



DESIGNING AND DEVELOPING APPLICATIONS ON THE CLOUD

INDIVIDUAL ASSIGNMENT

Module	CT071-3-5-3-DDAC
Project Title	Online Flight Booking System (UIA)
Lecturer	DR. KALAI ANAND A/L RATNAM
Intake	UC3F1701SE
Student Name	Mohammad Kazem Hassan Nejad
Student ID	TP035903
Hand-in Date	9th October 2017

Acknowledgement

Prima facie, I would like to thank my parents for their unconditional support and unceasing encouragement throughout this project.

I would like to thank Dr. Kalai for sharing his pearls of wisdom and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me throughout the project in the form of hands-on lab and other provided materials.

Lastly, I would like to thank Asia Pacific University for providing me the opportunity to excel academically by providing me all the necessary resources and facilities.

Table of Contents

1.0 Introduction.....	1
1.1 Background Information	1
1.2 Aim and Objectives	2
1.3 Scope	2
1.4 Major Functions	3
1.5 Assumptions	3
2.0 Project Plan	4
2.1 List of Tasks.....	4
2.2 Gantt Chart	4
3.0 Design	5
3.1 Design Modelling.....	5
3.1.1 Use Case Diagram	5
3.1.2 Entity Data Model	6
3.1.3 Sequence Diagram.....	7
3.1.4 View Map	9
3.2 Interface Design (UI Mockup)	10
3.3 Cloud Architecture Diagram	14
4.0 Implementation	16
4.1 Core Application Implementation.....	16
4.1.1 External Configuration Store (Cloud Design Pattern).....	16
4.1.2 Security and Authentication	17
4.1.3 Dynamic Seat Picker	19
4.1.4 GeoIP-based Currency.....	21
4.1.5 Developed Web Pages	24
4.2 Cloud Implementation and Deployment	38
4.2.1 Cost Breakdown	38
4.2.2 Web App.....	41
4.2.3 Azure SQL Database	45
4.2.4 Traffic Manager.....	49
4.2.5 Application Insights.....	52
4.2.6 Azure Blob Storage	55
5.0 Testing.....	56

5.1 Test Plan.....	56
5.1.1 Functional Testing	56
5.1.2 Performance Testing.....	61
5.2 Results & Analysis	62
5.2.1 Functional Tests.....	62
5.2.2 Performance Tests	63
6.0 Conclusion	70
Appendices.....	73

List of Figures

Figure 1. Use Case Diagram	5
Figure 2. Entity Data Model	6
Figure 3. Search Flight Sequence Model	7
Figure 4. Booking Sequence Model.....	7
Figure 5. Register Sequence Model	7
Figure 6. Login Sequence Model.....	8
Figure 7. Edit User Sequence Model	8
Figure 8. MVC View Map	9
Figure 9. Homepage/Search for flight Mockup	10
Figure 10.Search Result Page Mockup	11
Figure 11. Login Page Mockup	11
Figure 12. Register Page Mockup.....	12
Figure 13. Booking Page Mockup (1/3)	12
Figure 14. Booking Page Mockup (2/3)	13
Figure 15. Booking Page Mockup (3/3)	13
Figure 16. Edit Profile Page Mockup	14
Figure 17. Proposed Cloud Architecture Diagram	15
Figure 18. External Configuration Store Diagram (Narumoto, et al., 2017)	17
Figure 19. Upload Config Blob - Code Snippet	17
Figure 20. Config Checker Run - Code Snippet	17
Figure 21. Anti-forgery token Generation and Validation in View and Controller....	18
Figure 22. Authorize and AllowAnonymous Tags	18
Figure 23. OWIN-based Authentication - ASP.NET Identity	19
Figure 24. Boeing 737 SVG Seat Layout (AnyChart, 2017)	20
Figure 25. Div Tag to hold Seat SVG graph and Selected Seat - Code Snippet	20
Figure 26. Generate and hold SVG graph – Code Snippet	20
Figure 27. Update Selected Seat - Code Snippet	21
Figure 28. User IP Address (v4) based on Request variables	21
Figure 29. GetIPLocation - Code Snippet.....	22
Figure 30. Currency based on IP - Code Snippet.....	22
Figure 31. User Request from Malaysia - Ringgit Currency	23
Figure 32. User Request from Finland - Euro Currency.....	23

Figure 33. Homepage Implemented Design	24
Figure 34. Available Flights Implemented Design	25
Figure 35. Register Page Implemented Design.....	26
Figure 36. Login Page Implemented Design	27
Figure 37. Homepage with logged in user	28
Figure 38. View Bookings Implemented Design - No bookings.....	29
Figure 39. Edit Profile Implemented Design	30
Figure 40. Change Password Implemented Design - Profile	31
Figure 41. Search Result Page - Implemented Design	32
Figure 42. Book Flight Ticket Implemented Design - Part 1	33
Figure 43. Book Flight Ticket Implemented Design - Part 2	34
Figure 44. Book Flight Ticket Implemented Design - Part 3	35
Figure 45. Booking Confirmed Page	36
Figure 46. View Bookings Implemented Design.....	37
Figure 47. Traffic Manager - Cost Breakdown.....	39
Figure 48. Azure SQL Database - Cost Breakdown	39
Figure 49. Blob Storage - Cost Breakdown	39
Figure 50. Web Application S1 - Cost Breakdown	40
Figure 51. Web Application S2 - Cost Breakdown	40
Figure 52. Web Application S3 - Cost Breakdown	41
Figure 53. Basic vs Standard Pricing Tier	42
Figure 54. SEA-deployed Web App	43
Figure 55. West Europe Web App.....	43
Figure 56. Continuous Deployment on Azure Web App.....	44
Figure 57. GitHub Repository.....	44
Figure 58. Scale-out during high demand.....	45
Figure 59. SQL Database Service Tier Comparison (Rabeler, et al., 2017).....	46
Figure 60. Chosen Database Service Tier	47
Figure 61. Database Server Location.....	47
Figure 62. SQL Server	48
Figure 63. Database Security Settings	49
Figure 64. Traffic Manager Performance Routing Method (Dwivedi, et al., 2017)....	50
Figure 65. Traffic Manager Endpoints.....	51
Figure 66. Traffic Manager Configuration	51

Figure 67. Application Insights - Failures	52
Figure 68. Example of exception during failure	53
Figure 69. Performance Metrics - Application Insights.....	54
Figure 70. Response Time Visual Map.....	54
Figure 71. Response Time Analytics - Application Insights	54
Figure 72. Configuration Container – Blob Storage	55
Figure 73. Auditing Log Container - Blob Storage	55
Figure 74. Performance Metrics - S1 - 100 Concurrent Users	63
Figure 75. Performance Metrics - S1 - 300 Concurrent Users	64
Figure 76. Performance Metrics - S1 - 500 Concurrent Users	65
Figure 77. Performance Metrics - S1 - 1000 Concurrent Users	66
Figure 78. Throughput across service tiers with varying concurrent user load chart..	67
Figure 79. Latency across service tiers with varying concurrent user load chart	68

List of Tables

Table 1. Functional Test Plan	57
Table 2. Performance Test Plan	61
Table 3. Functional Test Results.....	62
Table 4. Throughput every second across varying user load and service tiers.....	67
Table 5. Average Response Time across varying user load and service tiers	68

1.0 Introduction

1.1 Background Information

In 1992, Ukraine International Airlines was established as a global airline of the autonomous Ukraine by the Ukrainian State Association of Civil Aviation and GPA, the world's largest aircraft lessor (Lavrov & Gordienko, 2014). Within the past 15 years, the airline has allured some of the worlds' top business and strategic investors. At first, UIA was intended to set up constant operations amongst Ukraine and Western Europe and effectively remain as a passenger carrier airline (Lavrov & Gordienko, 2014). In 2009, which marked UIA's 18th commemoration, they ended up as one of Ukraine's key aviation competitor through the domination of over 20% of the market.

In the midst of the competitive environment and extraordinary cost-based rivalry, UIA took a step forward to transform their business adaptability and permitted beginning change from a point-to-point into an international network airline (Lavrov & Gordienko, 2014). In 2013, after the business fall of the primary competitor, UIA ended up being the sole airlines with its level of infrastructural influence within the Ukraine market. The change procedure ended up being significantly more unique than it had been arranged at first. Over the next year, UIA doubled in size in terms of number of aircrafts and amidst the political and economic instability, and the debasement of the Ukrainian currency, UIA was striven to remain competitive through strategical overhauls to their staff, routes, and carriers resulting in reduction in ticket fares to enable them to further internationalize their services and contend as one of the world's low-fare carriers (Lavrov & Gordienko, 2014).

UIA has long used technology to help reduce costs and improve customer service. With the advancements in technology, and maturation of cloud computing, the Chief Information Officer, Dmitry Prudnikov, has realized that through migration of their website out of their own datacenters into the public cloud, they can gain a competitive advantage over their competitors and improve customer services with the extensive set of tools and services that are provided. UIA has considered the different available public cloud platforms, and has chosen Microsoft Azure, due to a range of reasons, including its compatibility with open source software.

1.2 Aim and Objectives

The aim of the project is to design and develop a single-tenant flight booking web application that will aid in the reduction of costs and improvement of customer service. The objectives of the project are:

1. Develop and implement an intuitive Ukraine International Airlines (UIA) flight booking web application.
2. Deploy the web application as an Azure App Service in two different regions.
3. Create and configure an Azure SQL database for the storage of customer data.
4. Implement traffic manager to balance load and optimize performance during peak seasons as well as providing scaling out/in to meet the demands at any given time.
5. Implement security measurements for the protection of the data on the database.

1.3 Scope

The scope of the project has been well-defined as follows:

- The web application will be hosted in the two most appropriate regions to serve as a proof of concept.
- The web application will consist of all the necessary functions for the pedestal needs of a customer in regard to booking a flight.
- The public cloud platform that will be used to deploy the web application is Microsoft Azure.
- Platform as a Service (PAAS) will be used within Microsoft Azure.
- The chosen Azure services and design considerations are based on the constrained budget of RM 150.
- The web application will enhance the usability through customized information, however it will only consist of currency changes with no actual conversion of the monetary value.
- The web application will exclude administrative actions, such as flight management.

1.4 Major Functions

The major functions of the web application are to provide:

- Create customer profile
- Login to the system
- Search for flight
- Select a seat using a dynamic seat picker.
- View customized ticket and booking information based on the user's geographical location.
- View and edit customer profile

The major functions provided through deployment on Microsoft Azure are as follows:

- Traffic management to direct and reduce the user load across the deployed web applications in the varying regions.
- Performance optimization through numerous performance tests
- Continuous deployment of the latest committed changes to the web application.
- Storage of customer data on Azure SQL database
- Scale-out to meet the needs of demands during peak seasons

1.5 Assumptions

There are numerous assumptions that set the basis for undertaking this project, which are as follows:

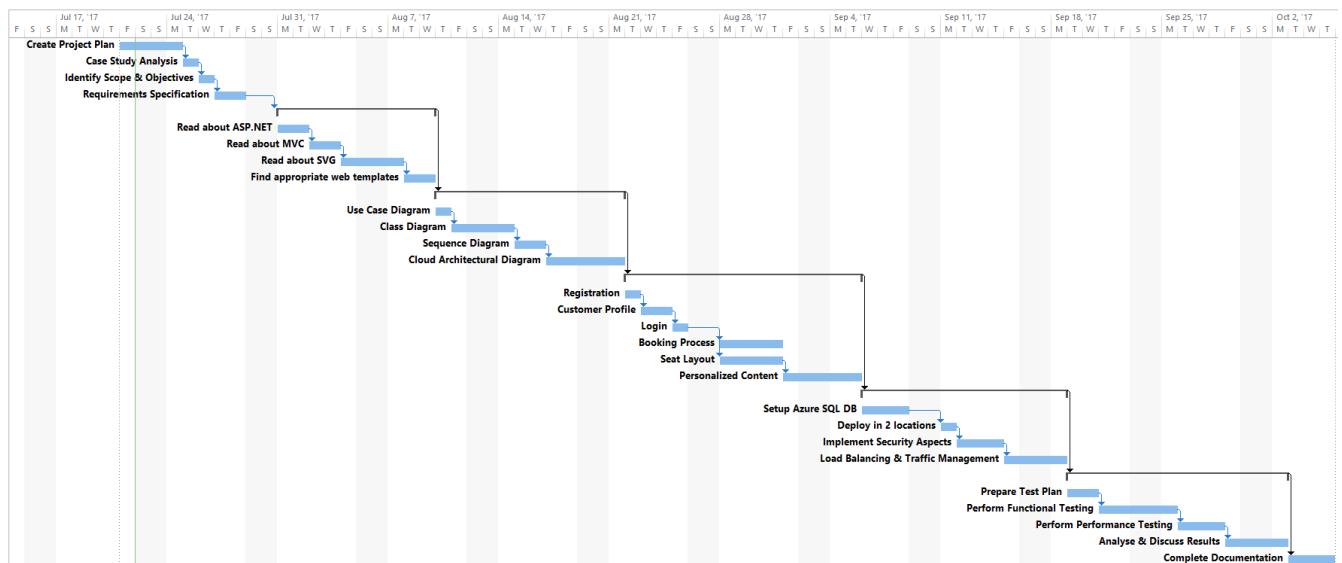
- There are flights from every airport to every other airport on a daily basis.
- There is a limited budget of RM 150 to be expended on the Azure services.
- The fleet used by UIA are all of the same aircraft type, which is the Boeing-737.
- The development team is situated in Malaysia.
- The project serves as a proof-of-concept rather than the delivery of a complete flight booking system with all the possible Azure services without a constrained budget.

2.0 Project Plan

2.1 List of Tasks

Task Name	Duration	Start	Finish
Create Project Plan	2 days	Fri 7/21/17	Mon 7/24/17
Case Study Analysis	1 day	Tue 7/25/17	Tue 7/25/17
Identify Scope & Objectives	1 day	Wed 7/26/17	Wed 7/26/17
Requirements Specification	2 days	Thu 7/27/17	Fri 7/28/17
Research on implementation	8 days	Mon 7/31/17	Wed 8/9/17
Read about ASP.NET	2 days	Mon 7/31/17	Tue 8/1/17
Read about MVC	2 days	Wed 8/2/17	Thu 8/3/17
Read about SVG	2 days	Fri 8/4/17	Mon 8/7/17
Find appropriate web templates	2 days	Tue 8/8/17	Wed 8/9/17
Design	8 days	Thu 8/10/17	Mon 8/21/17
Use Case Diagram	1 day	Thu 8/10/17	Thu 8/10/17
Class Diagram	2 days	Fri 8/11/17	Mon 8/14/17
Sequence Diagram	2 days	Tue 8/15/17	Wed 8/16/17
Cloud Architectural Diagram	3 days	Thu 8/17/17	Mon 8/21/17
Implementation	11 days	Tue 8/22/17	Tue 9/5/17
Registration	1 day	Tue 8/22/17	Tue 8/22/17
Customer Profile	2 days	Wed 8/23/17	Thu 8/24/17
Login	1 day	Fri 8/25/17	Fri 8/25/17
Booking Process	4 days	Mon 8/28/17	Thu 8/31/17
Seat Layout	4 days	Mon 8/28/17	Thu 8/31/17
Personalized Content	3 days	Fri 9/1/17	Tue 9/5/17
Deployment	9 days	Wed 9/6/17	Mon 9/18/17
Setup Azure SQL DB	3 days	Wed 9/6/17	Fri 9/8/17
Deploy in 2 locations	1 day	Mon 9/11/17	Mon 9/11/17
Implement Security Aspects	3 days	Tue 9/12/17	Thu 9/14/17
Load Balancing & Traffic Management	2 days	Fri 9/15/17	Mon 9/18/17
Test Phase	10 days	Tue 9/19/17	Mon 10/2/17
Prepare Test Plan	2 days	Tue 9/19/17	Wed 9/20/17
Perform Functional Testing	3 days	Thu 9/21/17	Mon 9/25/17
Perform Performance Testing	3 days	Tue 9/26/17	Thu 9/28/17
Analyse & Discuss Results	2 days	Fri 9/29/17	Mon 10/2/17
Complete Documentation	3 days	Tue 10/3/17	Thu 10/5/17

2.2 Gantt Chart



3.0 Design

3.1 Design Modelling

3.1.1 Use Case Diagram

The diagram below illustrates the various use cases that have been defined as per the requirements needed for the customer to manage their entire booking process, create a profile, and view customized information to enhance their usability.

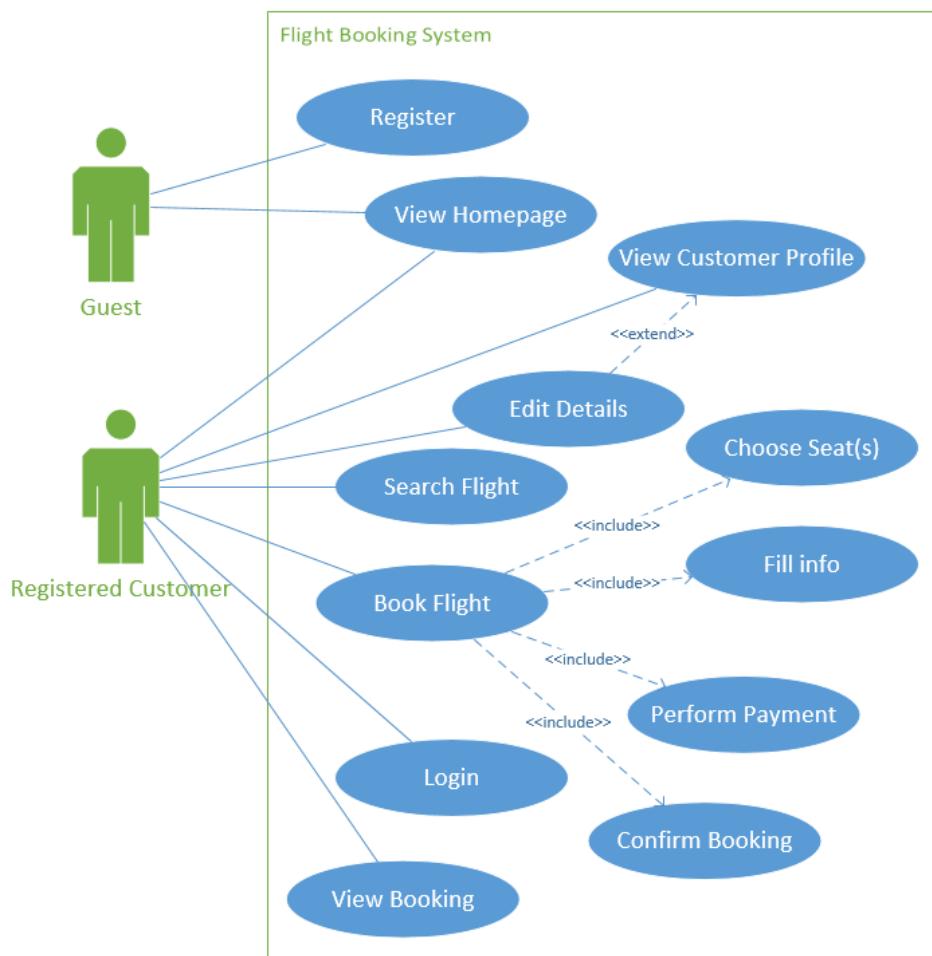


Figure 1. Use Case Diagram

3.1.2 Entity Data Model

The figure below illustrates the entity data model diagram that was designed as part of the proposed MVC architecture. It depicts the relational database entities in a similar format as to an Entity-Relationship Diagram (ERD).

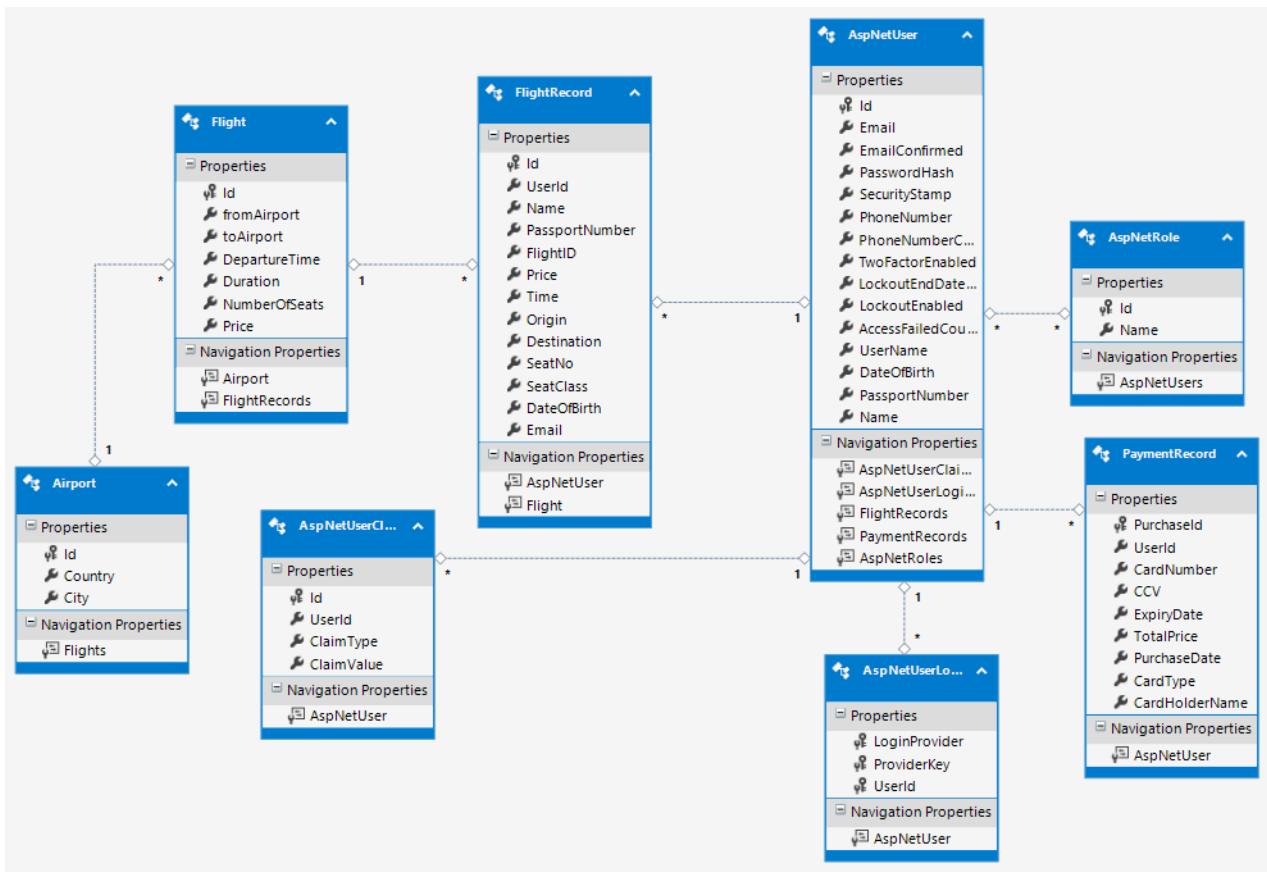


Figure 2. Entity Data Model

3.1.3 Sequence Diagram

The sequence diagrams below depict the various scenarios as defined in the use case using the Model-View-Controller (MVC) architecture.

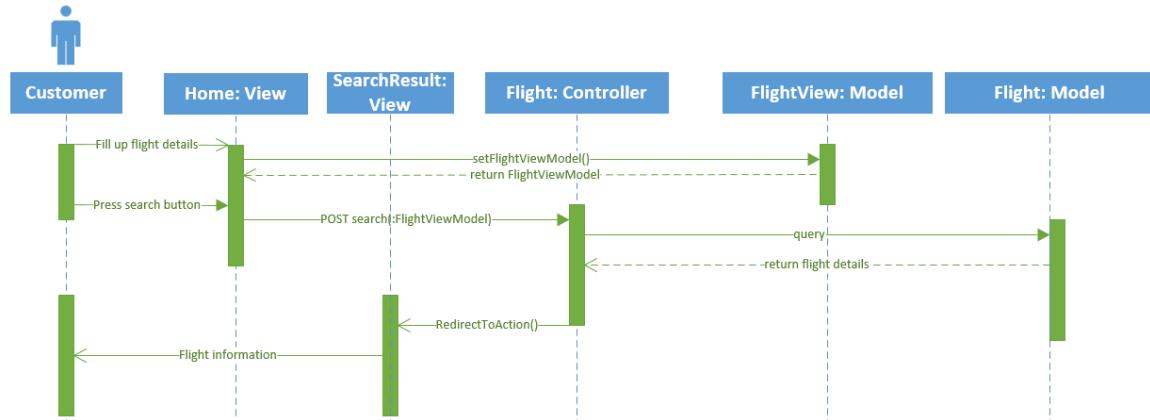


Figure 3. Search Flight Sequence Model

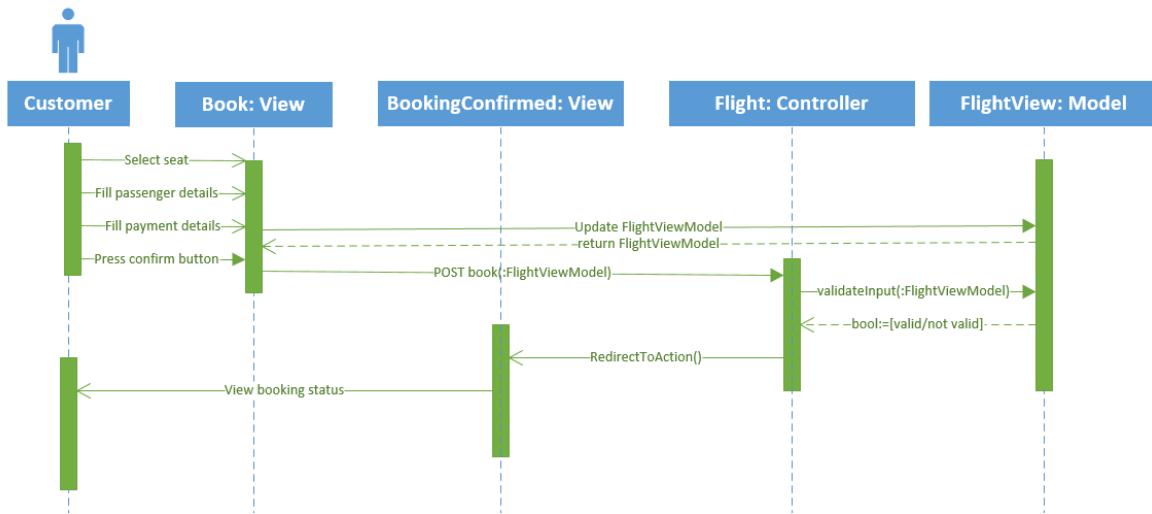


Figure 4. Booking Sequence Model

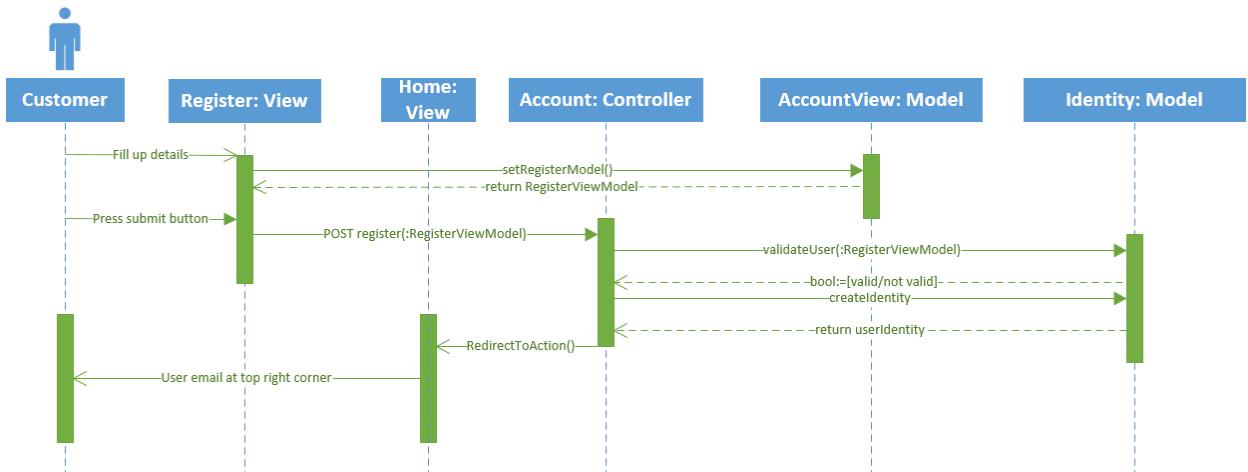


Figure 5. Register Sequence Model

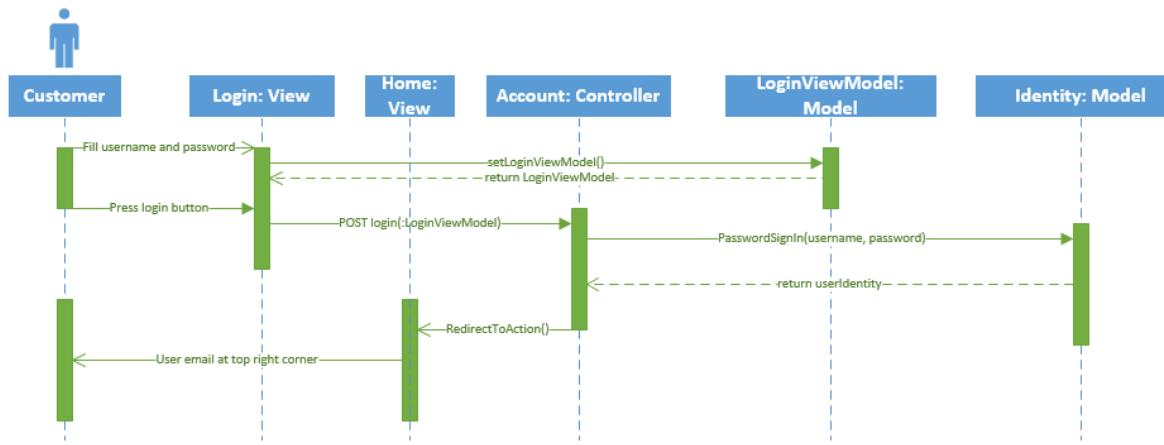


Figure 6. Login Sequence Model

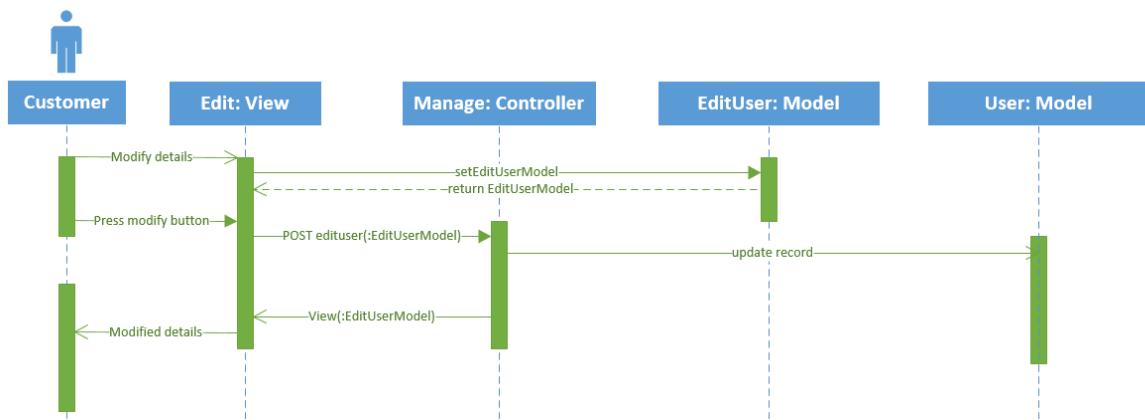


Figure 7. Edit User Sequence Model

3.1.4 View Map

The diagram below depicts the overall structural mapping of the various views within the proposed MVC architecture. It displays 14 view (web pages) that have been proposed to be implemented in the project.

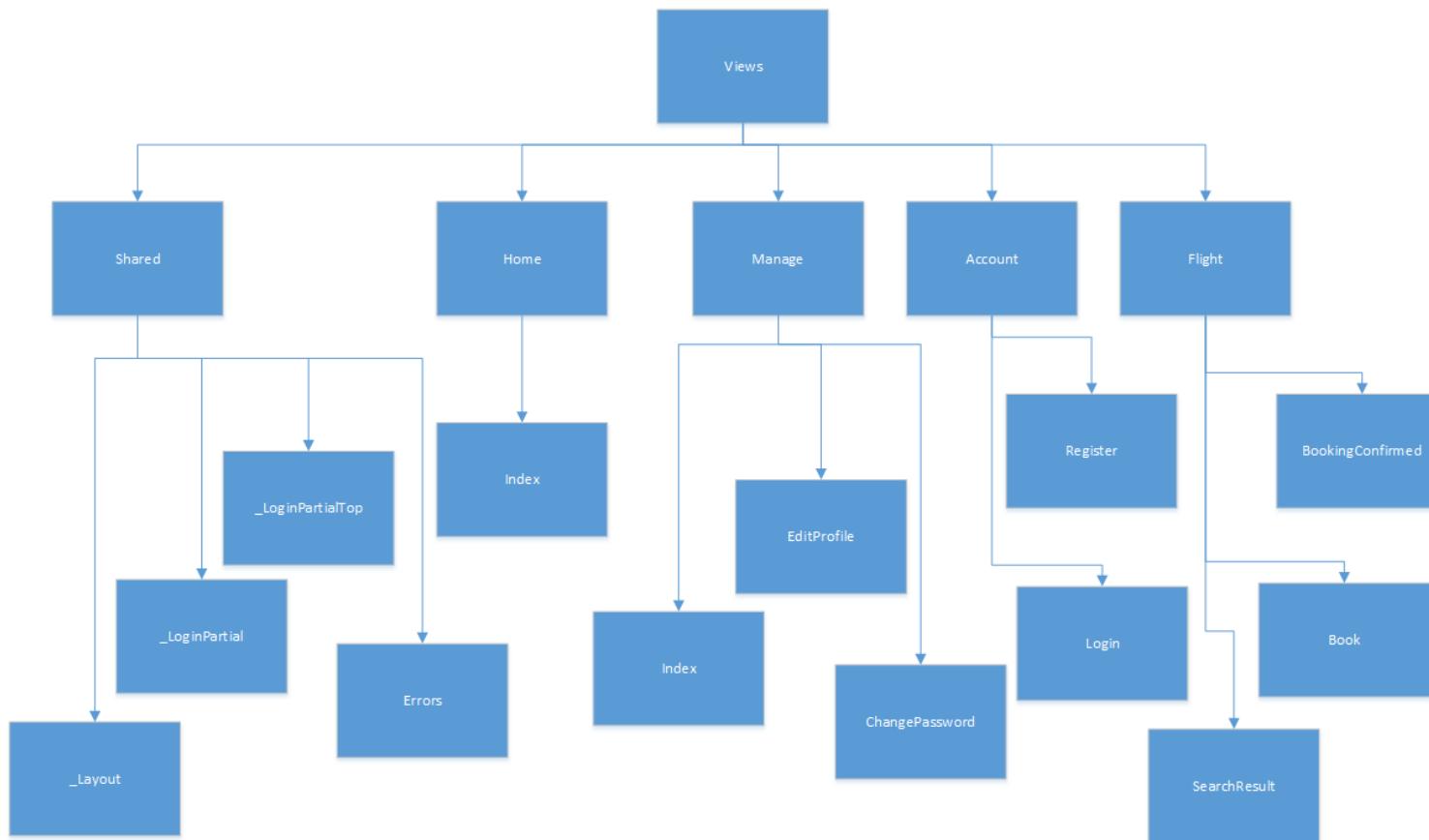


Figure 8. MVC View Map

3.2 Interface Design (UI Mockup)

The figures below illustrate the various abstract interface mockups for the web application that is to be developed. This guides the developer with the ability to contextualize the user requirements based on the design.

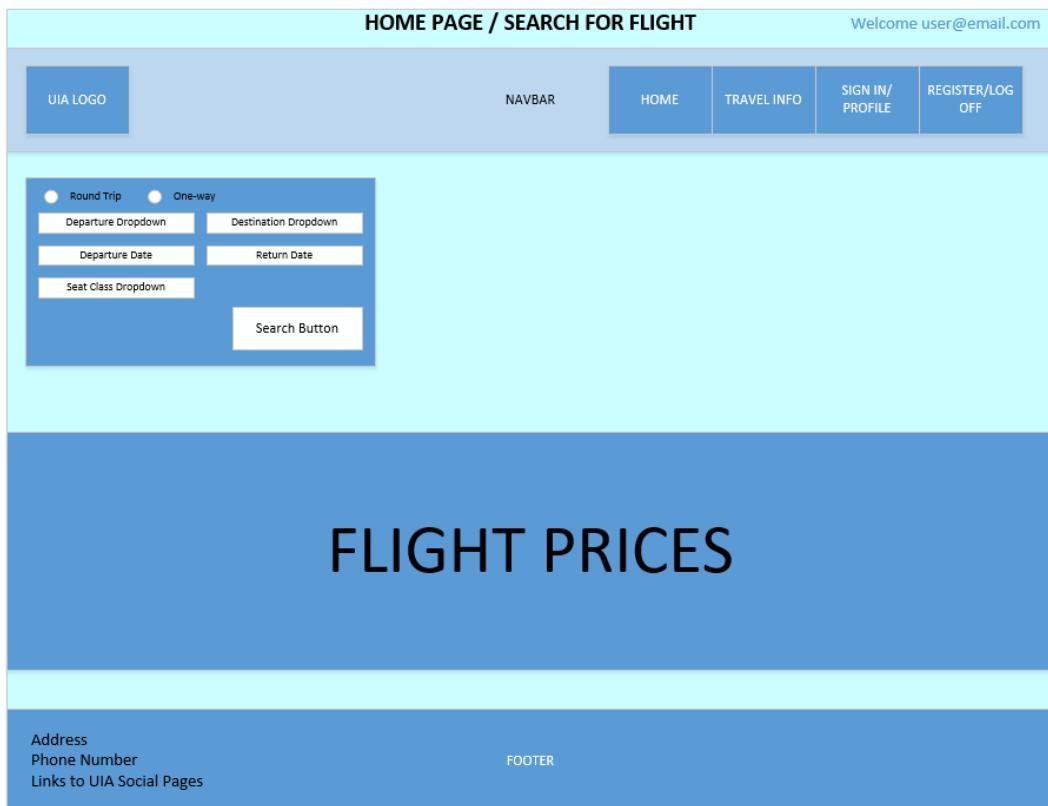


Figure 9. Homepage/Search for flight Mockup

SEARCH RESULT PAGE

Welcome user@email.com

UIA LOGO NAVBAR HOME TRAVEL INFO SIGN IN/ PROFILE REGISTER/LOG OFF

FromCity (Airport) to ToCity (Airport)
Seat Class

Departure Date Return Date

From Airport Duration To Airport Arrival Time PRICE
Outbound Flight

From Airport Duration To Airport Arrival Time PRICE
Return Flight

Book Button Search Again

Address
Phone Number
Links to UIA Social Pages

FOOTER

Figure 10. Search Result Page Mockup

LOGIN PAGE

Welcome user@email.com

UIA LOGO NAVBAR HOME TRAVEL INFO SIGN IN/ PROFILE REGISTER/LOG OFF

LOGIN FORM

Username
Password

Login

Address
Phone Number
Links to UIA Social Pages

FOOTER

Figure 11. Login Page Mockup

REGISTER PAGE

Welcome user@email.com

UIA LOGO NAVBAR HOME TRAVEL INFO SIGN IN/ PROFILE REGISTER/LOG OFF

Register Form

Name
Email
Passport Number
Phone Number
Date of Birth
Password
Confirm Password
Register

Address
Phone Number
Links to UIA Social Pages

FOOTER

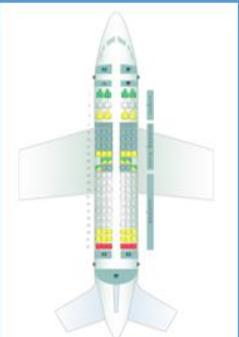
Figure 12. Register Page Mockup

BOOKING PAGE (1st third of the page)

Welcome user@email.com

UIA LOGO NAVBAR HOME TRAVEL INFO SIGN IN/ PROFILE REGISTER/LOG OFF

Fill your booking details
Select your seat



Address
Phone Number
Links to UIA Social Pages

FOOTER

Figure 13. Booking Page Mockup (1/3)

BOOKING PAGE (2nd third of the page)

Welcome user@email.com

UIA LOGO
NAVBAR
HOME
TRAVEL INFO
SIGN IN/
PROFILE
REGISTER/LOG
OFF

Fill your booking details

Passenger Details

Name	Email
Passport Number	Phone Number
Date of Birth	

Flight Details

Departure City	Arrival City	Departure Date & Time
Return Departure City	Return Arrival City	Departure Date & Time
Seat No	Price	

Address
Phone Number
Links to UIA Social Pages

FOOTER

Figure 14. Booking Page Mockup (2/3)

Welcome user@email.com

UIA LOGO
NAVBAR
HOME
TRAVEL INFO
SIGN IN/
PROFILE
REGISTER/LOG
OFF

Fill your booking details

Payment Details

Card Type	Card Number
Card Holder Name	CCV
Expiry Date	

CONFIRM

Address
Phone Number
Links to UIA Social Pages

FOOTER

Figure 15. Booking Page Mockup (3/3)

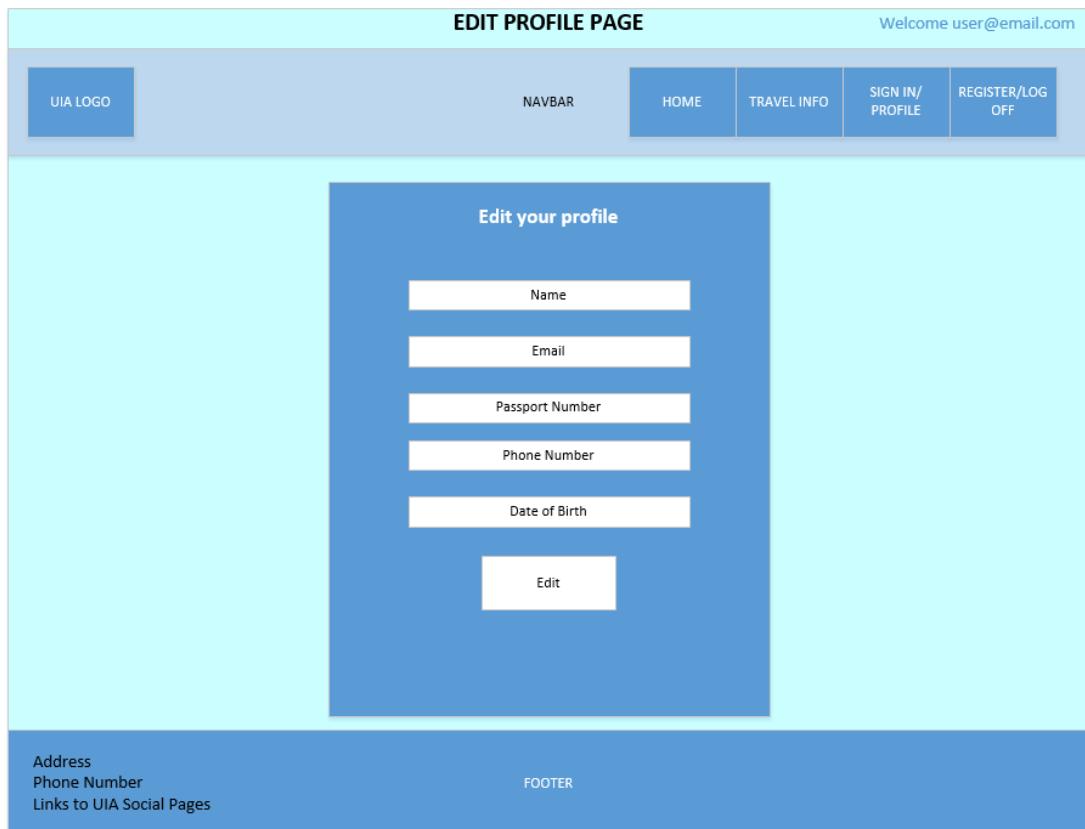


Figure 16. Edit Profile Page Mockup

3.3 Cloud Architecture Diagram

The architectural diagram below depicts the various elements that create the overall cloud architecture for this project. It includes two app services (UIA-SEA and UIA-West-EU) which hold the web applications and their associated service plans (S1). There are two Azure Storage blob containers, one holding the SQL auditing logs for security measurements, and the other the web configuration needed to implement the External Configuration Store Cloud Design Pattern. These are located within the same resource group as the Azure SQL Database, which contains the customer information. Furthermore, application insights was used to collect, analyze, and investigate both the performance metrics and failed requests (with raised exceptions) across the SEA web app. Lastly, the Traffic Manager has two registered endpoints, which are the aforementioned web applications, which have been enabled for continuous deployment through Git.

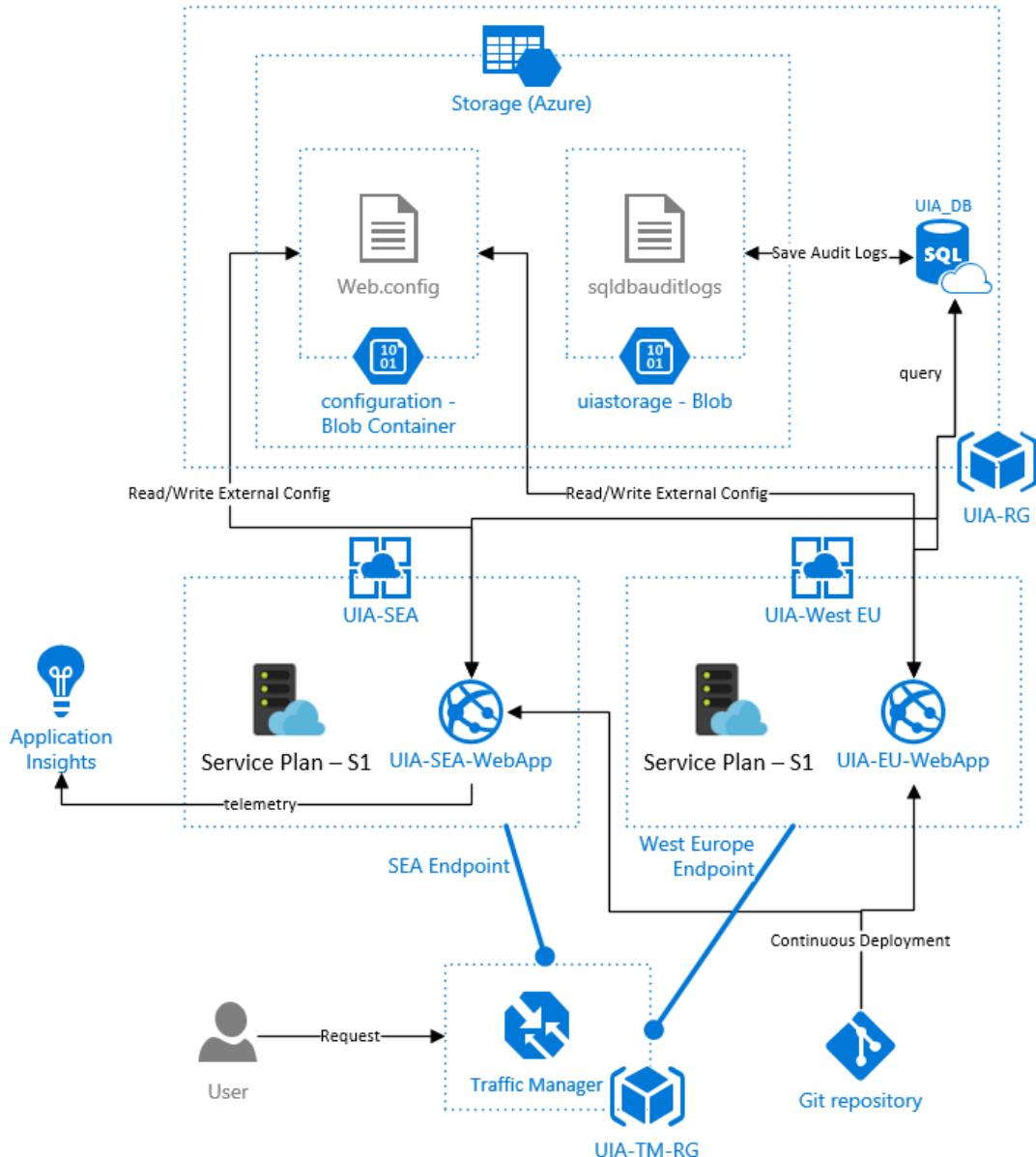


Figure 17. Proposed Cloud Architecture Diagram

4.0 Implementation

4.1 Core Application Implementation

4.1.1 External Configuration Store (Cloud Design Pattern)

Majority of web applications tend to embed their configuration information within the files that are deployed during the deployment phase. This often creates a hassle when changes occur to the configuration information. For instance, the migration of an SQL database from one server to another, which requires a change to the connection strings within all of the deployed web applications. External Configuration Store is a cloud design pattern that attempts to migrate the web configuration out of an application's deployment package to a centralized location (Narumoto, et al., 2017). This provides an interface through which the configuration settings of the web applications can be quickly and efficiently read, updated, and managed. This pattern is often used in various scenarios that add towards its suitability of implementation for this project, which are:

- Configuration settings are shared between various web applications. For this project, as the web application is deployed in two regions, it is necessary to ensure that they have the same web configuration that stores the database connection strings.
- Simplify the administration of various web applications, as each web application's configuration doesn't need to be manually updated as changes occur to the configuration. For this project, during the development phase, the database was stored locally, therefore its connection string was pointing to a local database. As the SQL database got deployed to Azure, the connection string needed to be changed, therefore this pattern provided an easy and efficient transition between these two stages. Furthermore, if there is a change to the database, such as moving it to another server or changing it into an Azure Table Storage, this needs to be appropriately reflected within all the deployed web applications. The image below shows an overall illustration of this cloud design pattern.

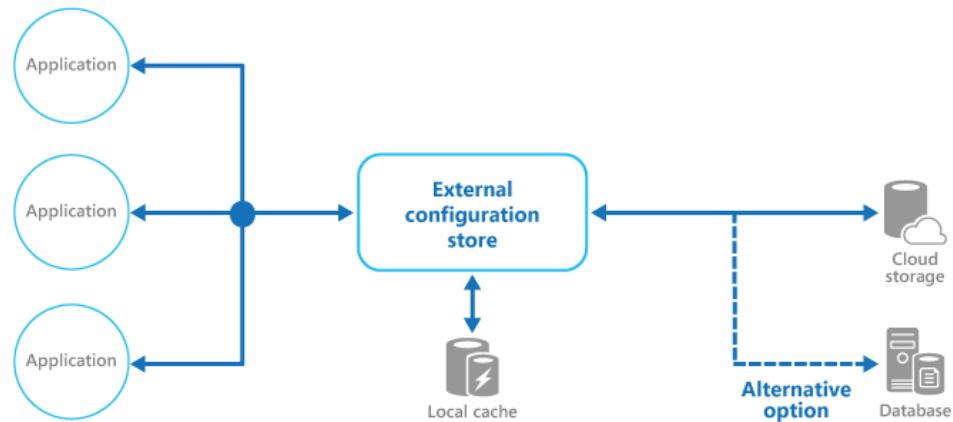


Figure 18. External Configuration Store Diagram (Narumoto, et al., 2017)

For this project, an Azure Blob Storage had been created that consisted of a container called “configuration”. This container held the web.config file from the web application, which was uploaded using the code snippet written below.

```
- references | Kazem, 1 day ago | 1 author, 1 change | 0 exceptions
private void UploadConfigurationBlob()
{
    var account = CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("externalconnectionuia_AzureStorageConnectionString"));
    var client = account.CreateCloudBlobClient();
    var container = client.GetContainerReference(configContainer);
    container.CreateIfNotExists();

    var productionBlob = container.GetBlockBlobReference(ConfigBlobNameApp);
    productionBlob.UploadFromFile(ConfigBlobNameApp);
}
```

Figure 19. Upload Config Blob - Code Snippet

Each time the config checker is executed, the connection strings within the deployed web.config is updated to ensure that the latest connection strings are being utilized, as shown below.

```
public void Run()
{
    ExternalConfiguration.Instance.ConfigChangeMonitor();
    System.Configuration.Configuration conf = WebConfigurationManager.OpenWebConfiguration("/");
    conf.ConnectionStrings.ConnectionStrings["UIADbContext"].ConnectionString = ExternalConfiguration.Instance.GetAppSetting("UIADbContext");
    conf.ConnectionStrings.ConnectionStrings["UIA_Entities"].ConnectionString = ExternalConfiguration.Instance.GetAppSetting("UIA_Entities");
    conf.Save();
}
```

Figure 20. Config Checker Run - Code Snippet

4.1.2 Security and Authentication

A core concern revolving around the implementation and usage of web applications is the security and authentication necessary for the forms and view-state. To address this, Microsoft has developed ASP.NET Identity system to replace their membership and simple membership systems (Anderson, et al., 2017). This is uses 128-bit AES for the encryption,

decryption, and validation of the forms-authentication and view-state data, which does not have any known practical attacks (Anderson, et al., 2017).

Several notable features of ASP.NET Identity have been implemented in this web project. Firstly, by utilizing the Anti-forgery token, which prevents Cross-site Request Forgery (CSRF) attacks, as shown below. For each form, an anti-forgery token is generated, and validated in the controller.

```
~~~~~
[HttpPost]
[ActionName("SearchResult")]
[ValidateAntiForgeryToken]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public ActionResult SearchResultPage(SearchFlightModel model){...}

@using (Html.BeginForm("SearchResult", "Flight", FormMethod.Post, new { @class = "small", role = "form" }))
{
    @Html.AntiForgeryToken()
}
```

Figure 21. Anti-forgery token Generation and Validation in View and Controller

Additionally, it provides the Authorize and AllowAnonymous tags, which dictate the access to certain controllers based on the level of authentication. For instance, AllowAnonymous indicates that a controller is accessible without user-authentication, while authorize requires the user to be logged in and meet the authentication requirements, as shown below.

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public async Task<ActionResult> Login(LoginViewModel model, string returnUrl){...}

[Authorize]
2 references | 0 changes | 0 authors, 0 changes
public class ManageController ...
```

Figure 22. Authorize and AllowAnonymous Tags

Furthermore, it supports plugging new profile data, by providing full control over the schema for the user and profile information, such as allowing the users to store their passport number and date of birth while registering. For this project, the Entity Framework code-first approach was used to migrate the changes to the user profile (adding phone number, name, passport number, and date of birth). Lastly, by integrating OWIN (Open Web Interface for .NET), it allows for the cookie-based user authentication (log in/out), as shown below.

```

public ApplicationSignInManager SignInManager
{
    get
    {
        return _signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
    }
    private set
    {
        _signInManager = value;
    }
}
public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    if (returnUrl == null)
    {
        returnUrl = "/Home/Index";
    }
    // This doesn't count login failures towards account lockout
    // To enable password failures to trigger account lockout, change to shouldLockout: true
    var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);
}

```

Figure 23. OWIN-based Authentication - ASP.NET Identity

4.1.3 Dynamic Seat Picker

A primary feature of most modern flight booking web applications are the ability to provide flexibility of use to the customer to improve the usability by allowing users to pick their seats in advance during the booking process. This is achievable in many ways; however, it should be tailored such that it can maximize the user experience and ease of use for users with varying mental models. To achieve this, a dynamic seat picker has been implemented using SVG and JavaScript from AnyChart, a flexible HTML-based solution provider for building interactive maps, including seat maps (AnyChart, 2017). It has been modified to allow the script to update the respective seat number fields after the customer has chosen a seat on the SVG image, as shown below.

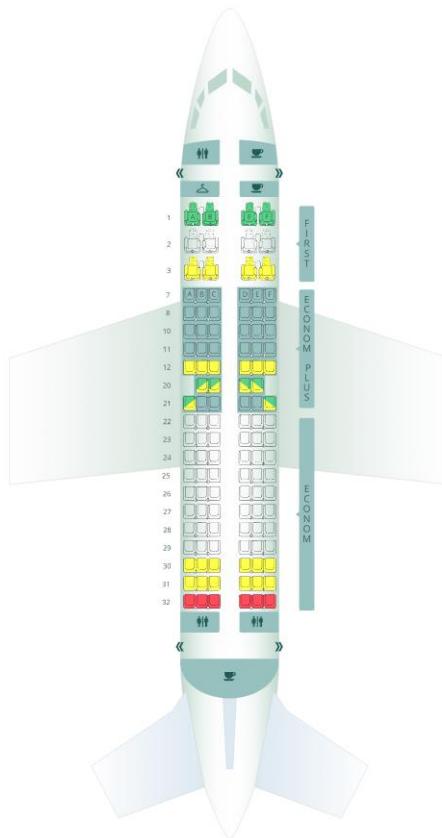


Figure 24. Boeing 737 SVG Seat Layout (AnyChart, 2017)

```
<div class="section-top-80 section-md-top-60">
    <h3 class="text-bold text-capitalize text-center">Select your @Model.Seat_Class seat</h3>
    <h4 class="offset-top-20 text-center" id="seat"></h4>
    <div id="container"></div>
</div>
```

Figure 25. Div Tag to hold Seat SVG graph and Selected Seat - Code Snippet

The code snippet below describes the JavaScript that loads the SVG image within the container div tag.

```
<script src="https://cdn.anychart.com/js/7.14.0/anychart-bundle.min.js"></script>
<script src="https://code.jquery.com/jquery-latest.min.js"></script>
<script src="https://cdn.anychart.com/csv-data/boeing_737.js"></script>
<script type="text/javascript">
    anychart.onDocumentReady(function () {
        stage = acgraph.create('container');
        // get svg file
        $.ajax({
            type: 'GET',
            url: 'https://cdn.anychart.com/svg-data/seat-map/boeing_737.svg',
            success: function (svgData) {
                // from the CDN https://cdn.anychart.com/csv-data/boeing_737.js to data file

```

Figure 26. Generate and hold SVG graph – Code Snippet

The code below describes the action listener that updates the seat number field when the user selects a seat on the SVG image (pointsSelect function).

```
// add pointsSelect listener to get select place info
chart.listen('pointsSelect', function (points) {
    var placesInfo = points.seriesStatus[0].points;
    var placesId = [];
    var totalPrice = 0;
    if (chart.getSelectedPoints().length) {
        for (var i = 0; i < placesInfo.length; i++) {
            placesId.push(points.seriesStatus[0].points[i].id);
            totalPrice += parseInt(placeInfoFunc(points.seriesStatus[0].points[i].id).price);
        }
        $("#seat").html("You've chosen: " + placesId[0]);
        document.getElementById("seat_no").value = placesId[0];
    }
});

});
```

Figure 27. Update Selected Seat - Code Snippet

4.1.4 GeoIP-based Currency

An important aspect of consideration regarding the design and implementation of the web application is localization. As UIA provides international flights, this will naturally resonate into a business market that consists of customers from varying countries. One way to achieve localization, which will ultimately improve the user experience, is through the adaptation of varying currencies based on the customer location.

Web applications work through requests. A client sends a request, which within an MVC architecture will be passed into the controller as an `HttpRequest` object (LeBlanc, et al., 2017). This object has various server variables that can be used to identify the IP address of the user. The request headers can be spoofed or be left empty. Furthermore, many users might be behind a proxy server (Forsyth, 2010). These can alter the server variables (returning null or wrong IP address). This falls out of the context of the implementation, as it defeats the purpose of localization as the users do not wish to expose their actual public IP address. Either way, the IPv4 address can either be detected through the `REMOTE_ADDR` element or `UserHostAddress` attribute, as shown below.

```
public string GetIPAddress()
{
    string ipAddress = HttpContext.Request.ServerVariables["REMOTE_ADDR"];
    if (!string.IsNullOrEmpty(ipAddress))
    {
        return ipAddress;
    }
    return HttpContext.Request.UserHostAddress;
}
```

Figure 28. User IP Address (v4) based on Request variables

After attaining the user's IP address, the location of an endpoint can be detected based on the IP address by querying a GeoIP database, such as MaxMind's GeoIP databases. However, as these often require the databases (free or paid) to be locally stored and queried, it is easier to query through an API that provides free GeoIP services, such as freegeoip.net, which provides up to 15,000 queries per hour from a single host. It provides the ability to parse a receiving response stream into a JSON object that holds all the different key/value pairs for the geographic location of the submitted IP address (Fiori, 2017), as shown below. However, we are mainly concerned with the `country_code` and `country_name` attributes, which for instance would return "MY" and "Malaysia" for an IP address originating from Malaysia, respectively.

```
public static IPlocation GetIPLocation(string IPAddress)
{
    IPlocation iplocation;
    string retJson = DownloadDataNoAuth(string.Format("http://www.freegeoip.net/json/{0}", IPAddress));
    var JO = JObject.Parse(retJson);
    iplocation = new IPlocation();
    iplocation.IPAddress = IPAddress;
    iplocation.CountryCode = JO["country_code"].ToString();
    iplocation.CountryName = JO["country_name"].ToString();
    return iplocation;
}
```

Figure 29. GetIPLocation - Code Snippet

Once the country code has been extracted, it can be compared to the varying country codes and the corresponding currency can be shown. The current implementation only supports Euro for majority of the European nations, Malaysian Ringgit for Malaysia, otherwise American Dollars is used, as shown below.

```
var euroCountries = new List<string>();
euroCountries.AddRange(new String[] { "DE", "DK", "SE", "IT", "NL", "PL", "NO", "FI" });
if (iplocation.CountryCode == "MY") {
    System.Web.HttpContext.Current.Items["currency"] = "RM ";
    currency = "RM ";
}
else if(euroCountries.Contains(iplocation.CountryCode))
{
    System.Web.HttpContext.Current.Items["currency"] = "€ ";
    currency = "€ ";
}
else
{
    System.Web.HttpContext.Current.Items["currency"] = "$ ";
    currency = "$ ";
}
```

Figure 30. Currency based on IP - Code Snippet

The screenshot displays the UIA flight booking system's search interface. On the left, there's a large image of the Eiffel Tower. To its right, the search parameters are set to "Paris" as the departure city and "France" as the destination country. Below this, a list of departure cities and their corresponding flight prices in Ringgit (RM) are shown:

Departure City	Price (RM)
Kiev	98.00
Kuala Lumpur	134.00
London	119.00
Tehran	159.00

To the right of the UI, a separate browser window shows the JSON output of a geolocation API call. The JSON object contains the following fields for a user in Malaysia:

```

{
  "ip": "██████████",
  "country_code": "MY",
  "country_name": "Malaysia",
  "region_code": "10",
  "region_name": "Selangor",
  "city": "Semenyih",
  "zip_code": "43500",
  "time_zone": "Asia/Kuala_Lumpur",
  "latitude": 2.9516,
  "longitude": 101.843,
  "metro_code": 0
}
  
```

Figure 31. User Request from Malaysia - Ringgit Currency

The screenshot shows the same UIA search interface as Figure 31, but the currency has been changed to Euro (€). The flight prices are now listed in Euros (€) instead of Ringgit (RM). The JSON geolocation response for a user in Finland is as follows:

```

{
  "ip": "██████████",
  "country_code": "FI",
  "country_name": "Finland",
  "region_code": "18",
  "region_name": "Uusimaa",
  "city": "Helsinki",
  "zip_code": "00300",
  "time_zone": "Europe/Helsinki",
  "latitude": 60.1756,
  "longitude": 24.9342,
  "metro_code": 0
}
  
```

Figure 32. User Request from Finland - Euro Currency

It is important to note that the current implementation does not take into consideration the currency value-conversion, as it is outside the scope of the current project.

4.1.5 Developed Web Pages

The design of the developed web pages were based on a web template. The images below illustrate the various implemented web page designs.

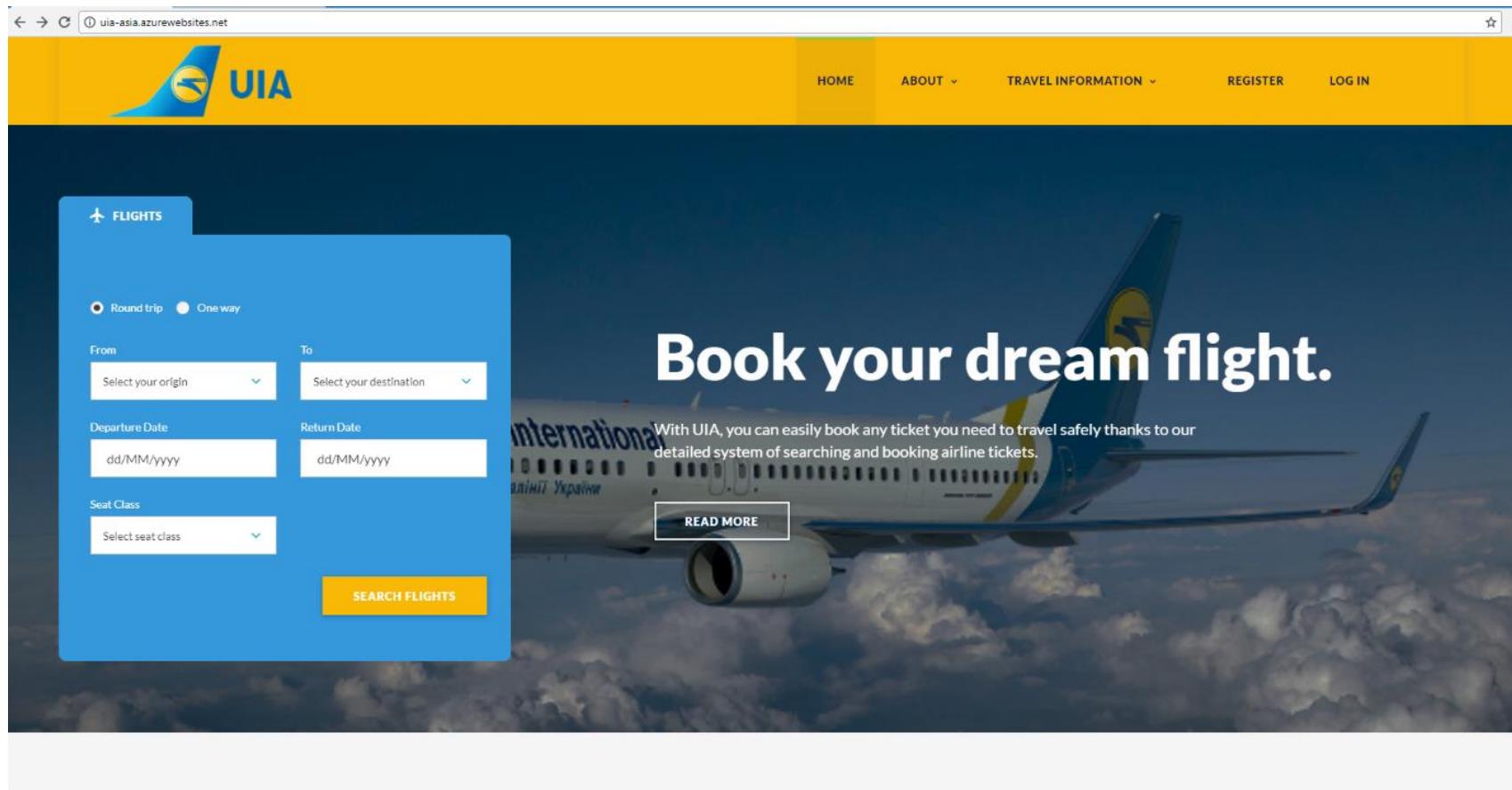


Figure 33. Homepage Implemented Design

The screenshot shows the 'Available Flights' page of the UIA Online Flight Booking System. The page has a yellow header with the UIA logo and navigation links for HOME, ABOUT, TRAVEL INFORMATION, REGISTER, and LOG IN. Below the header, the title 'Available Flights' is centered. The page displays three flight options in cards:

- Paris** (France)
Departure City:
 - Kiev RM 98.00
 - Kuala Lumpur RM 134.00
 - London RM 119.00
 - Tehran RM 159.00
- London** (United Kingdom)
Departure City:
 - Kiev RM 98.00
 - Paris RM 134.00
 - Kuala Lumpur RM 119.00
 - Tehran RM 159.00
- Kuala Lumpur** (Malaysia)
Departure City:
 - Kiev RM 98.00
 - Paris RM 134.00
 - London RM 119.00
 - Tehran RM 159.00



Figure 34. Available Flights Implemented Design

The screenshot shows the 'Register' page of the UIA Online Flight Booking System. At the top, there is a navigation bar with links for HOME, ABOUT, TRAVEL INFORMATION, REGISTER, and LOG IN. The 'REGISTER' link is highlighted. On the left, there is a logo for 'UIA' with a blue and white swoosh graphic. Below the navigation bar, there are two buttons: 'Login' (white background) and 'Register' (yellow background). The main form consists of several input fields: 'Full Name' (text input), 'Email' (text input), 'Passport/IC No.' (text input), 'Birth Date' (text input with placeholder 'mm/dd/yyyy'), 'Phone No.' (text input), 'Password' (text input), and 'Confirm password' (text input). A large yellow 'REGISTER' button is located at the bottom of the form.

Figure 35. Register Page Implemented Design

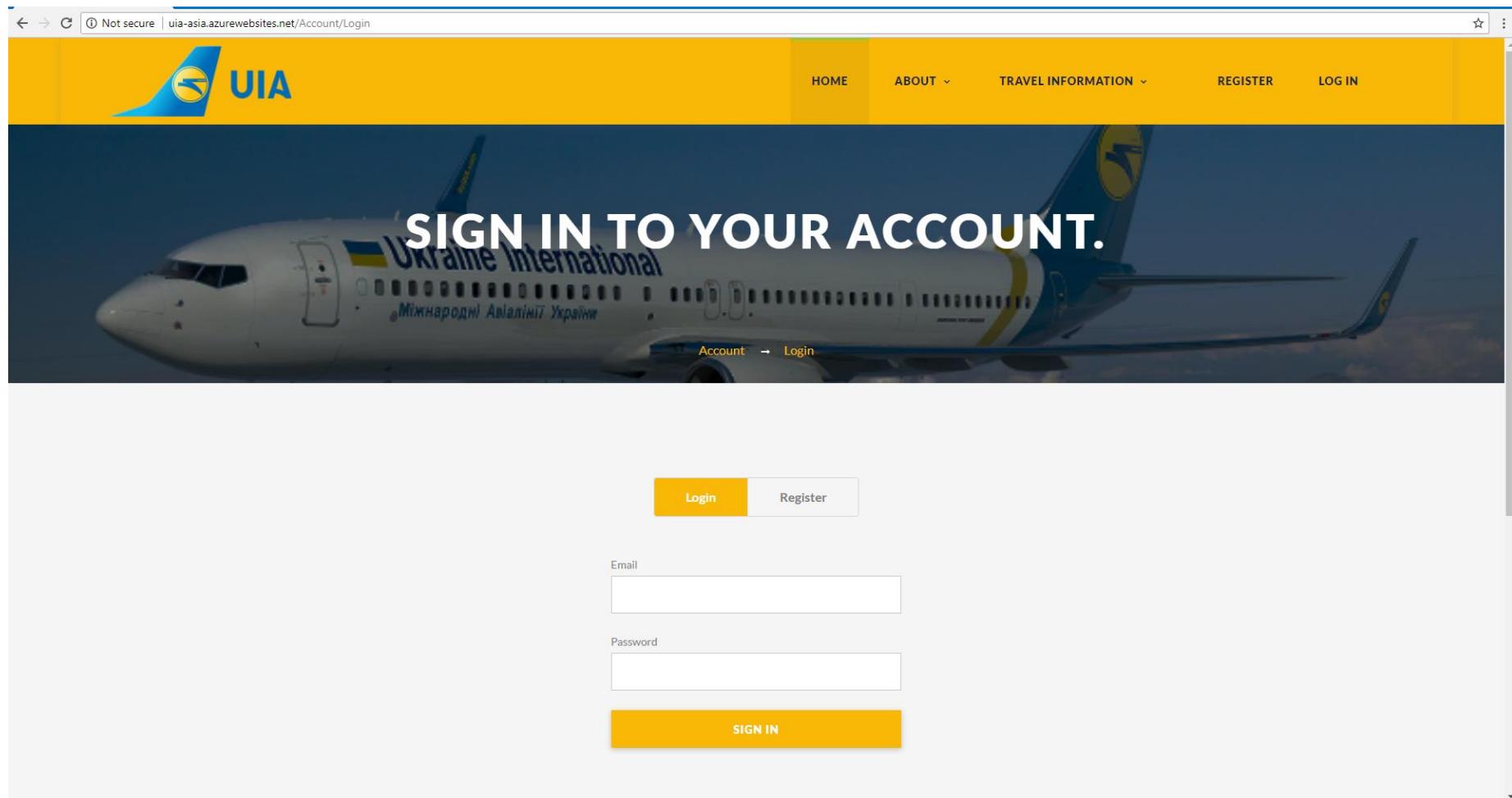


Figure 36. Login Page Implemented Design

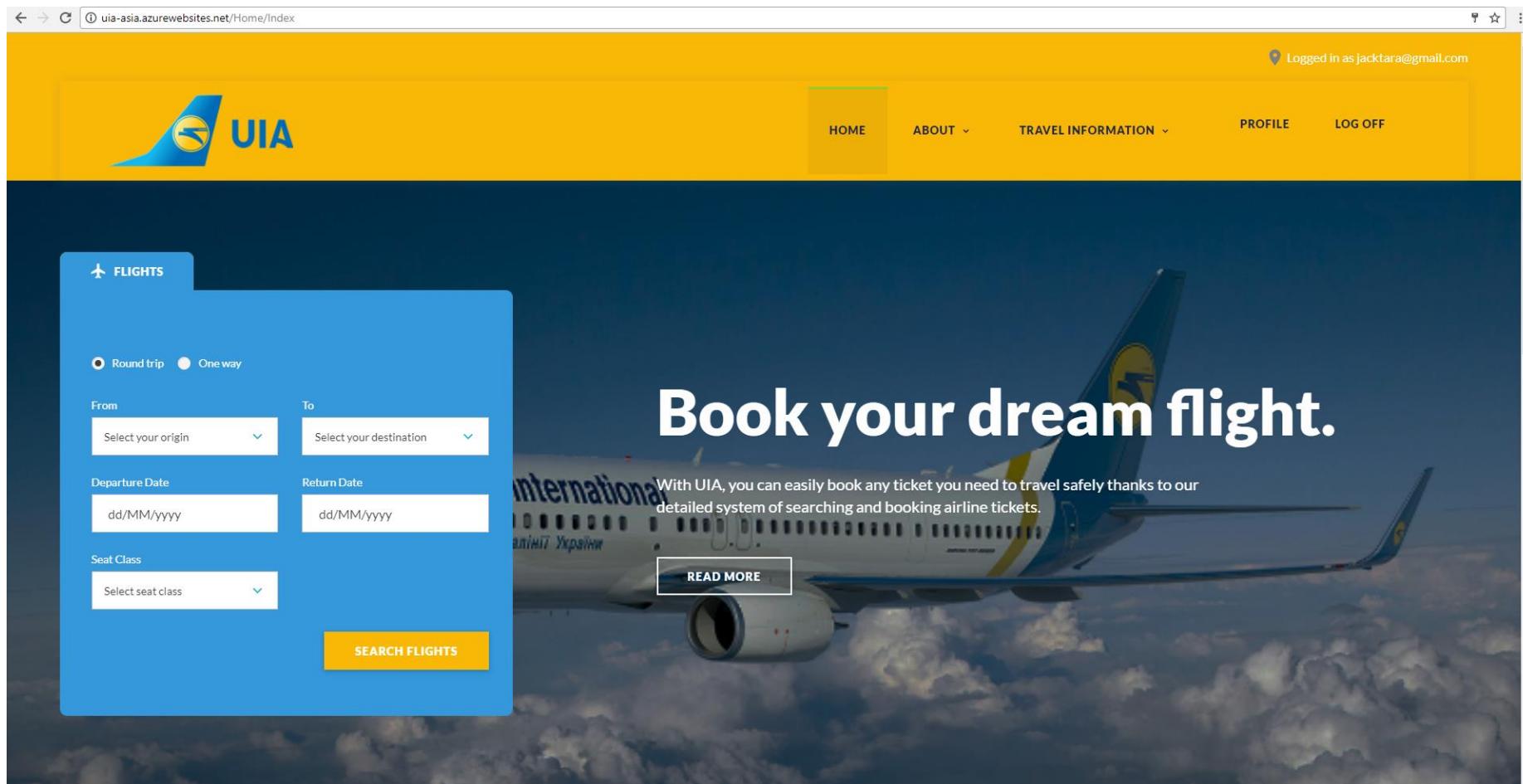


Figure 37. Homepage with logged in user

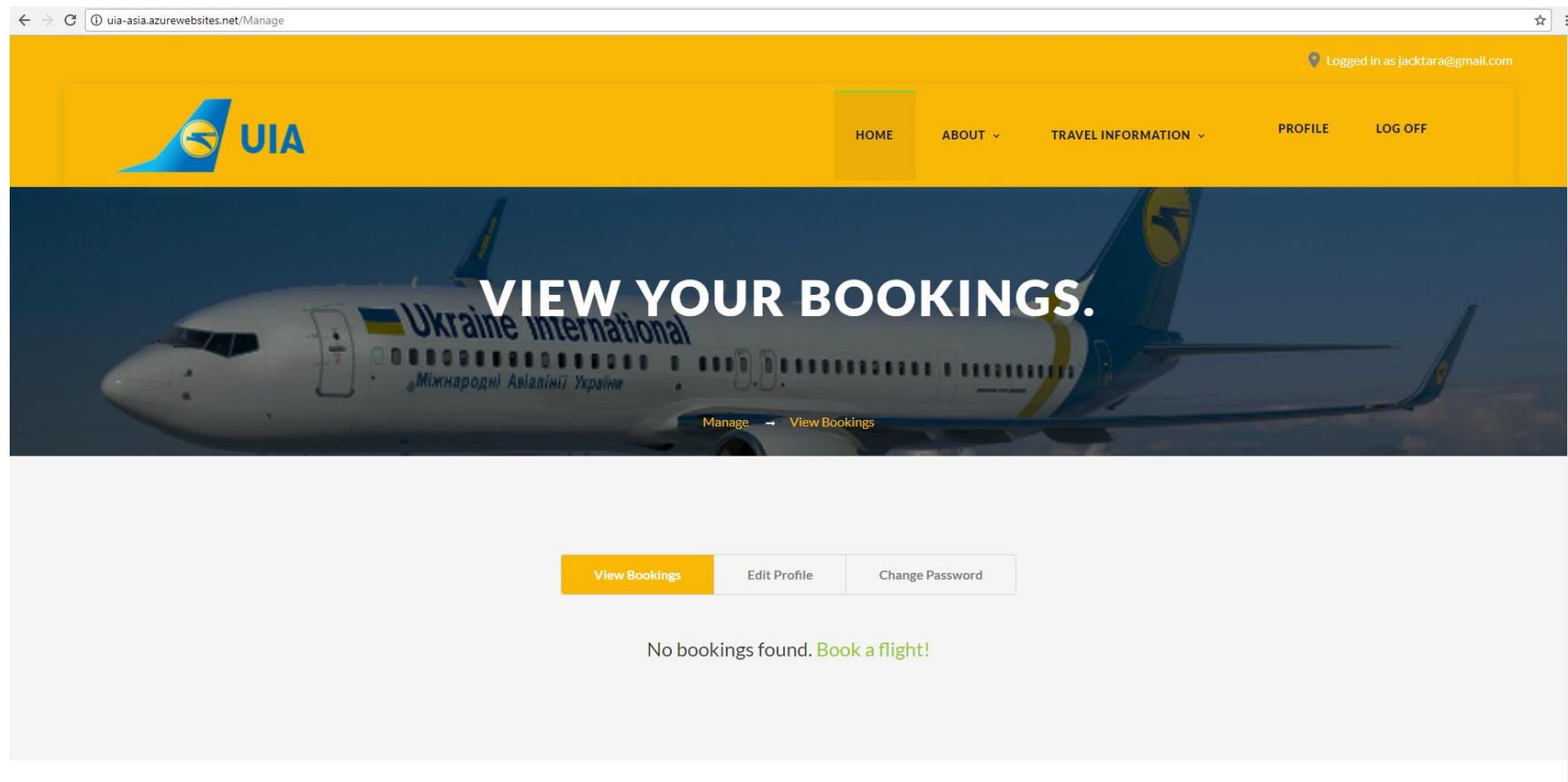


Figure 38. View Bookings Implemented Design - No bookings

The screenshot shows the 'Edit Profile' page of the UIA Online Flight Booking System. The page has a yellow header with the UIA logo and navigation links for HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF. The main content area contains fields for Name, Email, Passport/IC No., Birth date, and Phone No., each with a corresponding input field. A 'View Bookings' button is also present. The 'Edit Profile' button is highlighted in yellow, indicating it is the active tab. A 'SAVE CHANGES' button is located at the bottom of the form.

View Bookings **Edit Profile** Change Password

Name
Jack Tara

Email
jacktara@gmail.com

Passport/IC No.
H9242431

Birth date
29/09/2017

Phone No.
01338617423

SAVE CHANGES

Figure 39. Edit Profile Implemented Design

The screenshot shows a web browser window for the UIA Online Flight Booking System. The URL in the address bar is <https://uii-asia.azurewebsites.net/Manage/ChangePassword>. The page has a yellow header with the UIA logo and navigation links: HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF. Below the header is a large image of an airplane with the UIA logo. The main content area is titled "Manage" and "Change Password". It features three input fields: "Current password", "New password", and "Confirm new password", each with a corresponding text input box. A yellow "CHANGE" button is at the bottom. The top navigation bar includes "View Bookings" and "Edit Profile" links.

Contact Us

Figure 40. Change Password Implemented Design - Profile

Kuala Lumpur (KLIA) to Kiev (KBP)

with return flight.

First Class

Departure Date: 06/10/2017 Return Date: 20/10/2017

Outbound Flight

UIA	06/10/2017 12:00 PM	KLIA	240 minutes	→	06/10/2017 04:00 PM	KBP	Flight Price	RM 430
Direct								

Return Flight

UIA	20/10/2017 03:00 AM	KBP	120 minutes	→	20/10/2017 05:00 AM	KLIA	Flight Price	RM 830
Direct								

BOOK **SEARCH AGAIN**

Figure 41. Search Result Page - Implemented Design

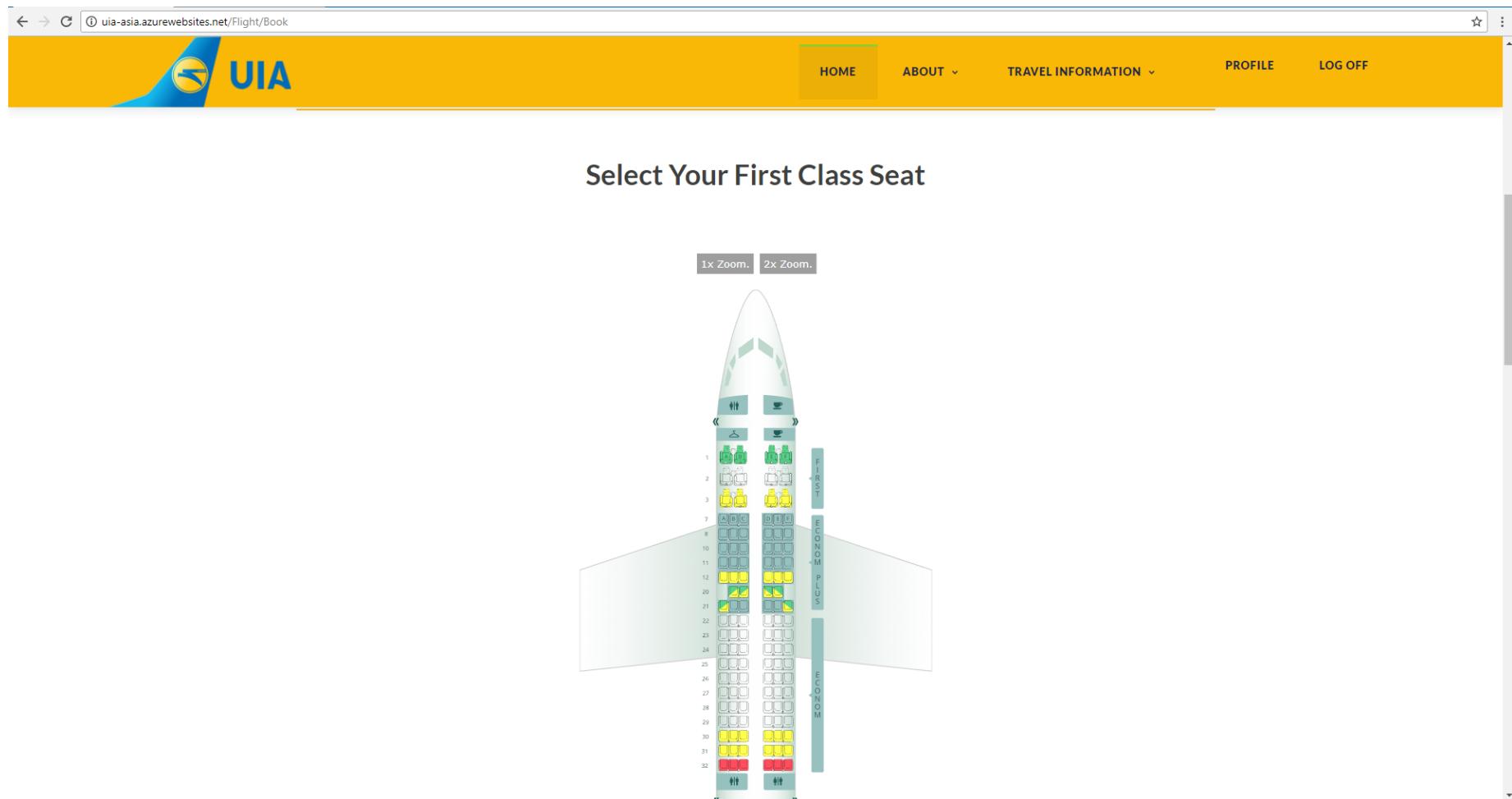


Figure 42. Book Flight Ticket Implemented Design - Part 1

The screenshot shows a web browser window for the UIA Online Flight Booking System. The URL in the address bar is uia-asia.azurewebsites.net/Flight/Book. The page has a yellow header with the UIA logo, navigation links for HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF, and a copyright notice for AnyChart Trial Version. Below the header is a graphic of a white airplane. The main content area is divided into two sections by a horizontal line: "Passenger Details" and "Flight Details".

Passenger Details

Full Name	Passport Number
Jack Tara	H9242431
Email	Phone Number
jacktara@gmail.com	01338617423
Date of Birth	
29/09/2017 00:00:00	

Flight Details

Figure 43. Book Flight Ticket Implemented Design - Part 2

The screenshot shows a web browser window for the UIA Online Flight Booking System. The URL in the address bar is uia-asia.azurewebsites.net/Flight/Book. The page has a yellow header with the UIA logo and navigation links for HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF. The main content area is divided into two sections: "Flight Details" and "Payment Details".

Flight Details

Origin City Kuala Lumpur	Destination City Kiev	Departure Time 06/10/2017 12:00:00
Return Trip Origin Airport Kiev	Return Destination City Kuala Lumpur	Return Trip Departure Time 20/10/2017 03:00:00
Seat Number 1-F	Total Price 1260	

Payment Details

Card Type Select your card type	Credit Card Number e.g. 5241-4241-4242-4214
Cardholder Name e.g. John Smith	CCV e.g. 949
Expiry Date e.g. 05/22	

Figure 44. Book Flight Ticket Implemented Design - Part 3

The screenshot shows a web browser window for the UIA website at uii-asia.azurewebsites.net/Flight/BookingConfirmed. The header features the UIA logo and navigation links for HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF. A large banner image of an Ukraine International Airlines Boeing 737 aircraft in flight is displayed, with the text "BOOKING STATUS." overlaid. Below the banner, a breadcrumb trail shows "Flight → Booking Status". The main content area displays a green success message: "Booking Successful! View your booking." On the left side, there is a "Contact Us" section with icons for location, phone, and email, and the address "201-203, Kharkivske Road. Nearest tube: Boryspilska." The bottom of the page includes a footer with social media icons for Facebook, Twitter, and YouTube.

Booking Successful! View your booking.

Contact Us

📍 201-203, Kharkivske Road.
Nearest tube: Boryspilska.

📞 +38 044 581 50 50

✉️ contact@flyuia.com

Figure 45. Booking Confirmed Page

The screenshot shows a web browser window for the URL uia-asia.azurewebsites.net/Manage. The page has a yellow header bar with the UIA logo on the left and navigation links for HOME, ABOUT, TRAVEL INFORMATION, PROFILE, and LOG OFF. Below the header is a search bar with placeholder text 'Search'. Underneath is a button labeled 'View Bookings' in yellow. To its right are links for 'Edit Profile' and 'Change Password'. The main content area displays a table titled 'Bookings' with columns: Reference ID, From, To, Time, and Seat No. The table contains four rows of booking information.

Reference ID	From	To	Time	Seat No.
407477eff8ae4824a97adcde134c42d0	Kuala Lumpur	Kiev	06/10/2017 12:00:00	1-F
61b6f2bdf1cc42a193cd64f465d1239e	Kiev	Kuala Lumpur	20/10/2017 03:00:00	1-F
8686bf0a6f4045d9a9da129b281f133e	Kiev	Kuala Lumpur	20/10/2017 03:00:00	1-F
92d7b81caf4047eca951ef8bf407040f	Kuala Lumpur	Kiev	06/10/2017 12:00:00	1-F

Contact Us

📍 201-203, Kharkivske Road.
Nearest tube: Boryspilska.

📞 +38 044 581 50 50

✉️ contact@flyuia.com

Figure 46. View Bookings Implemented Design

4.2 Cloud Implementation and Deployment

4.2.1 Cost Breakdown

The project is developed under the Visual Studio Dev Essentials package in Microsoft Azure, which provides RM 150 per month to be utilized. Therefore, it is important to identify the necessary resources and optimize the utilization of the resources to ensure that as much necessary features can be implemented within the limited monetary constraint. There are several core essential resources that have been identified, which are necessary for the project based on the set deliverables, which are:

- Web App (with its associated service plan)
- Azure SQL Database
- Azure Blob Storage
- Traffic Manager

Microsoft provides a price calculator for the provided Azure resources based on the needs. The cost has been broken down and analyzed using this tool. The figures below show the cost breakdown of each of the necessary resource groups based on the needs and chosen plans, the justification of which, will be discussed in the subsequent subsections.

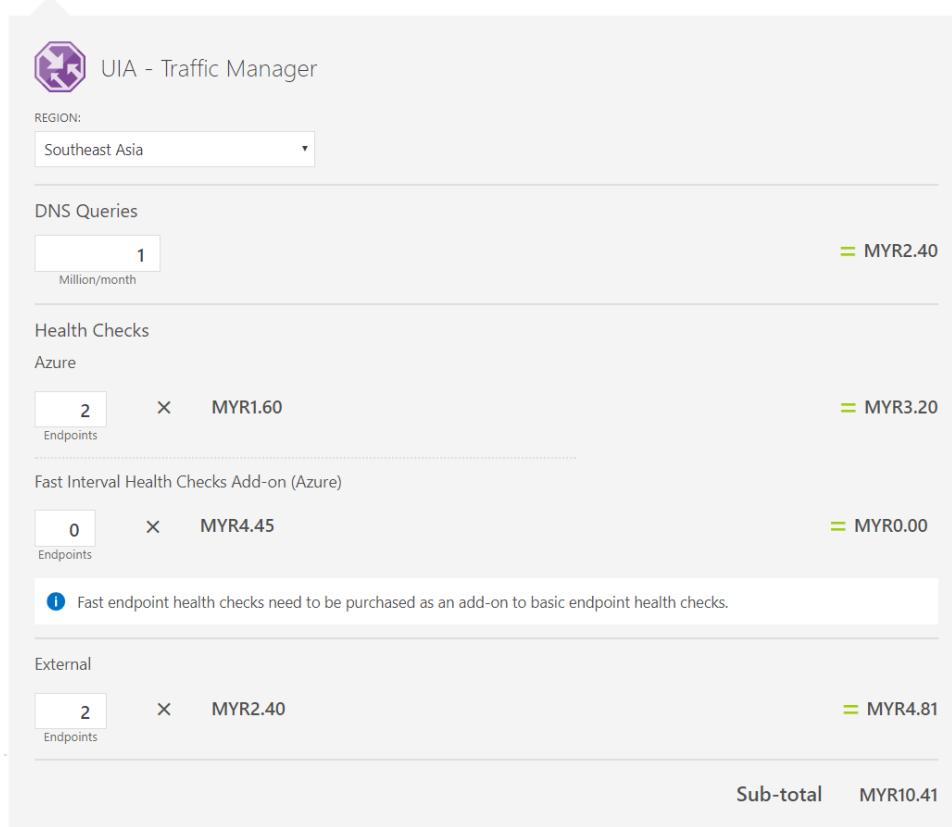
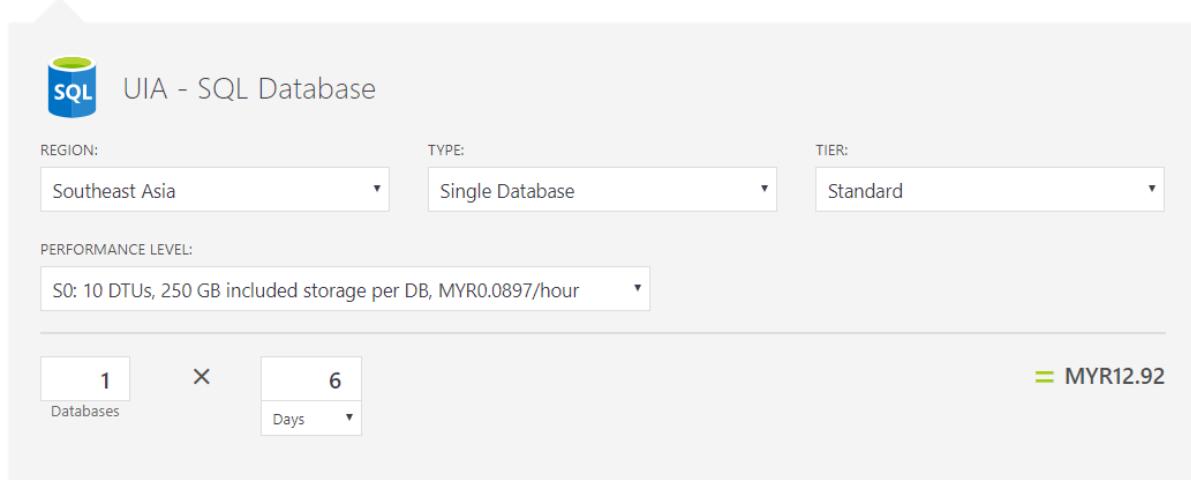
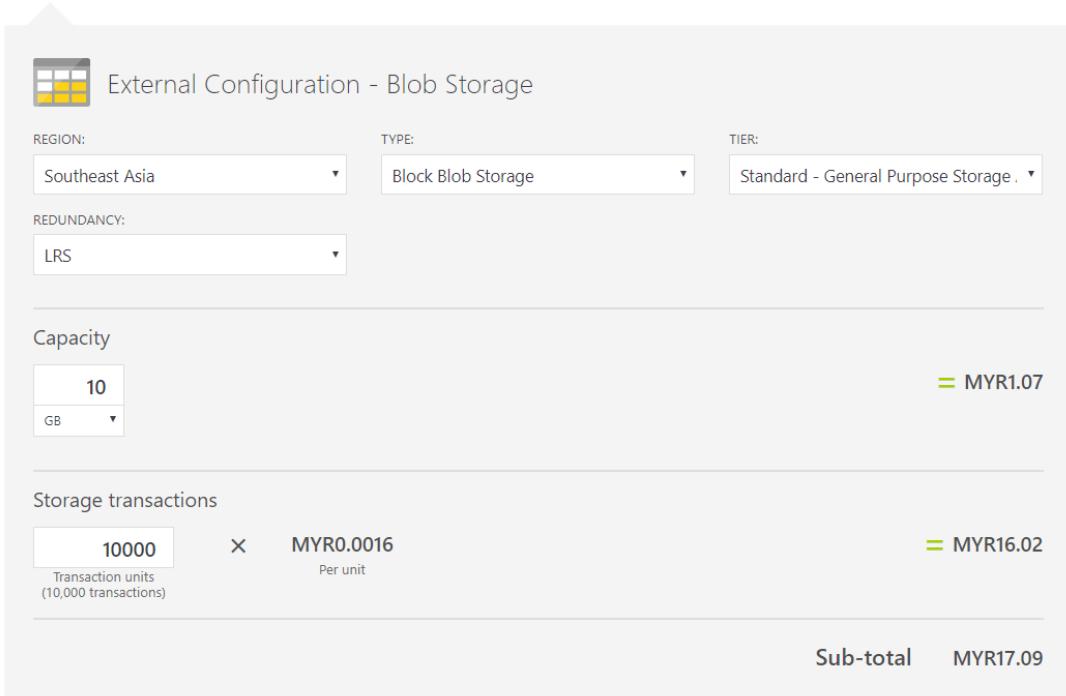


Figure 47. Traffic Manager - Cost Breakdown*Figure 48. Azure SQL Database - Cost Breakdown**Figure 49. Blob Storage - Cost Breakdown*

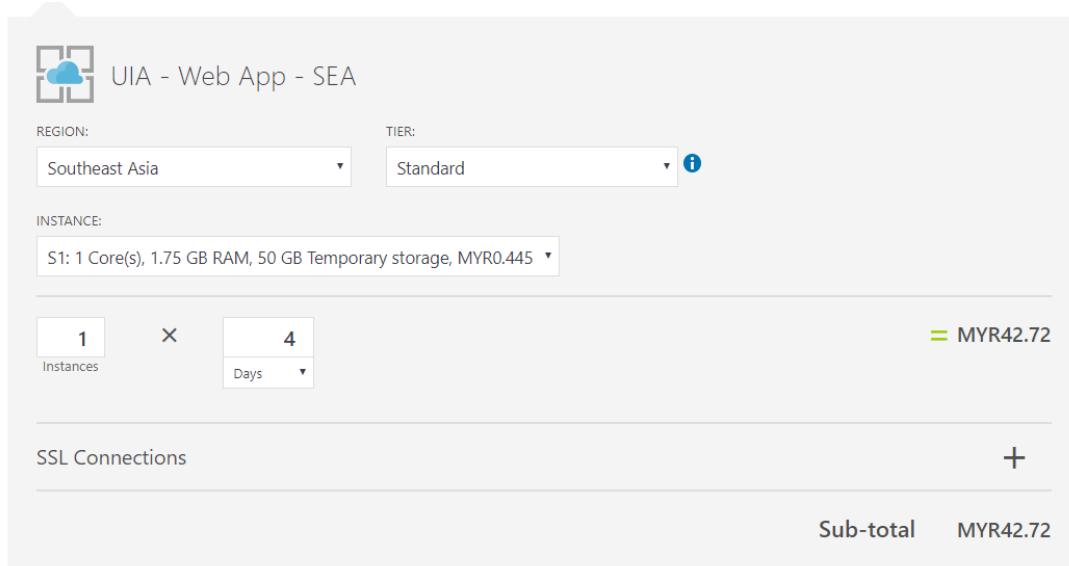


Figure 50. Web Application S1 - Cost Breakdown

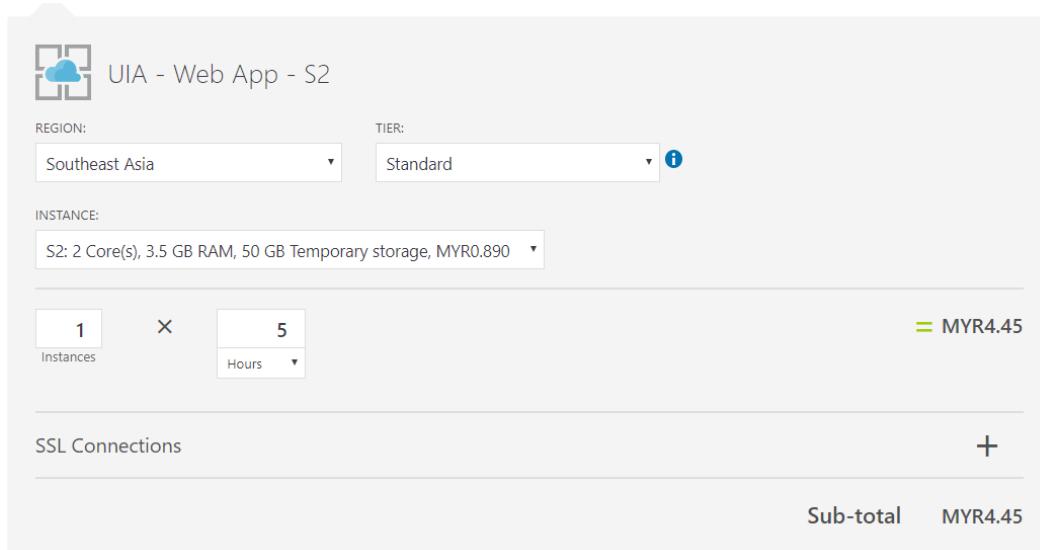


Figure 51. Web Application S2 - Cost Breakdown

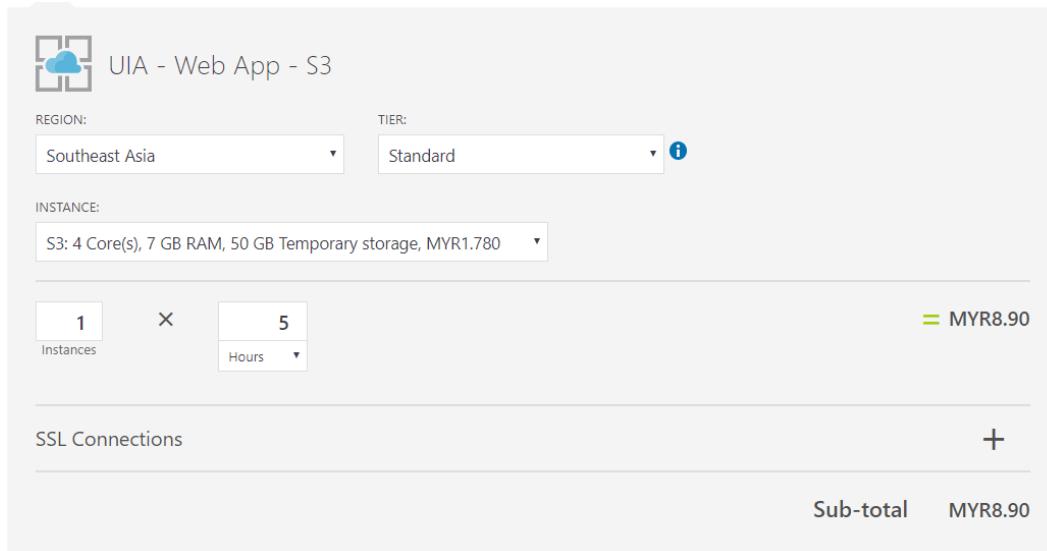


Figure 52. Web Application S3 - Cost Breakdown

The traffic manager's calculation was based off the two endpoints that will be deployed and the minimum 1 million query/month, which is sufficient to cover the needs. Additionally, the Azure SQL database was set at S0 (10 DTUs) for a duration of 6 days, which provides sufficient time to migrate the database to the cloud prior to the deployment of the web applications and throughout the duration of the deployment. Furthermore, the blob storage necessary to implement the external configuration store cloud design pattern was set for 10,000 transactions, as there is a constant remote dependency on the configuration container that will be hosted, while the storage is set at the bare minimum, as the web configuration is generally very small in size.

Lastly, the web applications, which account towards the majority of the cost have been analyzed in terms of pricing for several things. Firstly, the cost has been calculated for the S1 tier for a duration of 4 days, which provides sufficient time to perform all the necessary actions (such as performance testing, documentation, and video demonstration) for the web application. Secondly, the cost of S2 has been calculated for a duration of 5 hours, which is necessary, as the app will be scaled-up as part of the performance testing strategy, and lastly the same cost has been calculated for the S3 tier, which is also needed as part of the performance testing strategy. Therefore, the overall cost falls just below RM 140 and can be covered under the Visual Studio Dev Essentials package.

4.2.2 Web App

Microsoft Azure provides the ability to develop and deploy cloud-enabled web applications through its Platform as a Service (PaaS). For this project, the developed web

application will be published and deployed in two separate regions, namely Southeast Asia, and West Europe. Southeast Asia is chosen as the primary region for the web application as the development team is located in this region, and furthermore UIA is planning to extend their business network to the Asia Pacific region. Secondly, West Europe is chosen as the second region, as UIA's headquarter operates in Ukraine, and mostly in Europe. That is the closest endpoint to Ukraine as there are no data centers in Russia due to political and security stability influx. The standard tier is chosen as the service plan for both the web apps, as it is the lowest tier required to implement the traffic manager, which is another necessary resource required for achieving the deliverables. The prices within this tier double as you move across the levels (S1, S2, S3), and S1 is deemed suitable as it falls within the budget, while providing geo-availability through the utilization of a traffic manager.

S1 Standard	S2 Standard	S3 Standard
1 Core 1.75 GB RAM 50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	2 Core 3.5 GB RAM 50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	4 Core 7 GB RAM 50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability
331.08 MYR/MONTH (ESTIMATED)	662.16 MYR/MONTH (ESTIMATED)	1,324.32 MYR/MONTH (ESTIMATED)
B1 Basic	B2 Basic	B3 Basic
1 Core 1.75 GB RAM 10 GB Storage Custom domains SSL Support SNI SSL Included Up to 3 instance(s) Manual scale	2 Core 3.5 GB RAM 10 GB Storage Custom domains SSL Support SNI SSL Included Up to 3 instance(s) Manual scale	4 Core 7 GB RAM 10 GB Storage Custom domains SSL Support SNI SSL Included Up to 3 instance(s) Manual scale
248.31 MYR/MONTH (ESTIMATED)	496.62 MYR/MONTH (ESTIMATED)	993.24 MYR/MONTH (ESTIMATED)

Figure 53. Basic vs Standard Pricing Tier

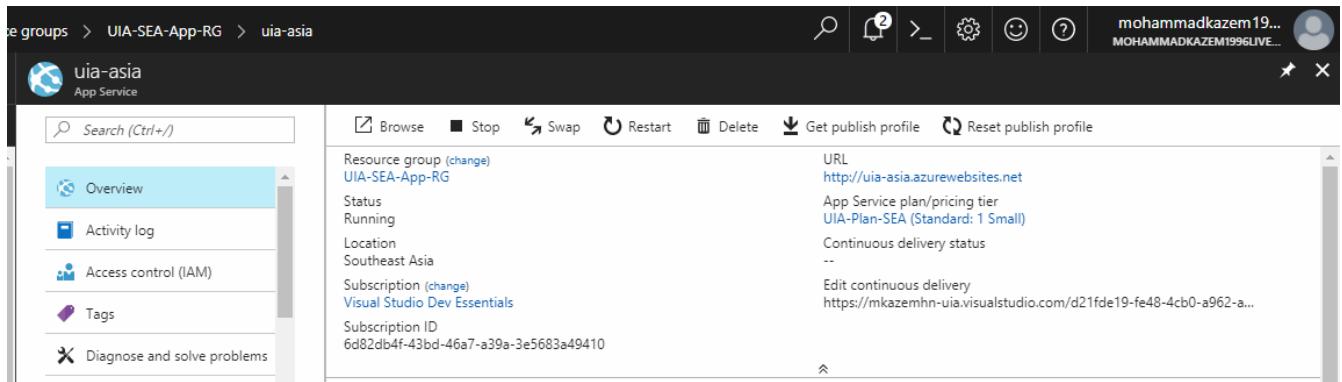


Figure 54. SEA-deployed Web App

The figure below shows the app service plan, URL, and location of the second deployed web app, which is located in West Europe.

The screenshot shows the Azure portal interface for managing an App Service named 'uia-europe'. The left sidebar lists navigation options: 'Search (Ctrl+)', 'Start', 'Swap', 'Restart', 'Delete', 'Get publish profile', and 'Reset publish profile'. A yellow warning box states: 'Your app is stopped. App Service plan charges still apply.' The main content area displays the following details:

- Resource group (change):** UIA-WestEU-App-RG
- Status:** Stopped
- Location:** West Europe
- Subscription (change):** Visual Studio Dev Essentials
- Subscription ID:** 6d82db4f-43bd-46a7-a39a-3e5683a49410
- URL:** http://uia-europe.azurewebsites.net
- App Service plan/pricing tier:** UIA-Plan-WestEU (Standard: 1 Small)

Figure 55. West Europe Web App

For the success of a web application, it is essential to bring about continuous changes that enhance the user experience. However, there are many bottlenecks associated with the manual deployment of a web application to Azure. As the number of endpoints increase, it becomes increasingly difficult to continuously push the latest changes to the live version. Microsoft Azure has introduced an option that is currently in the preview form, called continuous deployment, which allows for the continuous synchronization of the deployed web applications through pulling committed changes from a GitHub repository, as shown below.

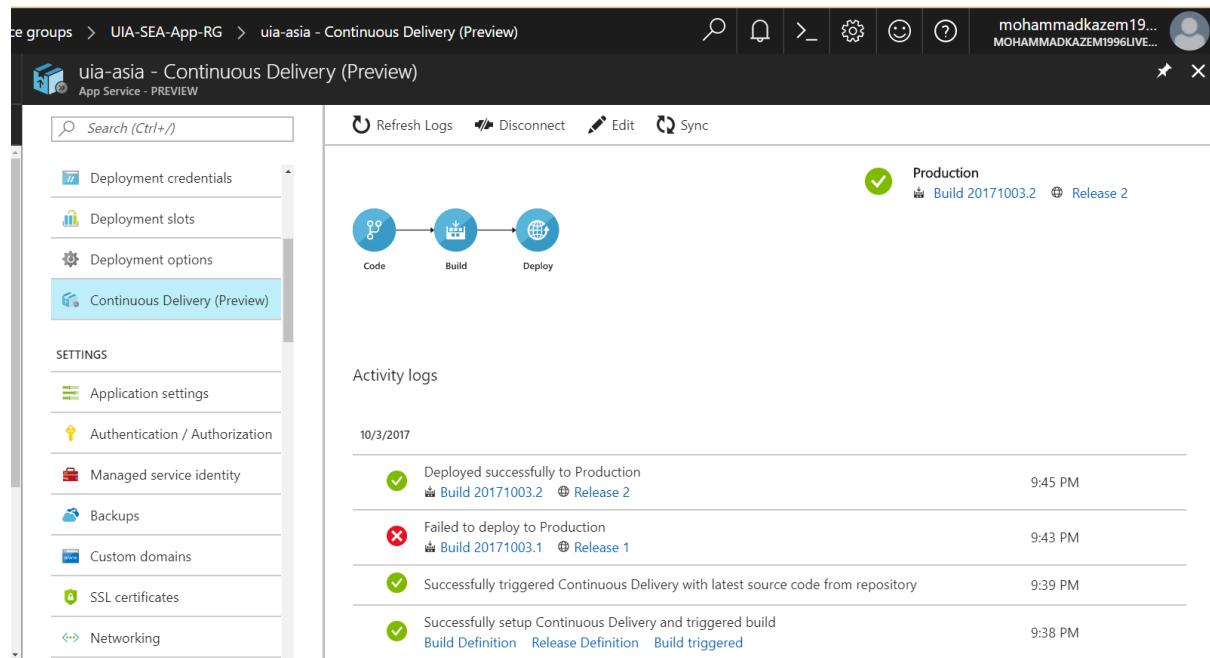


Figure 56. Continuous Deployment on Azure Web App

The screenshot shows a GitHub repository interface. At the top, it displays statistics: 2 commits, 1 branch, 0 releases, and 0 contributors. Below this is a navigation bar with buttons for 'Create new file', 'Upload files', 'Find file', and a prominent green 'Clone or download' button. The main area shows a list of commits under the 'master' branch:

Commit	Message	Time
Kazem Initial Commit		Latest commit 5a48920 2 days ago
UIA_Web	Initial Commit	2 days ago
.gitattributes	Add .gitignore and .gitattributes.	14 days ago
.gitignore	Add .gitignore and .gitattributes.	14 days ago
UIA-FlightWeb.sln	Initial Commit	2 days ago

Figure 57. GitHub Repository

One of the prioritized deliverables of this project is to be able to handle high demand during peak seasons that plays into the performance and scalability quality attributes of the system. In order to address this, web applications usually scale either horizontally (scale out/in) or vertically (scale up/down). For this project, the deployed web applications have been configured to scale-out by adding a second instance of the same service tier to assist in the

handling of increased load that can be detected through a set rule that is defined as an increase of over 90% to the CPU usage for the current instance, as shown below. However, it is also important to scale-in whenever the web application is idle and there is no longer a high demand that requires additional resources. Therefore, the instance count decreases to 1 when the average CPU usage is below 30%, as shown below.

Default Auto created scale condition			
Scale mode	<input checked="" type="radio"/> Scale based on a metric	<input type="radio"/> Scale to a specific instance count	
Scale out			
Rules	When	UIA-Plan-SEA (Average) CpuPercentage > 90	Increase instance count by 1
Scale in			
When	UIA-Plan-SEA (Average) CpuPercentage < 30	Decrease instance count to 1	
+ Add a rule			
Instance limits	Minimum	Maximum	Default
	1	2	1
Schedule	This scale condition is executed when none of the other scale condition(s) match		

Figure 58. Scale-out during high demand

4.2.3 Azure SQL Database

Flight booking web applications are hugely data-driven. This indicates the need for a database which is mainly reliable and secure. Furthermore, it is important to implement a solution that will provide seamless update and patches necessary to the SQL code base, and remove the unnecessary constraints set by managing the underlying infrastructure. Azure SQL Database can provide a high-level solution for this as a service through managed databases that are dynamically scalable with high-performance (Rabeler, et al., 2017).

By utilizing SQL database as a service, the database is isolated and a certain level of performance is guaranteed as governed by the chosen service tier. There are four service tiers provided for an Azure SQL database, namely basic, standard, premium, and premiumRS

(Rabeler, et al., 2017). The table below compares the varying service tiers provided for an Azure SQL database.

	Basic	Standard	Premium	Premium RS
Target workload	Development and production	Development and production	Development and production	Workload that can tolerate data loss up to 5-minutes due to service failures
Uptime SLA	99.99%	99.99%	99.99%	N/A while in preview
Backup retention	7 days	35 days	35 days	35 days
CPU	Low	Low, Medium, High	Medium, High	Medium
IO throughput	Low	Medium	Order of magnitude higher than Standard	Same as Premium
IO latency	Higher than Premium	Higher than Premium	Lower than Basic and Standard	Same as Premium

Figure 59. SQL Database Service Tier Comparison (Rabeler, et al., 2017)

The service tier chosen for this project's Azure SQL database is **standard**. As mentioned previously, web applications such as this are data-driven, therefore it is inherent that the I/O operations will be high with increased memory and CPU usage within the hosted server, which is defined as Database Transaction Units (DTU) by Azure. Basic provides the lowest DTU (low CPU, I/O throughput, and increased latency), which voids its appropriateness for this project, while premium and premium RS cannot be financially justified within the budget even though they provide the highest DTUs possible, therefore they are omitted, thus standard is the most optimal. The number of DTUs chosen for the standard tier is 10, which sets it as S0, with a storage size of 250GB.

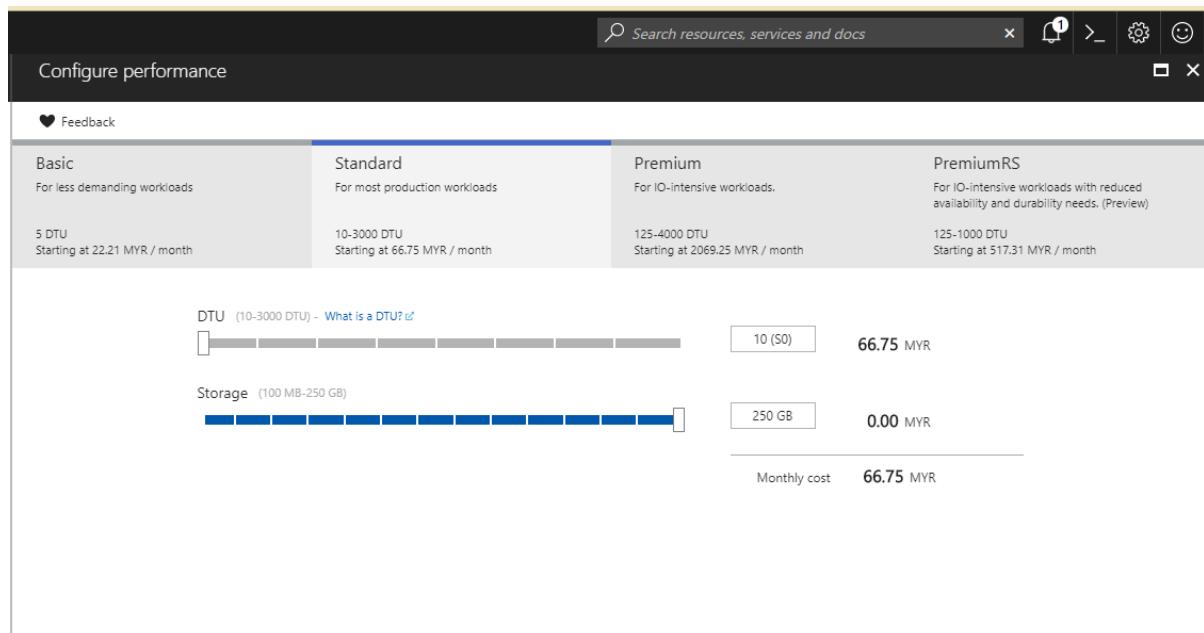


Figure 60. Chosen Database Service Tier

Furthermore, as the development office is located within Malaysia and UIA is planning to expand their business to the SEA region, the region chosen to deploy the database is Southeast Asia (SEA), as shown below. The interzone traffic is imminent as there are no regions between the two deployed regions (SEA and West Europe), without introducing two databases, which will significantly increase the operational cost, and increase the hassle for data synchronization and database management.

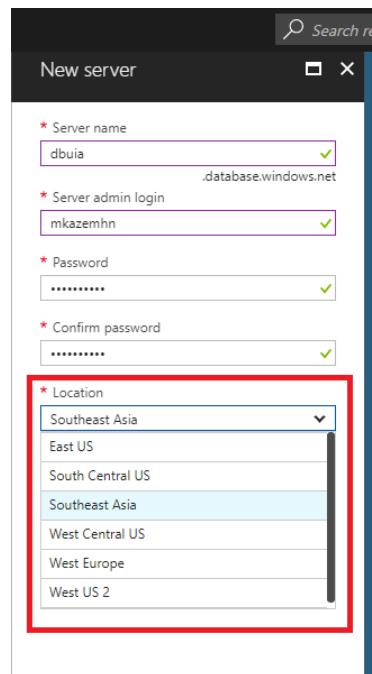


Figure 61. Database Server Location

After selecting the location, the database is isolated and managed on the created server as shown in the figure below.

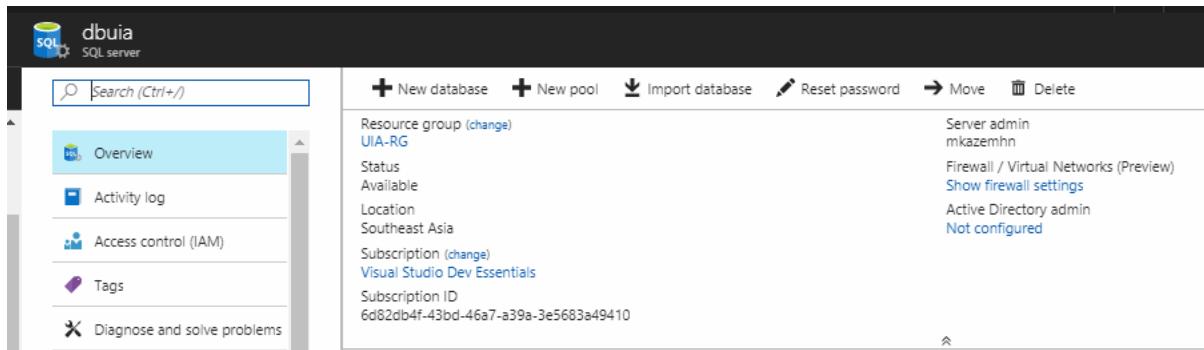


Figure 62. SQL Server

Lastly, as the developed web application has to handle confidential customer information and the privacy is of utmost importance, the database has been configured for three out of the four security measurements that Microsoft Azure provides within its database as a service, which are transparent data encryption, auditing, and threat detection, as shown below. Threat detection attempts to detect anomalous activity (such as SQL injections) that occur, while auditing constantly monitors and logs the database events, and transparent data encryption attempts to seamlessly encrypt the hosted database and its backups. These can be monitored through the security center within the Azure platform.

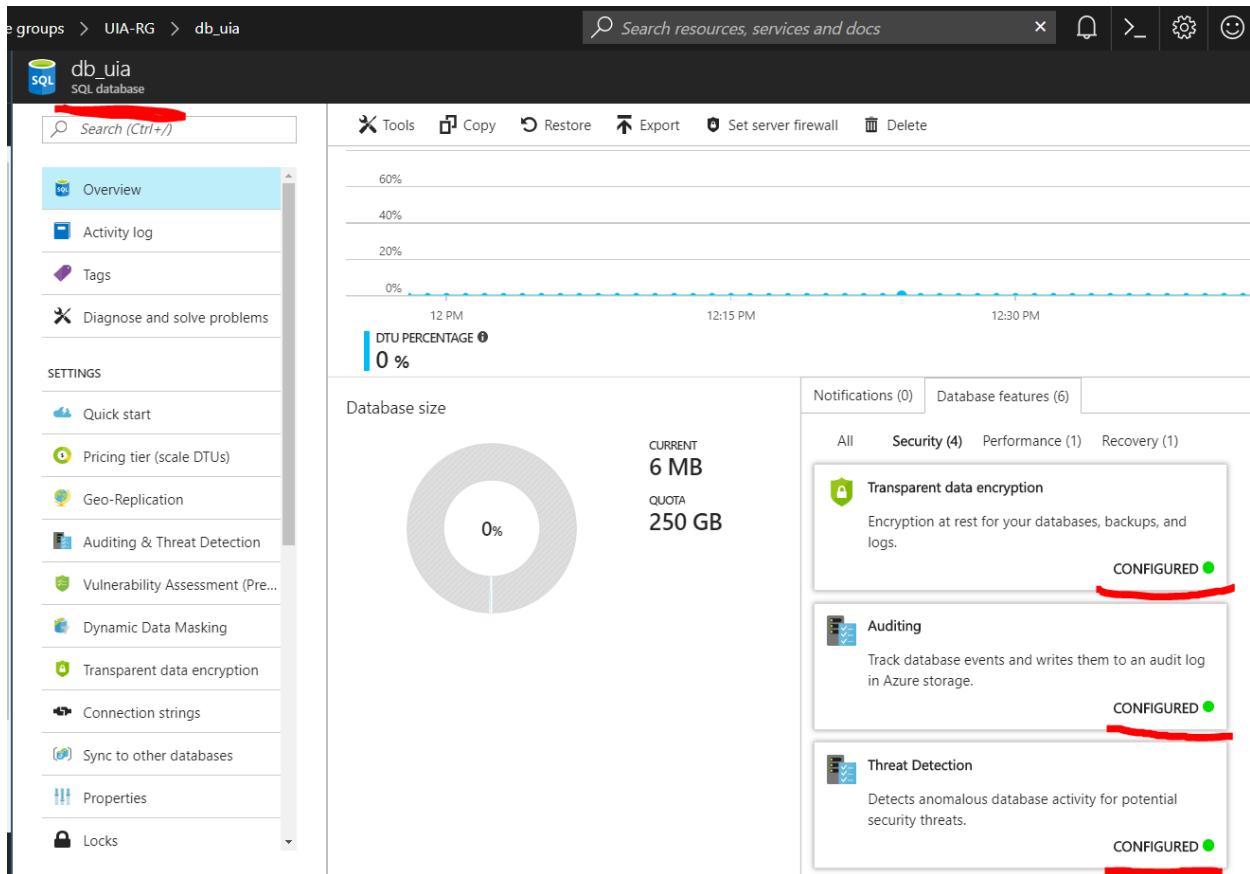


Figure 63. Database Security Settings

4.2.4 Traffic Manager

It is important to distribute user traffic for various endpoint services to improve responsiveness, availability, and implement an automatic failover when one of the endpoint fails (Dwivedi, et al., 2017). For instance, this will allow for zero downtime during peak seasons if there is a failure in one of the endpoints.

For virtual machines, this is often achieved through load-balancers, however for web apps, this can be directly achieved through an Azure Traffic Manager. This provides the ability to route traffic to the appropriate endpoint depending on the chosen routing method, and continuously monitor the endpoint health to automatically fall-back to another endpoint, incase an endpoint is down (maintenance or fault) without introducing any downtime. Furthermore, Azure Traffic Manager provides **four** routing methods, namely priority, weighted, performance, and geographic (Dwivedi, et al., 2017). The most suitable routing method for this application is **performance**. This selects the endpoint that provides the lowest network latency for a user when the endpoints are distributed geographically, thus allowing the user to use the

“closest” endpoint. It is preferable over both priority and weighted, as these can reduce the performance, in terms of latency, for the user, which is paramount as defined by the business needs. Furthermore, it is preferential over geographic routing method, as it is not necessarily true that the closest geographical endpoint (based on originating DNS query) would provide the lowest latency, which the geographic routing method would choose.

However, with performance routing-method, as UIA expands and more web apps are deployed within the same regions to improve performance, the traffic is distributed evenly across the available endpoints within that region. Furthermore, the performance routing method will fall-back to the next endpoint with the second lowest latency for the user if the first endpoint is degraded, an example is shown below.

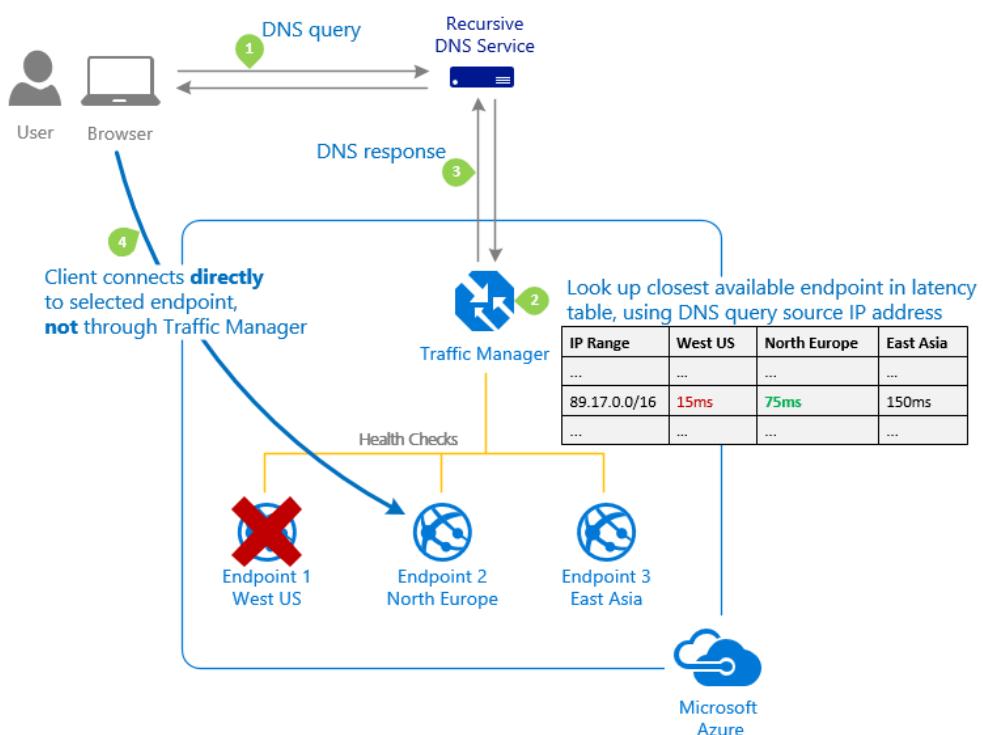


Figure 64. Traffic Manager Performance Routing Method (Dwivedi, et al., 2017)

The figure below depicts the two app services that are deployed in the two geographically different regions (SEA and West Europe), which have been chosen as the endpoints, as well as the DNS name that the user will query, through which the user will directly to the appropriate endpoint.

The screenshot shows the Azure Traffic Manager profile 'uia'. On the left, there's a navigation sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration, Real user measurements, and Traffic view. The 'Overview' tab is selected. The main area displays the 'Essentials' section with details such as Resource group (UIA-TM-RG), Status (Enabled), Subscription name (Visual Studio Dev Essentials), Subscription ID (6d82db4f-43bd-46a7-a39a-3e5683a49410), DNS name (uia.trafficmanager.net), Monitor status (Online), Routing method (Performance), and Location (Southeast Asia). Below this is a table titled 'Search endpoints' showing two entries: UIA-SEA and UIA-EU, both of which are Enabled, Online, and of type Azure endpoint, located in Southeast Asia and West Europe respectively.

Figure 65. Traffic Manager Endpoints

To continually monitor the health of the endpoints, the path has been set to the /favicon.ico, which is a static path across almost all web applications, as shown below.

This screenshot shows the configuration settings for the 'uia' Traffic Manager profile. The left sidebar is identical to Figure 65. The right pane is titled 'Save' and 'Discard'. It contains sections for 'Routing method' (set to Performance), 'DNS time to live (TTL)' (set to 300), 'Endpoint monitor settings' (Protocol: HTTP), 'Port' (set to 80), and 'Path' (set to /favicon.ico).

Figure 66. Traffic Manager Configuration

4.2.5 Application Insights

During the development of the web application, application insights was registered for the web application to collect, analyze, and investigate various aspects of the web application. Firstly, it provides perceptive information regarding varying levels of failed requests that can be further investigated with analyzing the corresponding GET/POST methods and the associated controller with the system/server exceptions that occurred, as shown below. The application insights were collected using the free tier that allows for up to 1 GB of data to be collected (through telemetry) as to eliminate any additional costs.

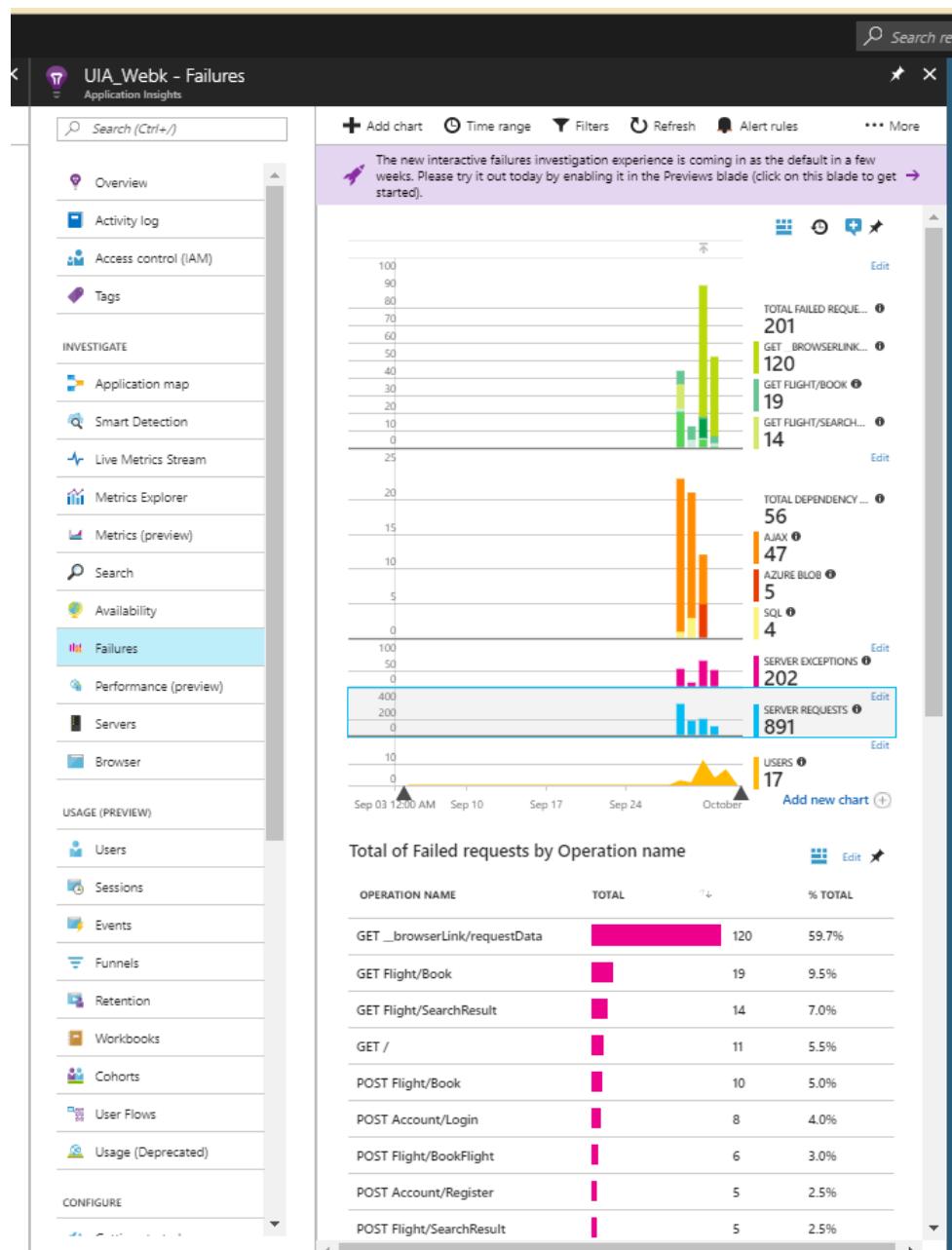


Figure 67. Application Insights - Failures

These allowed the developer to further investigate failures and commit changes that were pushed into the deployed web application, and also analyze the remote dependency calls, such as the pulling the latest configuration from the blob storage.

The screenshot shows the Microsoft Application Insights interface for a failed request. The top navigation bar includes 'Application Insights' > 'UIA_Web - Failures' > 'System.ArgumentException at UIA_Web.Controllers...'. The main title is 'Value cannot be null or empty.
Parameter name: url'. Below it, it says 'System.ArgumentException' and 'POST Account/Login'. The event time is '9/29/2017, 2:53:13 AM' and it is '1 of 6'. There is a link to 'Collect debug snapshots...'. The central panel displays the error message 'Value cannot be null or empty. Parameter name: url'. It details the failed method as 'UIA_Web.Controllers.AccountController+<Login>d__11.MoveNext'. It also shows user information: User Id (d/ye+), Session Id (e5EXM), Client IP address (127.0.0.1), Device type (PC), and Cloud role instance (DESKTOP-BB7TVGN). There is a link to 'See all properties'. Below the error message, there are three options: 'Search for this error online', 'Show telemetry for: this operation this session 5 minutes before and after this event', and 'Show timeline for: this session this user'. A note below says 'Related items: Traces for this exception Example request affected by this exception'. At the bottom, there is a 'Call Stack' section with a 'METHOD' column, a 'FILE' column, and a 'LINE' column. The stack trace shows:

METHOD	FILE	LINE
System.ArgumentException		
System.Web.Mvc.Controller.Redirect		
UIA_Web.Controllers.AccountController+<Login>d__11.MoveNext	AccountController.cs	84
[external code]		
System.Web.HttpApplication+CallHandlerExecutionStep.OnAsy...		

Ellipses (...) are shown at the bottom of the call stack.

Figure 68. Example of exception during failure

Additionally, it allowed the developer to gain some metrical insight into the performance aspect of the application to further justify the need for scaling. However, based on the units of measurement, the application performed very well and the service plan that was chosen for it sufficed with the current load.

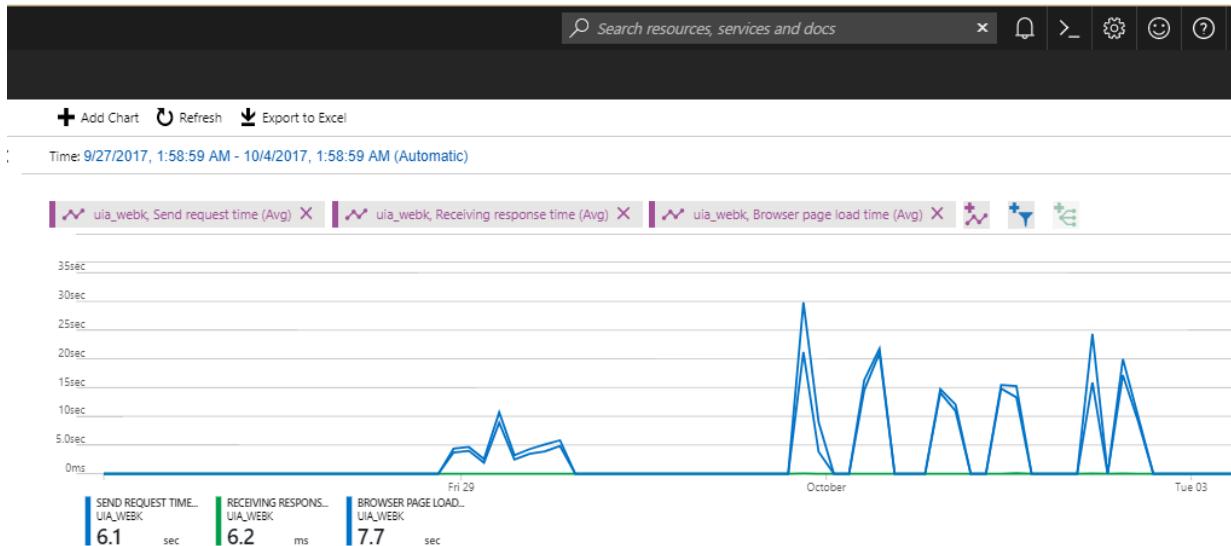


Figure 69. Performance Metrics - Application Insights

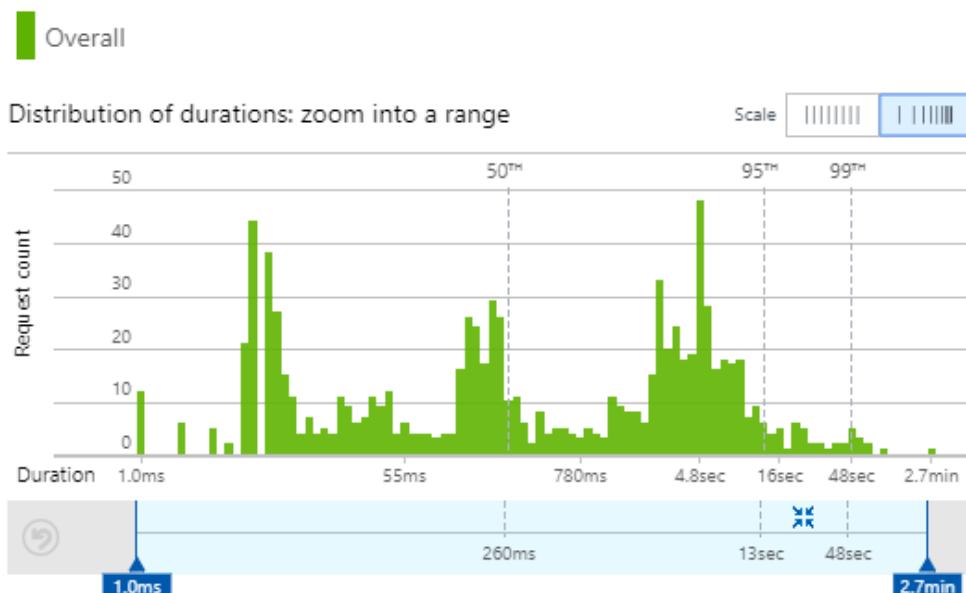


Figure 70. Response Time Visual Map

OPERATION NAME	DURATION (AVG)	COUNT
Overall	3.78 sec	891
POST Account/Register	15.9 sec	11
POST Flight/Book	15.1 sec	28
POST Flight/BookFlight	10.2 sec	7
POST Flight/SearchResult	7.57 sec	102
POST Account/Login	5.87 sec	52
GET Manage/Index	4.15 sec	22
GET Flight/Book	3.77 sec	115
GET Home/Index	3.39 sec	243

Figure 71. Response Time Analytics - Application Insights

4.2.6 Azure Blob Storage

The implemented cloud design pattern, external configuration store, calls for a centralized location through which the deployed web applications can upload and read the latest configuration file from. Microsoft Azure provides blob storage as a service that stores unstructured data or binary objects in the cloud (Myers, et al., 2017). The number of transaction units chosen for this blob storage was 10,000 as the web applications are highly remotely dependent on the configuration container, and query for the configuration information constantly. The configuration container that has been set-up as a blob storage for the external configuration store is shown below.

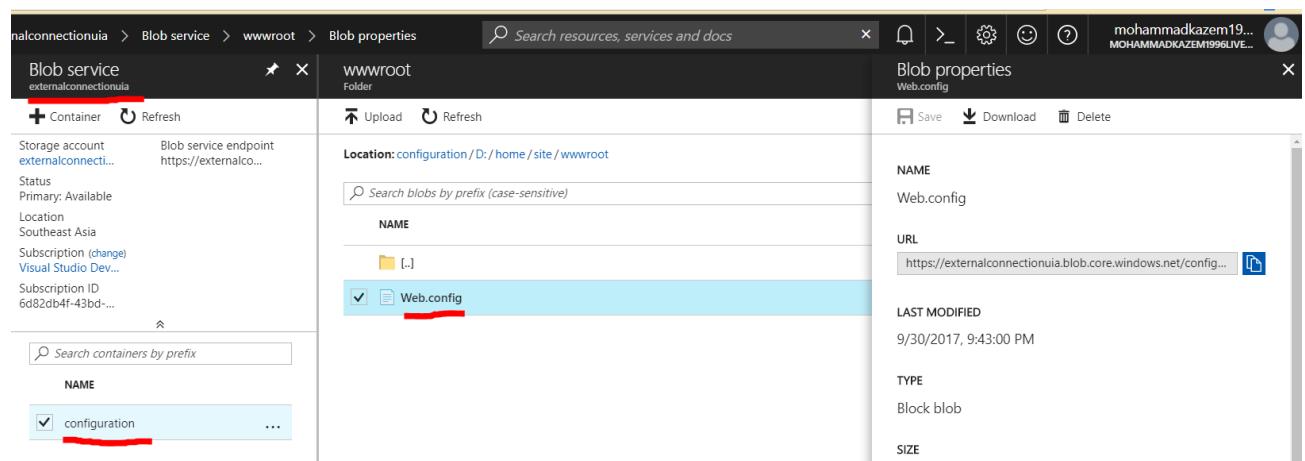


Figure 72. Configuration Container – Blob Storage

Lastly, another blob storage was used to store the auditing logs through the configured security measurement, as shown below.

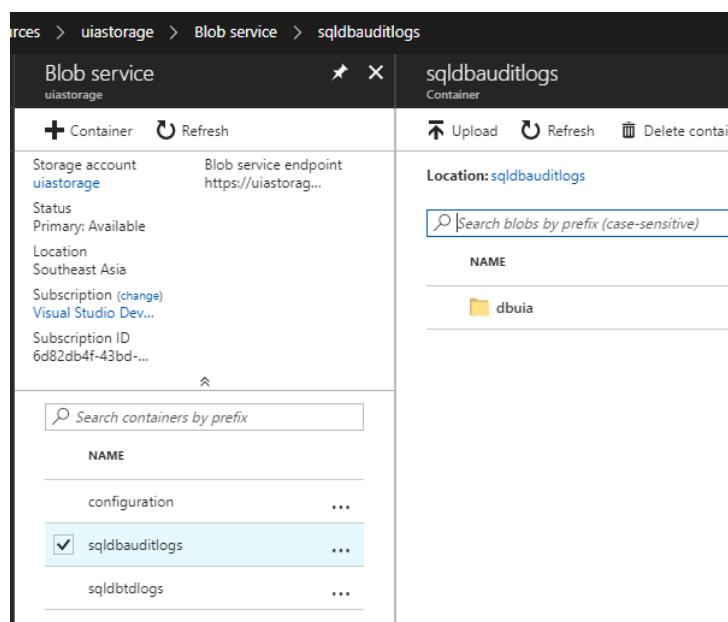


Figure 73. Auditing Log Container - Blob Storage

5.0 Testing

5.1 Test Plan

5.1.1 Functional Testing

Functional testing is a core testing approach that assures the conformance of the developed functions/modules per the requirement specification. This will allow the different features to be tested in terms of their functionality under different scenarios, and the actual result is compared to the expected result. The functional test cases will run after the development phase. For this project, the scope of modules that are to be tested and those that will be excluded need to be defined. The features that will be tested under the functional test are:

- **Login**
- **Register**
- **Search Flight**
- **Geo-IP Based Currency**
- **Book Flight**

The modules that are out-of-scope for the functional test are:

- **Modify User Details** – all the various functionalities needed are being covered with the other features, therefore it would be redundant.
- **Change Password** – it is using a well-established module (as part of the ASP.NET Identity System provided by Microsoft), therefore it has been well-tested prior to this project.

Table 1. Functional Test Plan

ID	Feature/Function Tested	Test Procedure/Scenario	Description	Expected Result	Priority
FT1	Login	User presses on the Sign in button in the navigation bar. User fills out his username and password correctly. User presses on the login button.	This test involves testing the login module when a user has already registered and attempts to login with their correct credentials and the sign-in manager that is used to authenticate the user.	User is authenticated and cookie created. User is redirected to the home page. The user's email is viewed as login in the top right corner of every webpage.	High
FT2	Search Flight	User has logged in. User chooses the trip type, seat class, origin, destination, departure date, and return date. User presses search flight.	This test involves checking one of the primary requirements of the system, which is to allow for users to search for flights based on their needs.	Database is queried for available flights for the specified criteria. User is redirected to the result page that displays the respective flights and their prices	High
FT3	Book Flight	User has searched for his flight of choice. User presses on the book button on the flight search result page. User selects his/her desired available seat. User fills out the passenger details. User reviews the flight details. User fills out payment details	This test encapsulates one of the primary modules that is required as part of the system requirements, which is to successfully book a flight ticket based on the search result.	Records are added to the database. User is re-directed to the confirmed booking webpage.	High

ID	Feature/Function Tested	Test Procedure/Scenario	Description	Expected Result	Priority
		User presses on the confirm button.			
FT4	Register User	<p>User wants to search and book a flight, but doesn't have a customer profile.</p> <p>User presses on the register button in the navigation bar.</p> <p>User fills out the needed details.</p> <p>User presses on the register button.</p>	This test focuses on one of the primary requirements of the project, which is the creation of customer profiles, by testing its functionality and conformance to the requirement itself.	<p>User validation is done.</p> <p>Identity created for user.</p> <p>User properties are inserted into the database.</p> <p>User is authenticated and cookie created.</p> <p>User is redirected to the home page.</p> <p>The user's email is viewed as login in the top right corner of every webpage.</p>	High
FT5	Register User	<p>User wants to search and book a flight, but doesn't have a customer profile.</p> <p>User presses on the register button in the navigation bar.</p> <p>User fills out some of the needed details.</p> <p>User presses on the register button.</p>	This test involves checking how the validation of one of the primary requirements of the system, which is the creation of customer profile.	<p>The system validates the model.</p> <p>If, invalid then the system returns the set of fields that were invalid back in the view.</p>	Low
FT6	Geo-IP based Currency	User views the homepage from Malaysia.	This test involves the conformance to one of the main requirements, which is providing customized	System extracts the user's IP address from the request's "REMOTE_ADDR" variable.	High

ID	Feature/Function Tested	Test Procedure/Scenario	Description	Expected Result	Priority
		User scrolls to the Available Flights section and views the prices.	information to the customer based on their location.	System sends a request to freegeoip.net using their free API. System parses the JSON response and extracts the country. For country_name “Malaysia”, “RM” is displayed as the currency in the homepage.	
FT7	Geo-IP based Currency	User views the homepage from Finland. User scrolls to the Available Flights section and views the prices.	This test involves the conformance to one of the main requirements, which is providing customized information to the customer based on their location.	System extracts the user’s IP address from the request’s “REMOTE_ADDR” variable. System sends a request to freegeoip.net using their free API. System parses the JSON response and extracts the country. For country_name “Finland”, “€” is displayed as the currency in the homepage.	High
FT8	Search Flight	User has logged in. User chooses the trip type, seat class, origin, departure date, and return date, but leaves the destination field empty	This test involves checking how the validation of one of the primary requirements of the system is done, which is searching for a flight.	The system validates the SearchFlightView model. If, invalid then the system returns the set of fields that were invalid back in the view.	Low

ID	Feature/Function Tested	Test Procedure/Scenario	Description	Expected Result	Priority
		User presses search flight.			
FT9	Book Flight	<p>User has searched for his flight of choice.</p> <p>User presses on the book button on the flight search result page.</p> <p>User selects his/her desired available seat.</p> <p>User fills out the passenger details.</p> <p>User reviews the flight details.</p> <p>User fills out payment details</p> <p>User presses on the confirm button.</p>	<p>This test involves checking how the validation of one of the primary requirements of the system is done, which is book a flight after the search result.</p>	<p>The system validates the SearchFlightView model. If, invalid then the system returns the set of fields that were invalid back in the view</p>	Low

5.1.2 Performance Testing

Performance testing is another vital software test that assesses the performance software quality metrics in terms of several units of measurement. Firstly, it can be measured in terms of throughput, which can be defined in terms of total number of requests per second. Secondly, it can be measured through the average response time or average latency that provides an insight as to how the current web application can handle varying loads and its impact to the performance, which will allow the developer to identify if the web application requires to be scaled or as-to the breaking point (stress testing) that will allow the developer to identify the conditions under which the web application needs to be scaled. For instance, when the CPU percentage is over 90% or if the average response time is above 20s.

Microsoft Azure allows for the execution of performance tests for a web app that has been deployed. It is done through the Visual Studio Team Services (VSTS) that provides 20,000 virtual user minutes (VUM) for free. The VUM is a product of the concurrent user load and the duration of the performance test, as shown in the table below. It is important to note that throughout each performance test, the instance count will remain 1.

Table 2. Performance Test Plan

Test No.	User Load	Duration (minutes)	Service Tier	Virtual User Minutes (VUM)
1	100	2	S1	200
2	300	2	S1	600
3	500	2	S1	1000
4	1000	2	S1	2000
5	100	2	S2	200
6	300	2	S2	600
7	500	2	S2	1000
8	1000	2	S2	2000
9	100	2	S3	200
10	300	2	S3	600
11	500	2	S3	1000
12	1000	2	S3	2000
				Total: 11,400 VUMs

5.2 Results & Analysis

5.2.1 Functional Tests

The functional tests were performed manually without the usage of automated tools such as IntelliTest or Visual Studio's in-built Unit Test. The results are based on the test cases that were defined in the prior section. The table below shows the set of results achieved through the execution of the test cases defined.

Table 3. Functional Test Results

ID	Actual Result	Status
FT1	As expected	Pass
FT2	As expected	Pass
FT3	As expected	Pass
FT4	As expected	Pass
FT5	New record inserted into the database with several empty fields	<u>Fail</u>
FT6	As expected	Pass
FT7	As expected	Pass
FT8	New record inserted into the database with several empty fields	<u>Fail</u>
FT9	New record inserted into the database with several empty fields	<u>Fail</u>

The results of the functional test dictate that several of the test cases had actually failed. However, these are mainly low-priority that does not directly affect the achievement of the requirements, and can be implemented within the next commits for the deployed web application. These were mostly validation issues, and as the web application serves as a proof of concept, as set by the well-defined scope of the project, validation was not necessary as the deliverable is not to deliver a fully commercial web application. Overall, majority of the test cases had passed, most notably, all of the test cases that were directly related to the requirements of the project.

5.2.2 Performance Tests

The performance tests were carried out as planned in the prior section. Microsoft Azure provides numerous data-points that were captured across each performance test, which were the throughput, average latency, user load, and other metrics. The figures below illustrate the various captured information through Microsoft Azure through the respective 100, 300, 500, and 1000 concurrent user load increments.

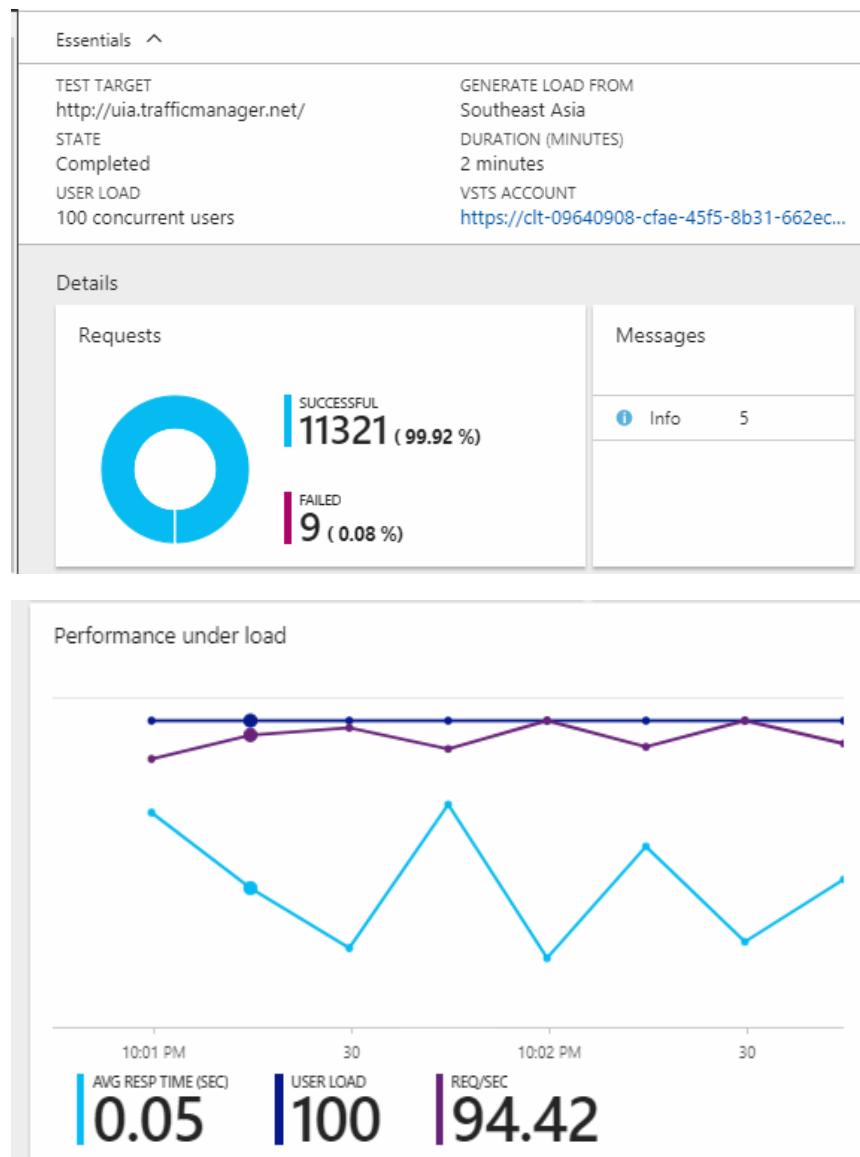


Figure 74. Performance Metrics - S1 - 100 Concurrent Users

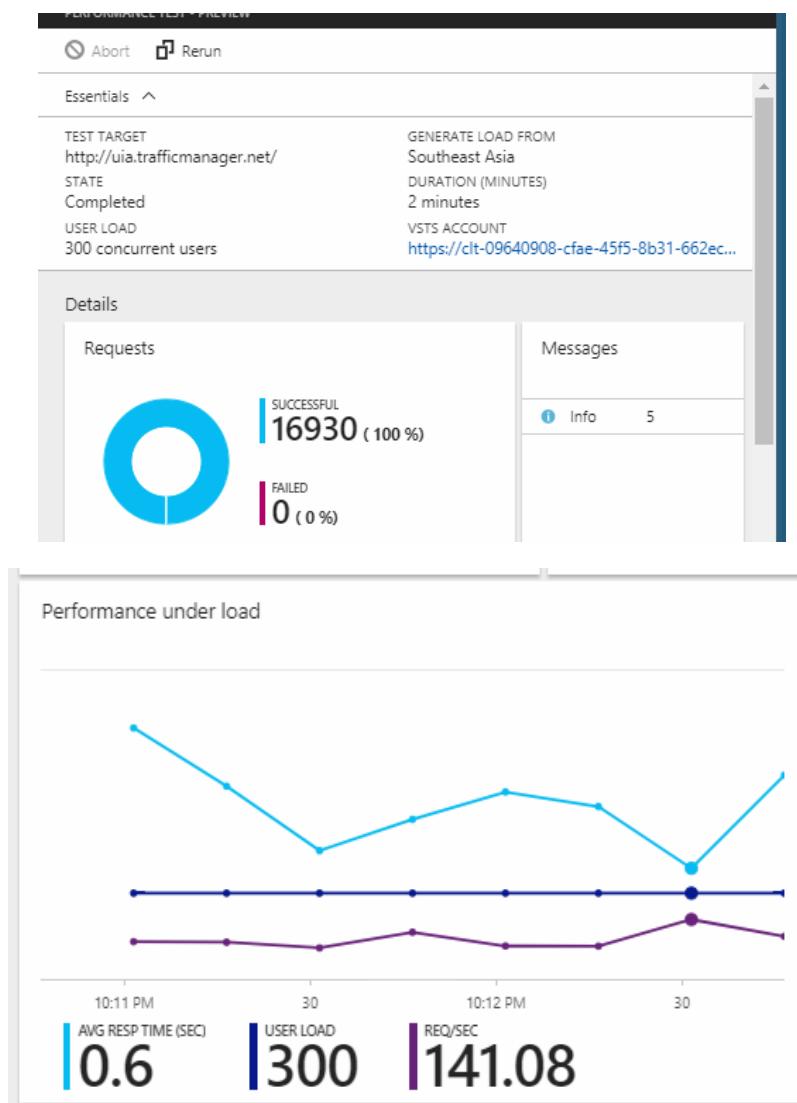


Figure 75. Performance Metrics - S1 - 300 Concurrent Users

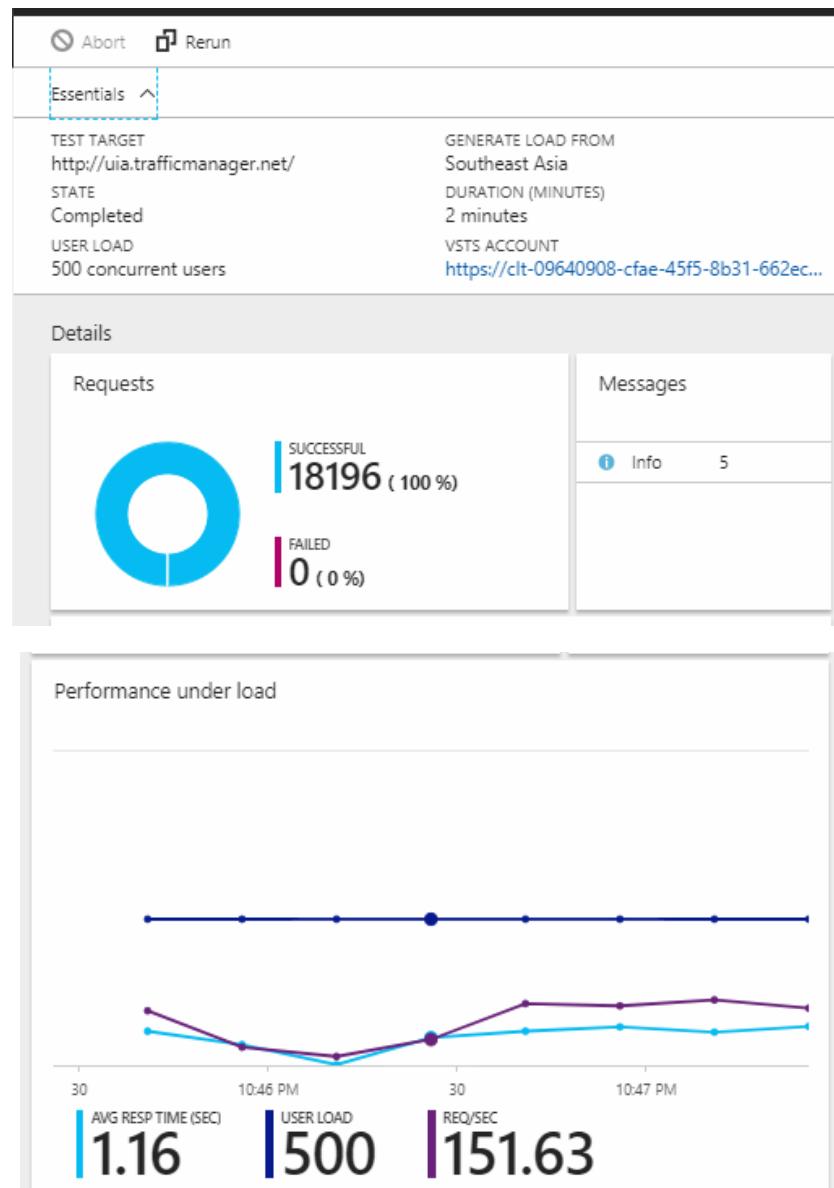


Figure 76. Performance Metrics - S1 - 500 Concurrent Users

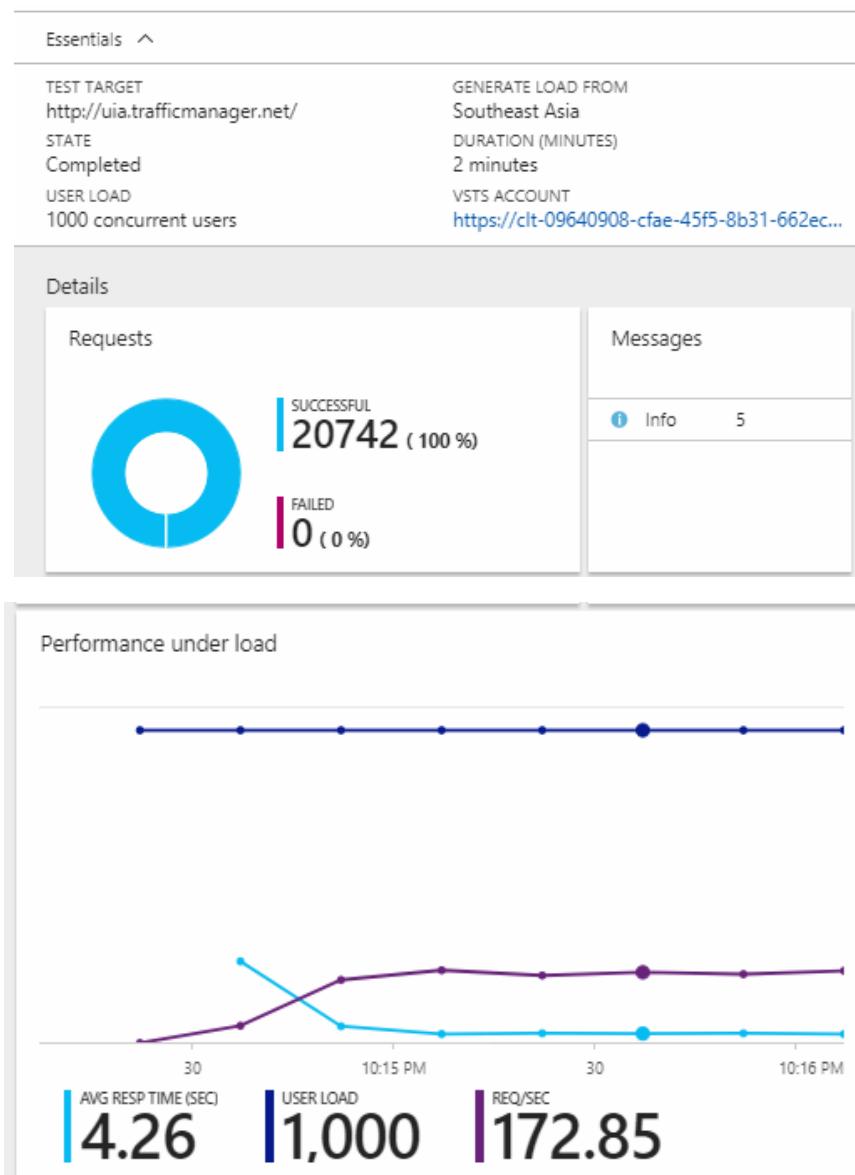


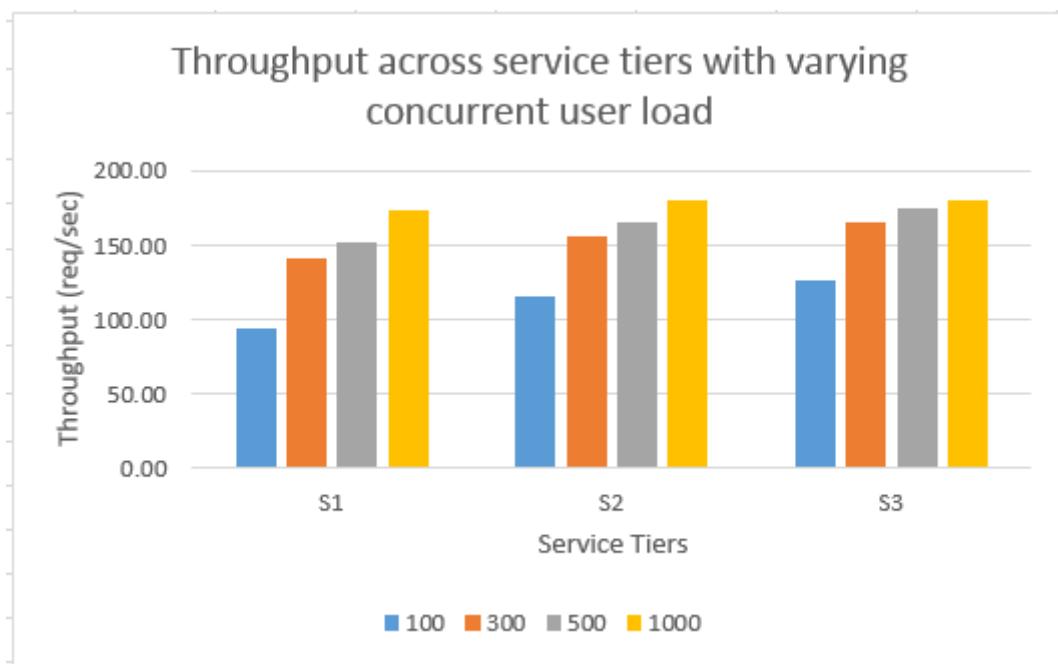
Figure 77. Performance Metrics - S1 - 1000 Concurrent Users

The web application was then scaled-up to an S2 service plan and the same measurements were taken, after which it was scaled-up to an S3 service plan and the same measurements were taken one last time. The data points captured across all the 12 performance tests were then comparatively evaluated, as shown in the tables below.

Table 4. Throughput every second across varying user load and service tiers

User Load \ Service Tier	S1	S2	S3
100	94.42 req/sec	115.25 req/sec	125.54 req/sec
300	141.08 req/sec	155.32 req/sec	165.44 req/sec
500	151.63 req/sec	165.54 req/sec	174.65 req/sec
1000	172.85 req/sec	179.54 req/sec	180.29 req/sec

The throughput, which is measured as the number of requests per second, naturally increases as the userload increases. The results can then be visualized to provide a more meaningful representation of the collected data points. From the chart below, there are numerous takeaways that mark the importance of the throughput as part of the performance test. It can be seen that there a general increase in throughput across an increase in the number of user load. However, there is no major difference between the throughput across high load of 500 and 1,000 concurrent users, while the web application running the S1 tier performed relatively poorly with 100 and 300 concurrent users when compared to S2 and S3. However, there was no major difference seen between S2 and S3 within these two user loads that would justify the **doubled cost** between S2 and S3.

*Figure 78. Throughput across service tiers with varying concurrent user load chart*

The second set of metrics that were calculated through the collected data points is the average latency with variance in the concurrent user loads. This has been captured in the table below.

Table 5. Average Response Time across varying user load and service tiers

User Load \ Service Tier	S1	S2	S3
100	0.05 sec	0.05 sec	0.04 sec
300	0.6 sec	0.32 sec	0.5 sec
500	1.16 sec	0.95 sec	0.86 sec
1000	4.26 sec	3.43 sec	3.2 sec

There is a general increase in the latency, as expected, across all the service tiers when the number of concurrent users increases. However, there is a notable increase from 500 to 1,000 concurrent users, especially with S1. However, S2 and S3 have performed almost equally, however S3 costs double as much as S2.

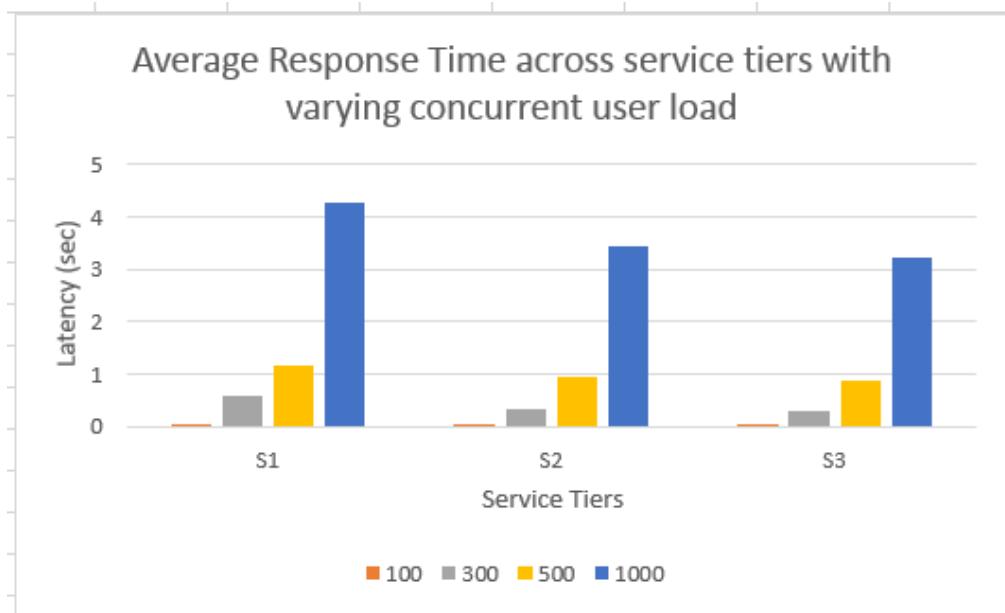


Figure 79. Latency across service tiers with varying concurrent user load chart

Therefore, based on the results that were collected and visualized, it can be seen that there is a notable difference between S1 and S2 with the variance in the concurrent user load across both the throughput and the average response time (latency), which justifies the price difference. However, there was slight improvements in throughput and reduction in average

response time across all the concurrent varying loads between S2 and S3, which indicates that the performance provided by the S3 is not well-justified for the doubled cost against S2, even though the resources were doubled. Therefore, the most justifiable scale-up option is to scale from S1 to S2 to meet the high demands during peak seasons, as S1 performs well enough during normal seasons, and S2 has shown drastic improvements with over 500 concurrent user load.

6.0 Conclusion

Ukraine International Airlines has come a long way since its establishment. Over the period of 15 years, they have placed a strong footing within the competitive airline market with a dominance of over 20% of the market segment. However, due to the recent political and economic instability, and de-valuation of the Ukrainian currency, UIA was striven to remain competitive through strategical overhauls to their staff, routes, and carriers resulting in reduction in ticket fares to enable them to further internationalize their services and contend as one of the world's low-fare carriers. UIA has long used technology to help reduce costs and improve customer service. The Chief Information Officer of UIA has realized that through migration of their website out of their own datacenters into the public cloud, they can further gain a competitive advantage over their competitors and improve customer services with the extensive set of tools and services that are provided.

The aim of this project was to develop and deploy a single-tenant flight booking web application with the minimal requirements of customer profile creation, management of the entire booking process, and customized information based on user location. The development that was based in Malaysia had a budget of RM 150 to utilize for the efficient design of a cloud architecture with the necessary Azure services to achieve the set deliverables.

The web application, which was developed using the MVC architecture complemented with Entity Framework, utilized ASP.NET Identity System for secure authentication of the user, GeoIP-based currency to provide varying currencies based on the user's geographical location, and dynamic seat picker to further enhance the user experience.

The cost of the various Azure services were broken down to analyze and dictate the service plans that were required and the associated cost with each necessary Azure service. The web application was deployed in two regions, Southeast Asia, as the software development team is located primarily in Malaysia, and UIA is planning to expand their carrier network, and secondly, West Europe, as UIA's headquarter functions in Ukraine, and this is the closest region to Ukraine. The web applications were configured for continuous delivery and to scale-out to meet the demands during peak seasons. The Azure SQL Database was configured for numerous security measurements to further enhance the database security and monitoring due to the criticality of the privacy and confidentiality of the information. Traffic manager was also implemented to allow for the effective distribution of user load based on the performance routing method and continuous endpoint health monitoring. Furthermore, application insights

was added to the SEA web application to gather, analyze, and investigate numerous metrics, such as performance as well as failed requests (with raised exceptions) during the development and deployment phase. However, the free version was used, which provided up to 1GB of free data storage through telemetry. Lastly, two blob storage containers were set-up, first of which allowed for the implementation of the external configuration store cloud design pattern, and the second of which was used for the auditing logs as part of the active security configuration of the Azure SQL database. After the implementation phase, the developed modules were tested in terms of functionality to ensure the compliance of the modules to the defined requirements. Additionally, performance testing was done to analyze how the application can handle user load at varying concurrent user load sizes with different service tiers to find the optimal scale-up solution, through which the developer concluded that S2 provides the best performance for its given price, while S1 will suffice for non-peak seasons, and S3 was well-beyond justifiable given its enormous cost (triple the price of S1) and the minor performance enhancement that it provides.

There are several future enhancements that can be proposed to enhance and further advance the web application. Firstly, by utilizing the cache-aside pattern, the implemented cloud design pattern, external configuration store, can be complemented, as it will require lesser remote dependency on the centralized location, as it will only update once there is a change, while the configuration is held inside the cache. Secondly, if there were no budget constraints, then the premium service tiers would provide in magnitudes higher performance as evident through the comparative table shown previously. Lastly, the web application can be deployed in more than two regions as UIA is planning to further expand their aircraft network, which will allow users to experience better performance when using the web application with a nested traffic manager that manages the various endpoints in all supported regions.

All in all, the web application was successfully developed by utilizing the services provided by Microsoft Azure as a platform. It allowed the developer to gain an in-depth understanding of the various cloud computing forms and the services that are provided by Microsoft Azure, such as the traffic manager, application insights, performance test, continuous deployment, Azure SQL Database, and Blob storage. Furthermore, it allowed the developer to architecturally design an efficient application within the Azure platform as there was a budget constraint of RM 150 set. The developer also gained vast experience in developing a web application using the MVC architecture complemented with the latest technologies such as Entity Framework and ASP.NET Identity System.

References

- Anderson, R. et al., 2017. *Introduction to Identity on ASP.NET Core*. [Online] Available at: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?tabs=visual-studio%2Caspnetcore2x>
- AnyChart, 2017. *Boeing 737*. [Online] Available at: <https://www.anychart.com/products/anymap/gallery/Seat%20Maps/Boeing%20737.php>
- Dwivedi, K., Wheeler, S., Mauderia, J. & Tuliana, J., 2017. *Overview of Traffic Manager*. [Online] Available at: <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview>
- Dwivedi, K., Wheeler, S., Mauderia, J. & Tuliana, J., 2017. *Traffic Manager routing methods*. [Online] Available at: <https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-routing-methods>
- Fiori, A., 2017. *IP geolocation web server*. [Online] Available at: <https://github.com/fiorix/freegeoip>
- Forsyth, S., 2010. *Viewing all Server Variables for a Site*. [Online] Available at: <https://weblogs.asp.net/owscott/viewing-all-server-variables-for-a-site>
- Lavrov, V. & Gordienko, O., 2014. *Ukraine International Airlines, nation's top, conceals ownership*. [Online] Available at: <https://www.kyivpost.com/article/content/legal-quarterly/ukraine-international-airlines-nations-top-conceals-ownership-385153.html>
- LeBlanc, M., Wenzel, M. & Latham, L., 2017. *How to: Request Data Using the WebRequest Class*. [Online] Available at: <https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-request-data-using-the-webrequest-class>
- Myers, T., Lin, C. & Shahan, R., 2017. *Get started with Azure Blob storage using .NET*. [Online] Available at: <https://docs.microsoft.com/en-us/azure/storage/blobs/storage-dotnet-how-to-use-blobs>
- Narumoto, M., Bennage, C., Wasson, M. & Wood, P., 2017. *External Configuration Store pattern*. [Online] Available at: <https://docs.microsoft.com/en-us/azure/architecture/patterns/external-configuration-store>
- Rabeler, C., Popovic, J., Crider, K. & Stein, S., 2017. *What is the Azure SQL Database service?*. [Online] Available at: <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview>

Appendices

Azure Account used: mohammadkazem1996@live.com

GitHub Private Repository: <https://github.com/nhmezakm/UIA-FlightWeb>

Microsoft Teams/OneNote Email: TP035903@mail.apu.edu.my

Traffic Manager URL: uia.trafficmanager.net

SEA Web App URL: uia-asia.azurewebsites.net

West-Europe Web App URL: uia-europe.azurewebsites.net

Microsoft Streams Presentation and Demonstration:

<https://web.microsoftstream.com/video/508f04dc-3404-4406-8997-73c86944b877>