст таблиць бази даних необхідно у верхньому меню вибрати пункти View та Server Explorer, як це показано на рисунку нижче:



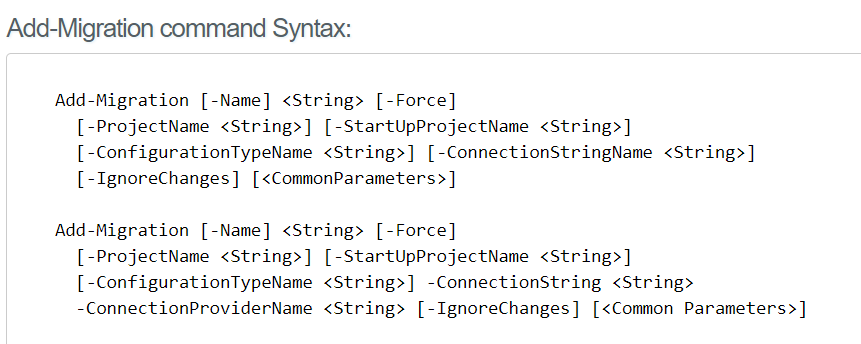
Після цього програміст матиме змогу бачити структуру таблиць, яка виглядає у нашому випадку наступним чином:

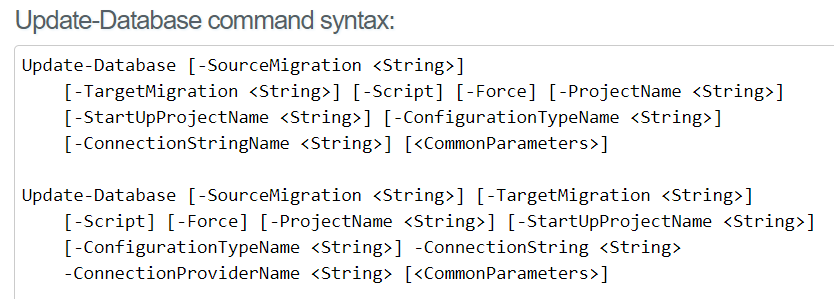


Щоб додати ще одну таблицю, або змінити чи додати полу до тієї чи іншої таблиці, необхідно додати ще одну модель до існуючого списку або змінити необхідні властивості існуючої моделі.

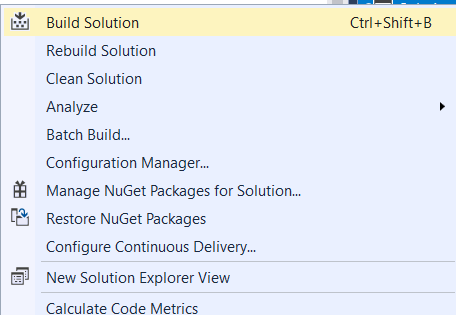


Для того щоб зміни було видно у базі даних, потрібно здійснити міграцію. Це досягається за допомогою двох команд: add-migration та update-database.

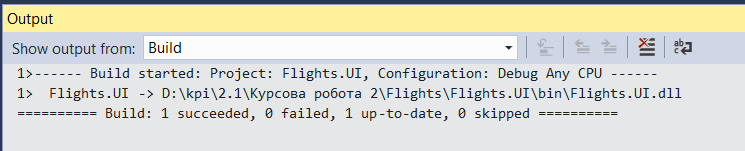




Для того, щоб збілдити проект, потрібно вибрати з контекстного меню пункт Build Solution:



В консолі з’явиться повідомлення про результат білда.

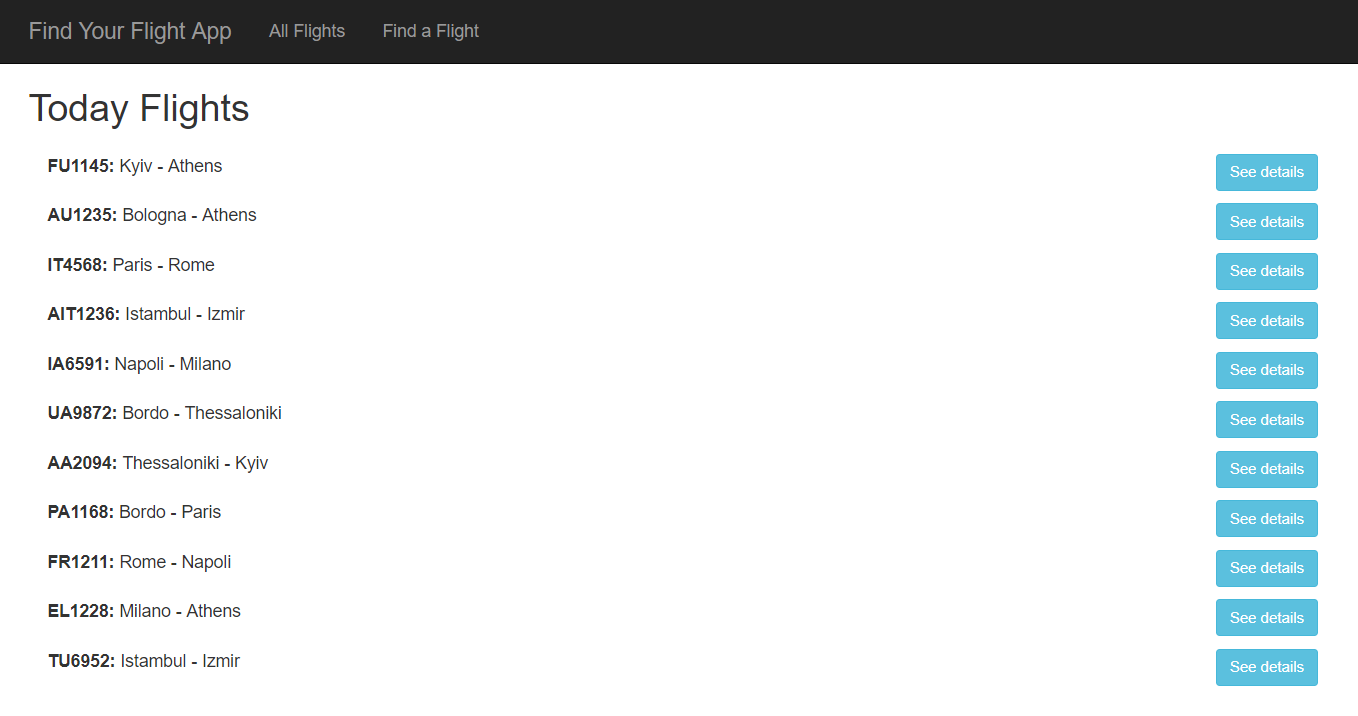


Щоб запустити проект локально, потрібно натиснути кнопку з назвою обраного браузера у верхньому меню. Після цього запуститься браузер і ми матимемо змогу бачити початкову сторінку проекту.

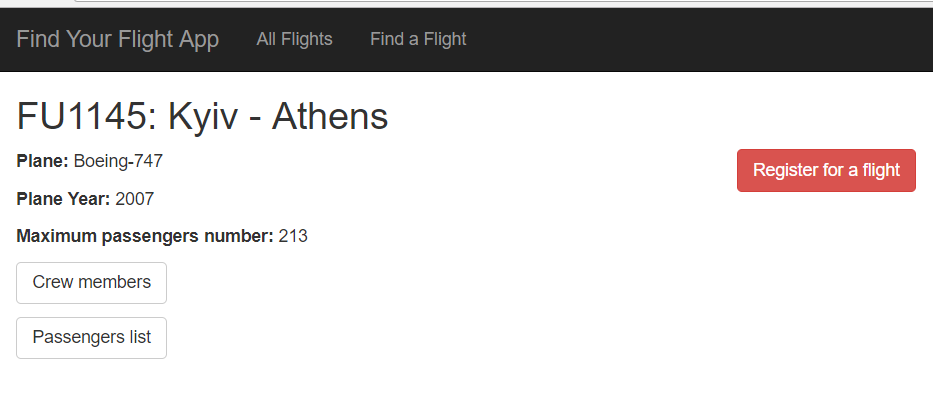


# КЕРІВНИЦТВО КОРИСТУВАЧА

Програма під назвою Flights для користувача має вигляд веб-сайту, на якому він може знайти інформацію про рейси на поточку дату. Оскільки це веб-сайт, то жодного особливого програмного забезпечення для користування нею не потрібно. Єдине, що необхідно користувачеві – це веб-браузер, який частіше за все встановлений на майже всіх операційних системах за замовчуванням.



Користувач має змогу як переглянути список та деталі усіх рейсів, так і знайти якийсь окремий політ, якщо його такий цікавить. Ці дві основні опції доступні у верхньому основному меню поруч з назвою веб-сайту – «Find Your Flight».



При перегляді деталей окремого рейсу, користувач може ознайомитись з такою інформацією як модель та рік випуску літака, максимальну кількість пасажирів, яку може перевозити судно, список пасажирів, які уже зареєструвались на рейс та склад екіпажу. Також користувач має змогу зареєструватись на рейс, ввівши усі необхідні дані, такі як: ім’я, прізвище, країну, уподобання у їжі та наявність особливих потреб.

# ВИСНОВКИ

У ході курсової роботи було створено у Microsoft Visual Studio 2017 солюшен з 2 взаємозв'язаними проектами. У результаті виконання було створено веб-проект, який можна переглянути у будь-якому браузері. Проект складається з п’яти взаємопов’язаних веб-сторінок. Користувач має змогу здійснювати запити до баз даних, шукаючи рейс, який його цікавить. Пошук розбито на поетапні пошуки країни, міста, а на фінальному етапі – окремого рейсу.

Відповідно до завдання та вимог, які висувались до курсової роботи, була організована об’єктна модель та реалізоване трьохрівневе наслідування. Кожна модель (Country, City, CrewMember, Passenger, Flight) має не менше п’яти властивостей різних типів даних.

Також в роботі було реалізовано логіку зведення та відображення даних, інкапсуляцію, поліморфізм, підключення баз даних, реалізацію клієнт-серверної архітектури.