Acknowledgement

I would like to take this opportunity to extend my immense gratitude to my module lecturer, Dr Kalai Anand Ratnam for his endless support, motivation and guidance throughout the course of the module. He is always available to respond to questions and issues that I faced throughout the development process be it in person or through various channels.

The completion of this project would not have been possible without the full support provided by Asia Pacific University, its management and team of lecturers for its students. I thank my course mates and friends alike who have gone through thick and thin together as we work to put our dreams to reality. In addition, I would like to express my love and thanks to my caring family members who have been very supportive in all that I need and wish to do.

Finally, to all whom either have directly or indirectly contributed to the completion of this project, I thank you from the bottom of my heart and I wish the best for all of you.

Thank You!

Yours Sincerely,

Tai Wai Jin, James

1.0 Introduction

1.1. Project Background

Ukraine International Airlines (UIA) is the flagship carrier and largest airline of Ukraine, operating domestic and international passenger flights and cargo services to Europe, Asia, the Middle East, and the United States. Its strategic plan involves expanding into new markets but problems on its existing website that include severe denial-of-service (DOS) attacks and insufficient infrastructures have affected its potential business growth beyond Ukraine due to weak performance and reliability of the website to serve visitors from around the world. As online shoppers are notoriously fickle, if a website lags for even a few seconds, shoppers are just a couple of clicks away from other competitive options.

Historically, UIA have been applying technology to reduce costs, innovate, and improve customer service. A paperless cockpit and sophisticated software for fuel economy analysis are two of many innovations undertaken by UIA. With innovation in its culture, UIA decided to find a breakthrough for its current web challenges by migrating the website out of UIA datacentres into a public cloud. UIA plans to design and develop an Online Flight Booking System and considered both Microsoft Azure and Amazon Web Services. Finally, Azure was chosen as it is also very compatible with open source software.

1.2. Aims

To design and develop an Online Flight Booking System on Microsoft Azure to improve its website performance and reliability for users from around the world.

1.3. Objectives

- To implement measures to scale the site according to its foreseen workload.
- To implement measures to host visitors from around the world with the lowest network latency.
- To implement web application for online flight booking that manages the flight booking process.
- To implement Azure Storage / SQL Databases for secure storage of transactional data.

1.4. Requirement Specifications

The UIA Online Flight Booking System will allow **Visitors**: -

- To search for flights
- To view flight details
- To register a customer account

The UIA Online Flight Booking System will allow **Customers**: -

- To login and logout of the system
- To search for flights
- To view flight details
- To book flights
- To view details of flight bookings
- To edit flight bookings
- To cancel flight bookings

1.5. Functionalities

The UIA Online Flight Booking System is a web application for users to search for UIA flights to various destinations and manage their bookings. It comprises of three modules namely Flight, Booking, and Account.

The Flight module includes Search Flights and View Flights functionalities. Users can perform these functionalities without login into the system. Flights are filtered and returned from the database based on the Origin, Destination, and Date of Departure.

The Booking module provides functionalities to book flights and manage existing bookings. Bookings can be updated in terms of individual tickets and cancellation function is made available for individual tickets or the entire booking. The precondition to perform any of these functionalities is that the user must either login to an existing account or create a new customer account.

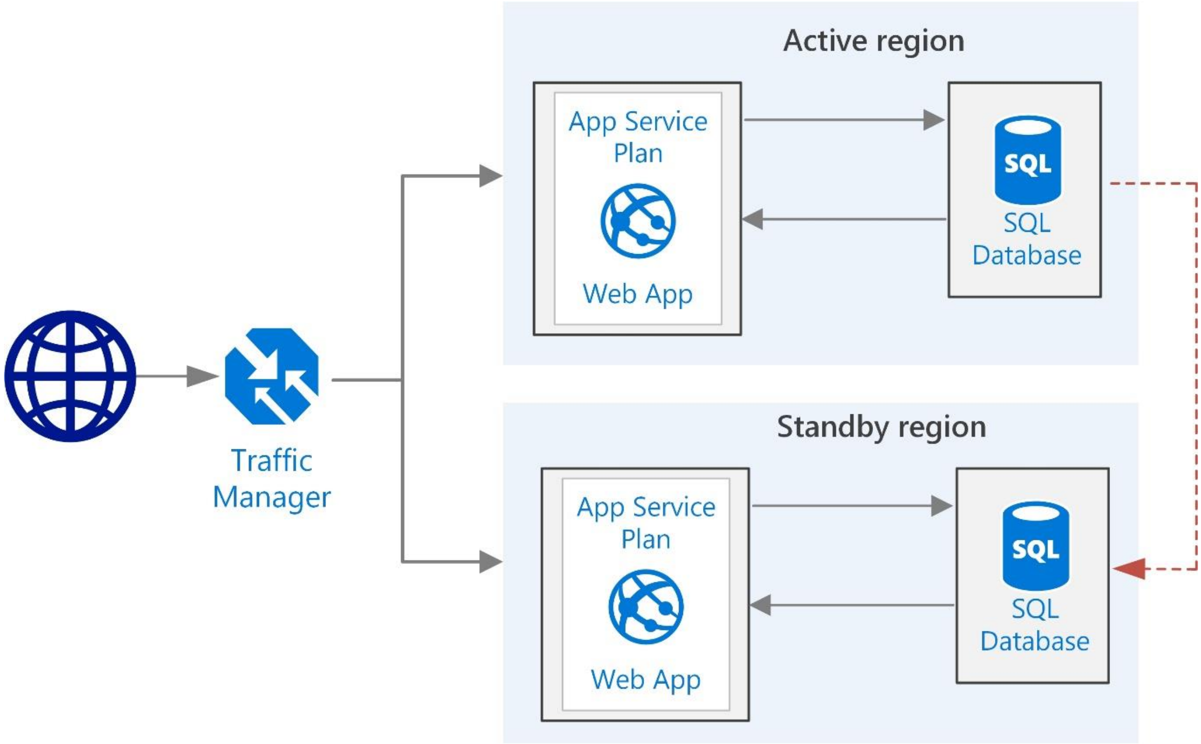
The Account module provides Login, Register, and Change Password functionalities to users. This enforces authentication and security for the different functionalities available on the system.

2.0 Project Plan

WBS	Task Name	Duration	Start	Finish
1	**Ukraine International Airlines (UIA) Online Flight Booking System**	**78 days**	**Wed 02/08/17**	**Fri 17/11/17**
1.1	**Introduction**	**3 days**	**Wed 02/08/17**	**Fri 04/08/17**
1.1.1	Background Information	1 day	Wed 02/08/17	Wed 02/08/17
1.1.2	Objectives	1 day	Thu 03/08/17	Thu 03/08/17
1.1.3	Scopes	1 day	Fri 04/08/17	Fri 04/08/17
1.2	**Analysis**	**6 days**	**Mon 07/08/17**	**Mon 14/08/17**
1.2.1	Requirements Specification	5 days	Mon 07/08/17	Fri 11/08/17
1.2.2	Functions Summary	1 day	Mon 14/08/17	Mon 14/08/17
1.3	**Design**	**14 days**	**Tue 15/08/17**	**Fri 01/09/17**
1.3.1	Cloud Design Patterns	3 days	Tue 15/08/17	Thu 17/08/17
1.3.2	Architectural Diagrams	5 days	Fri 18/08/17	Thu 24/08/17
1.3.3	Design Considerations	1 day	Fri 25/08/17	Fri 25/08/17
1.3.4	Modelling	5 days	Mon 28/08/17	Fri 01/09/17
1.4	**Implementation**	**42 days**	**Mon 04/09/17**	**Tue 31/10/17**
1.4.1	Setup Source Control Repository	1 day	Mon 04/09/17	Mon 04/09/17
1.4.2	Local Web Application Development	30 days	Mon 11/09/17	Fri 20/10/17
1.4.3	Publish Application to Azure	1 day	Mon 23/10/17	Mon 23/10/17
1.4.4	Implement Azure Storage / SQL Database	1 day	Tue 24/10/17	Tue 24/10/17
1.4.5	Application Scaling	2 days	Wed 25/10/17	Thu 26/10/17
1.4.6	Investigate & Analyse Application	3 days	Fri 27/10/17	Tue 31/10/17
1.5	**Testing**	**45 days**	**Mon 04/09/17**	**Fri 03/11/17**
1.5.1	Test Plan	5 days	Mon 04/09/17	Fri 08/09/17
1.5.2	Unit Testing	6 days	Mon 23/10/17	Mon 30/10/17
1.5.3	Performance Testing	3 days	Wed 01/11/17	Fri 03/11/17
1.6	**Documentation**	**10 days**	**Mon 06/11/17**	**Fri 17/11/17**
1.6.1	Compilation	5 days	Mon 06/11/17	Fri 10/11/17
1.6.2	Video Production	5 days	Mon 13/11/17	Fri 17/11/17

3.0 Design

3.1. Architectural Diagram



3.2. Design Considerations

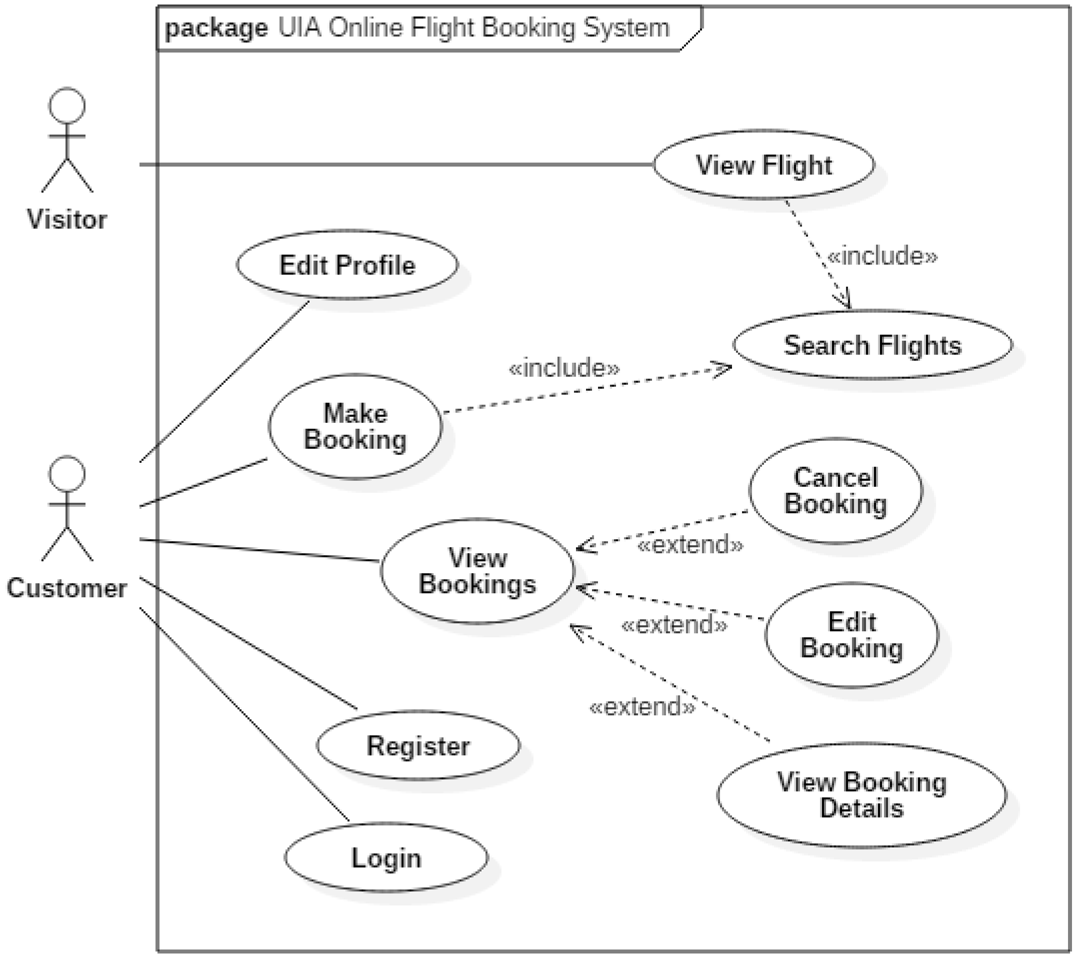
The design of the UIA Online Flight Booking System requires consideration for the design of both the web application and the cloud solution to be hosted on cloud.

In the context of the web application, it is a single tenant (Ukraine International Airlines) web application that manages the flight booking process that includes searching for flights, booking a flight, editing details of the booking and cancelling a booking. Thus, there are two physical users of the web application, namely Visitor and Customer. The design of the system is based off the assumption that there is an existing Flight Management System in used by UIA that provides the data required for the operation of the Online Flight Booking System (Airports, Flights, Planes, and Tariff information). Furthermore, a Payment class has been catered for in the design to allow for future integration of payment facilities for the flight bookings. On top of that, the ApplicationUser class inherits the IdentityUser provided in the ASP.NET Framework library that allows for future expansion of the system to include new user roles.

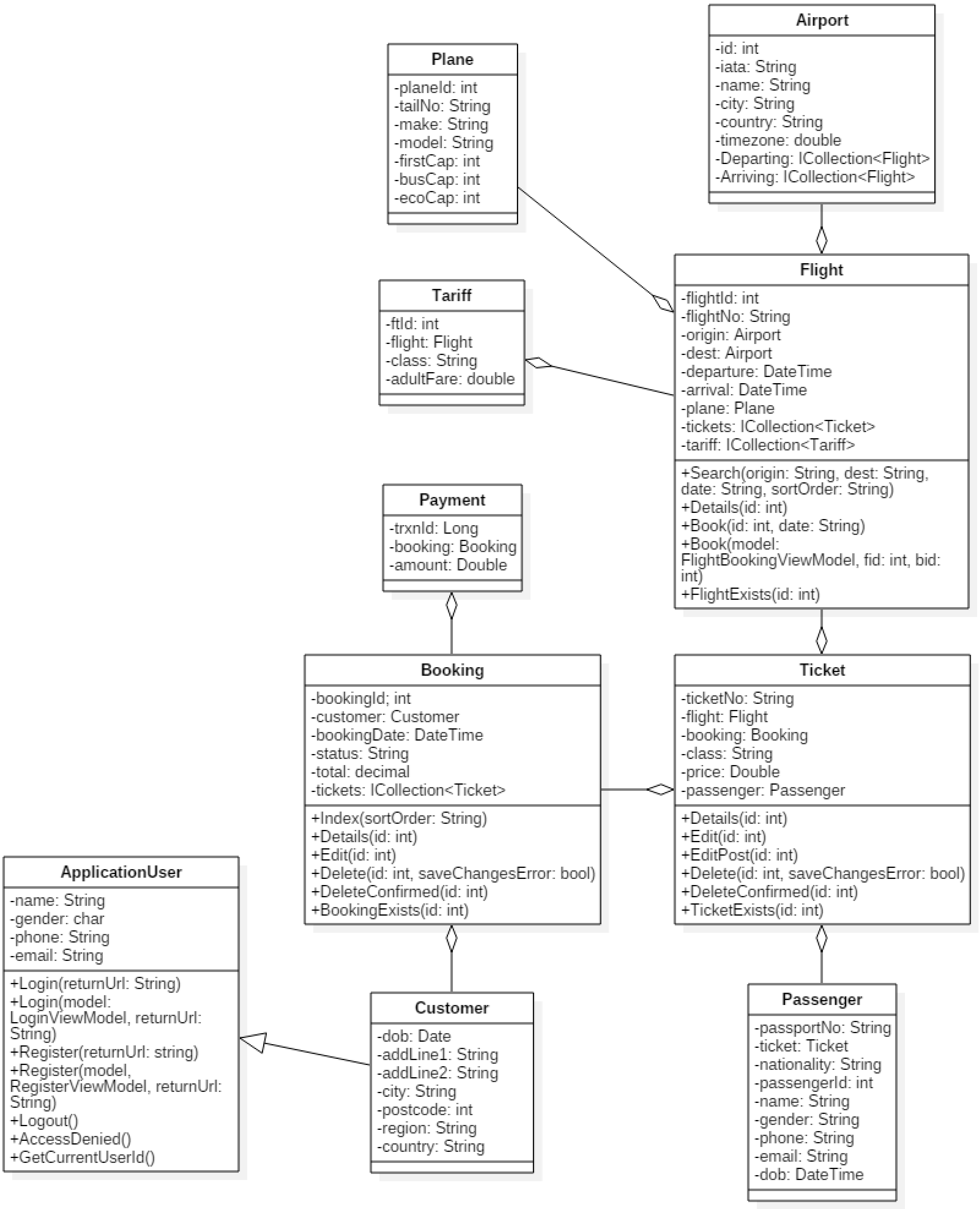
Going over to the design of the cloud implementation on Microsoft Azure, the developer work with an assumed budget of RM150 per month in which services and pricing tiers must be balanced to make the most out of the allocation. The services that would be applied as introduced in the architectural diagram in section 3.1 are the App Service web app, SQL Database and Logical Server, and Traffic Manager. App Service Web Apps allows for the hosting of web applications of various programming languages on the cloud with security, load balancing, auto-scaling and high availability. This allows us to host the Online Flight Booking System on Azure and capitalise on the pay-as-you-use infrastructure to improve availability and performance. Azure SQL Database is a relational database-as-a-service in the cloud and hosted on a Logical Server. It allows for the data associated with the application to be hosted securely with failover policies and backup mechanisms. Finally, Traffic Manager controls the distribution of requests from web clients to Azure App Service web apps and monitor continuously the endpoints' health to provide automatic failover as required. This component ensures that the site is highly available and accessible with lowest network latency.

3.3. Modelling

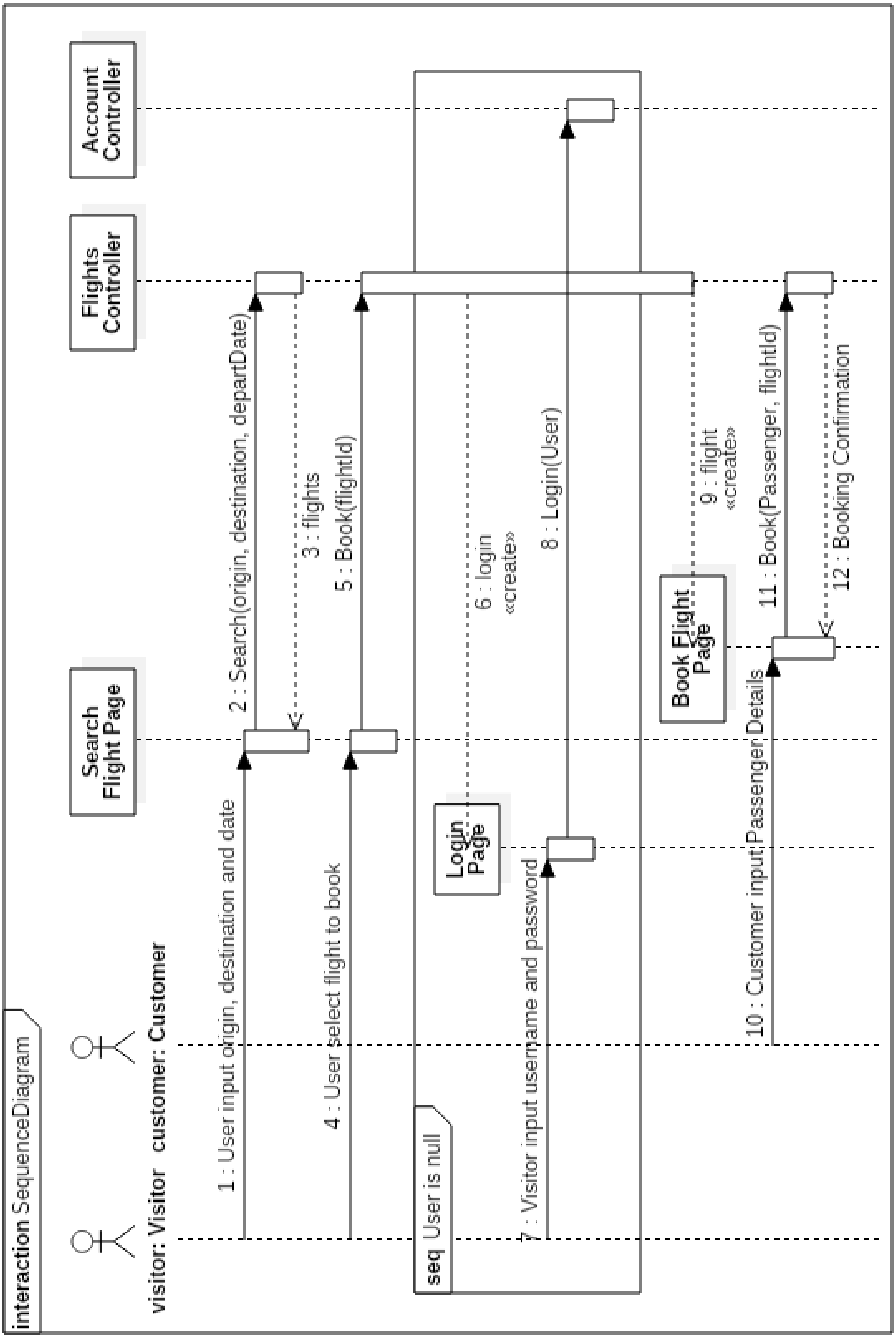
3.3.1. Use Case Diagram



3.3.2. Class Diagram



3.3.3. Sequence Diagram



3.4. Web Application User Interface

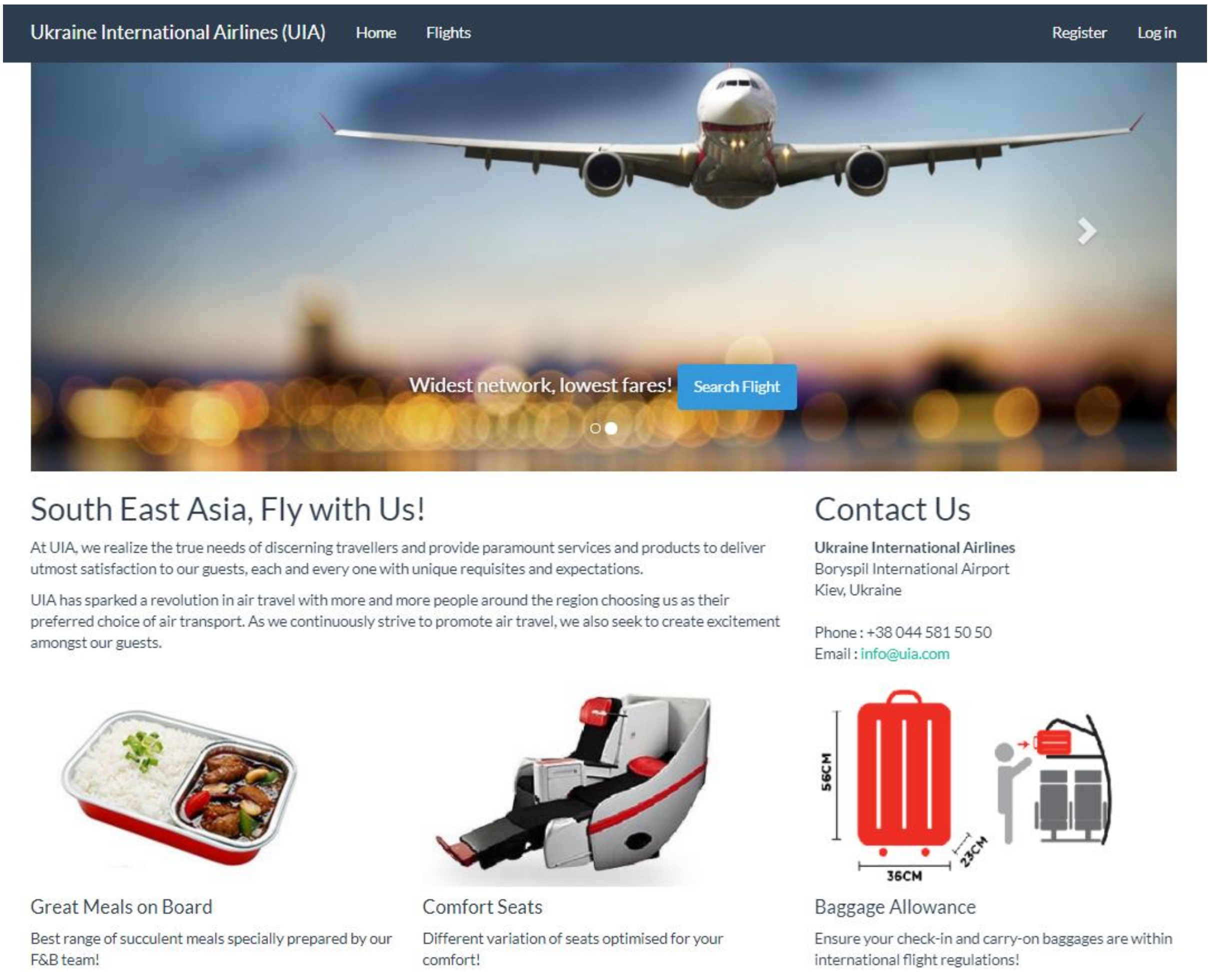


Figure 3.4.1 Home Page User Interface

Search Flights

Origin: Glasgow International Airport ▾ Destination: Kota Kinabalu International Airport ▾ Departure Date: 18/11/2017 Search

Glasgow International Airport (GLA) >>> Kota Kinabalu International Airport (BKI)

Flight No	Departure Date Time	Arrival Date Time	Economy Class Fare	Business Class Fare	First Class Fare	
BA893	Sat, 18/11/2017, 1930 hrs	Sun, 19/11/2017, 0158 hrs	$1,101.00	$2,752.50	$11,010.00	Book
BA893	Mon, 20/11/2017, 1930 hrs	Tue, 21/11/2017, 0158 hrs	$1,258.00	$3,145.00	$12,580.00	Book
BA893	Mon, 27/11/2017, 1930 hrs	Tue, 28/11/2017, 0158 hrs	$376.00	$940.00	$3,760.00	Book
MH21	Thu, 30/11/2017, 0835 hrs	Thu, 30/11/2017, 1455 hrs	$1,263.00	$3,157.50	$12,630.00	Book
BA893	Fri, 01/12/2017, 1930 hrs	Sat, 02/12/2017, 0158 hrs	$669.00	$1,672.50	$6,690.00	Book

Figure 3.4.2 Search Flights User Interface

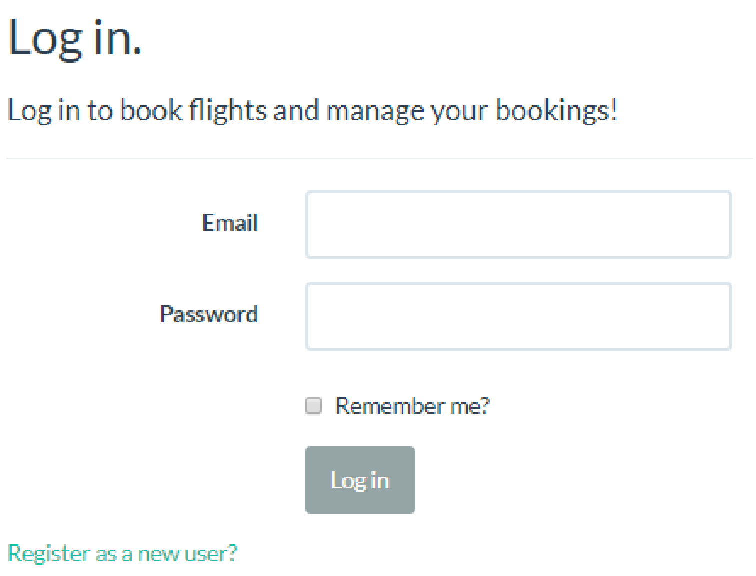


Figure 3.4.3 User Login User Interface

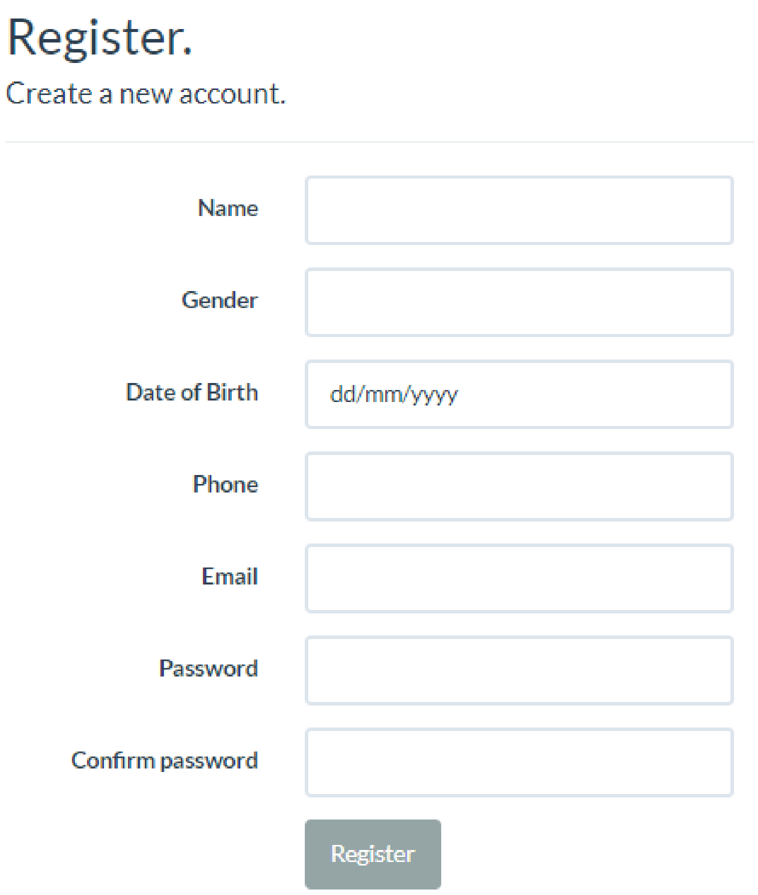


Figure 3.4.4 User Registration User Interface

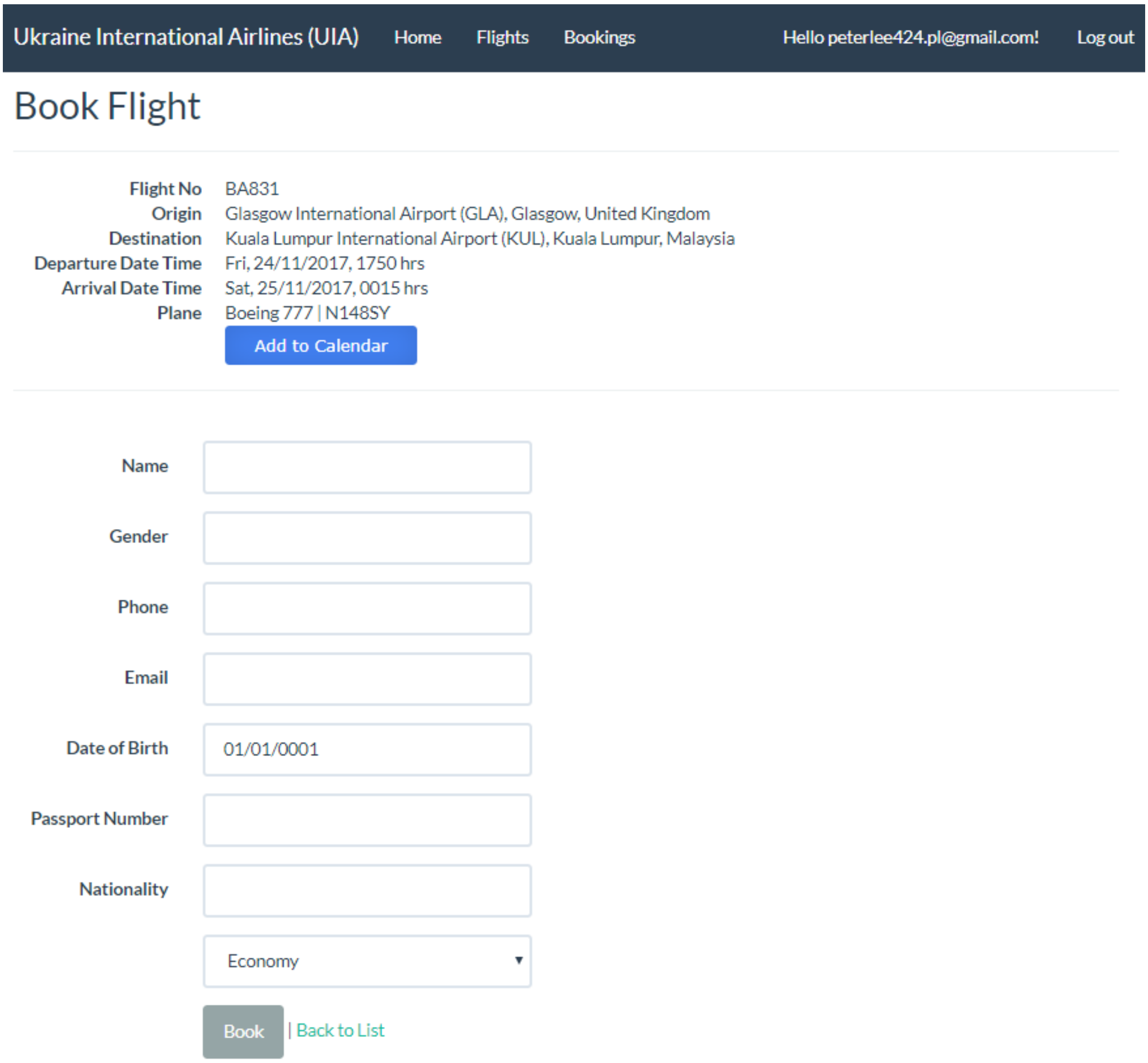


Figure 3.4.5 Book Flight User Interface

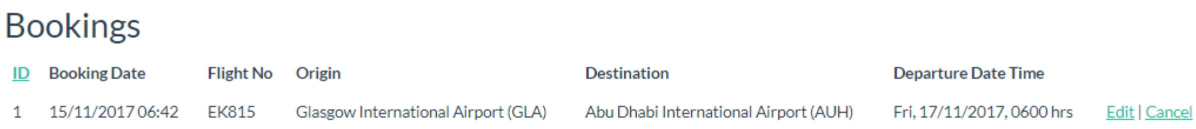


Figure 3.4.6 My Bookings User Interface

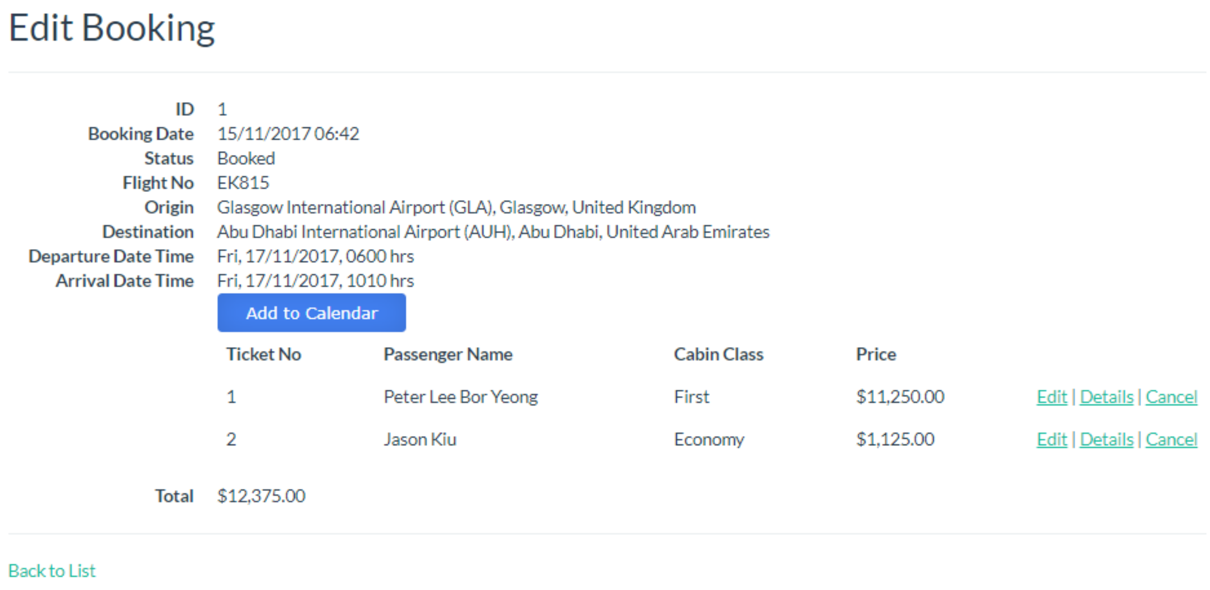


Figure 3.4.7 Edit Booking User Interface

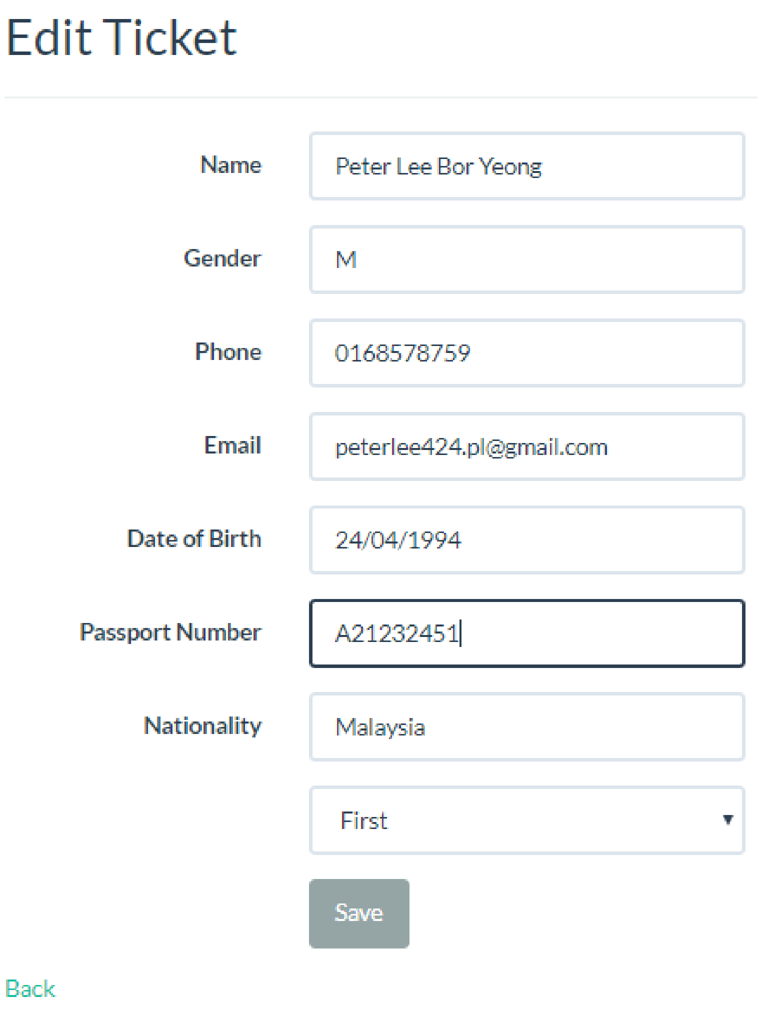


Figure 3.4.8 Edit Ticket User Interface

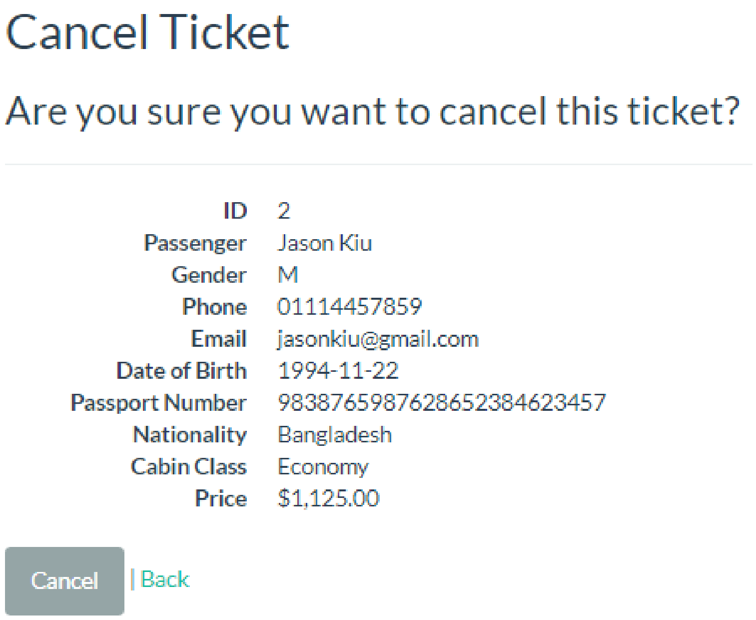


Figure 3.4.9 Cancel Ticket User Interface

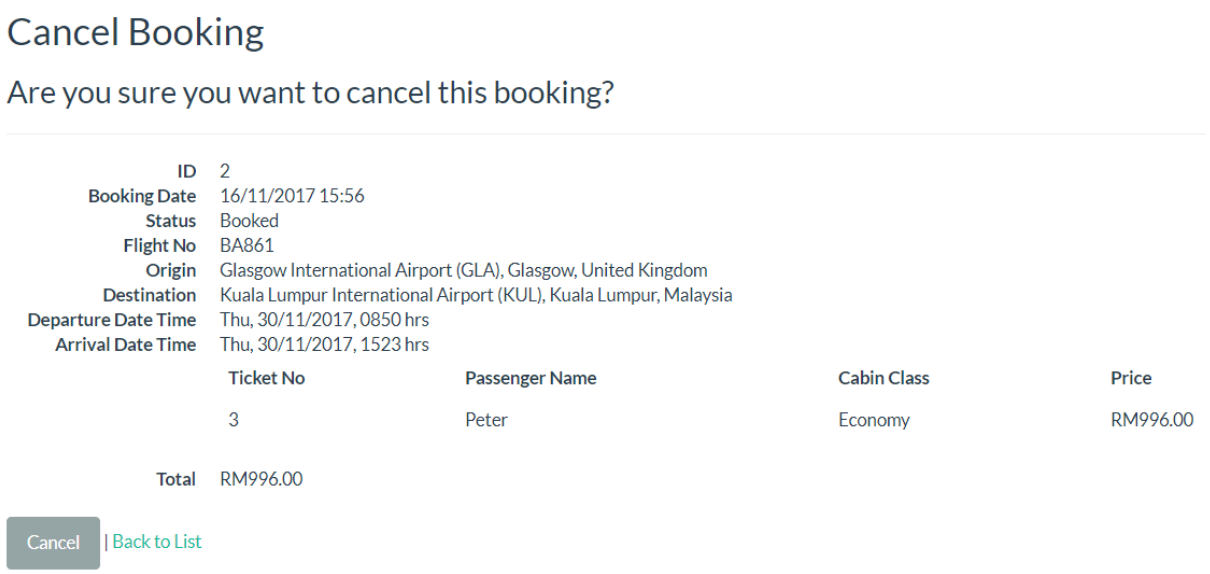


Figure 3.4.10 Cancel Booking User Interface

4.0 Implementation

4.1. Publishing an Application to Microsoft Azure

The web application for the UIA Online Flight Booking System was developed using ASP.NET Core 1.1 which is a cross-platform and open source framework with Microsoft Visual Studio Enterprise 2017. The web application uses the Model-View-Controller (MVC) pattern and Entity Framework which is an object-relational mapper that facilitates application development with database and eliminates the need to write data access codes. With Entity Framework, objects or Models in the application are automatically mapped to tables in the database while the attributes of the objects are mapped to columns in the tables.

A Web App is created on Microsoft Azure with the details as shown in figure 4.1.1. A new resource group is created as a logical container for Azure resources. In addition, a new App Service plan is created for the Web App as shown in figure 4.1.2. Here, the pricing tier, S1 Standard is selected (figure 4.1.3) in consideration of the balance between price, performance, storage, and scalability. Furthermore, traffic manager which is a core component of the architecture is only available from the Standard tier upwards. Once the configurations have been defined, the Web App is created.

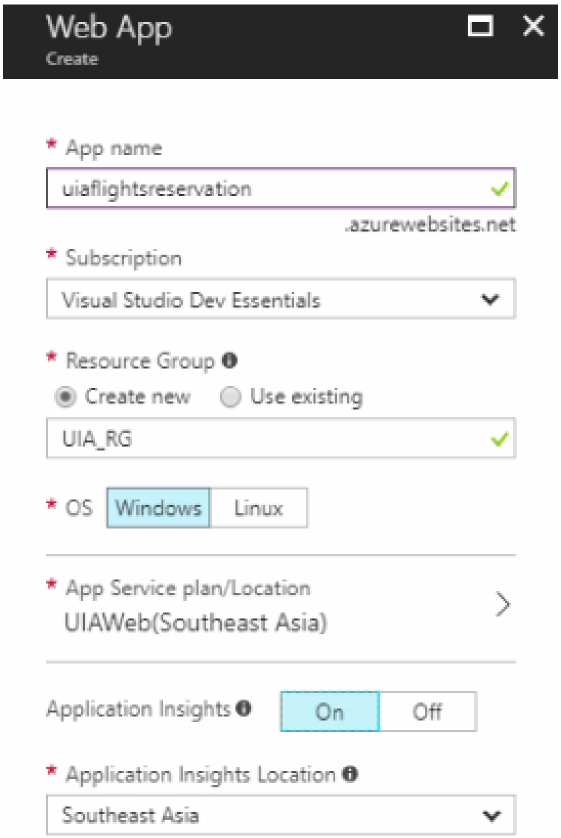


Figure 4.1.1 Web App Creation on Azure Portal

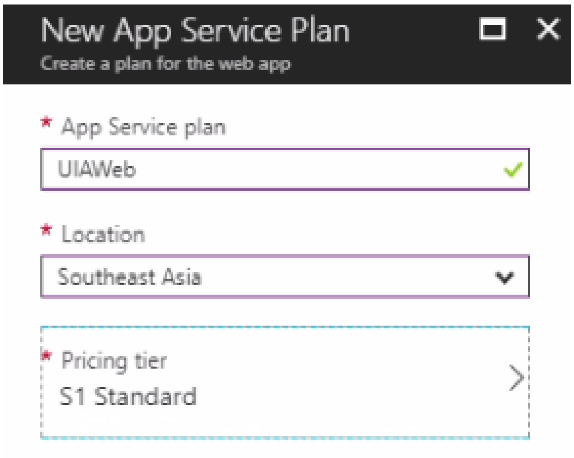


Figure 4.1.2 App Service Plan Creation

Figure 4.1.3 Standard and Basic Pricing Tier

Following the deployment of the web app resources, the application connection string for the database is changed from the local Microsoft SQL Server database to the Azure SQL database (figure 4.1.4) created prior (section 4.4) in the appsettings.json file of the MVC web application solution. The same connection string is added in the Application Settings of the App Service as shown in figure 4.1.5.

```
"ConnectionStrings": {
    //"DefaultConnection": "Server=JAMEST-PC\\SQLEXPRESS;Database=UIAReservation;Trusted_Connection=True;MultipleActiveResultSets=true"
    "DefaultConnection": "Server=tcp:flyuia.database.windows.net,1433;Initial Catalog=UIAFlightsReservation;Persist Security Info=False;User ID=uiauser;Password=Pass@1234;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
}
```

Figure 4.1.4 Connection Strings Configuration in AppSetttings.json

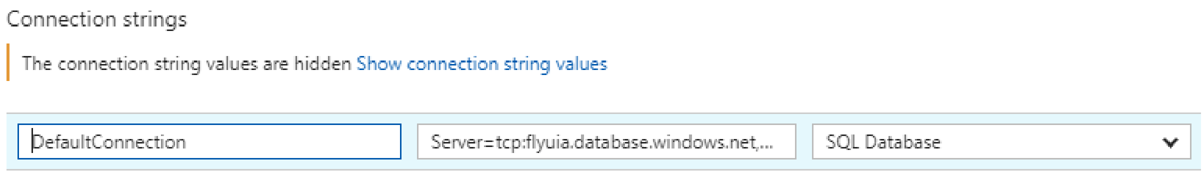


Figure 4.1.5 Connection Strings Configuration in Azure Portal

Once the connection strings configuration has been revised, the application was published to the previously created App Service from within Visual Studio Enterprise 2017 as illustrated in figure 4.1.6.

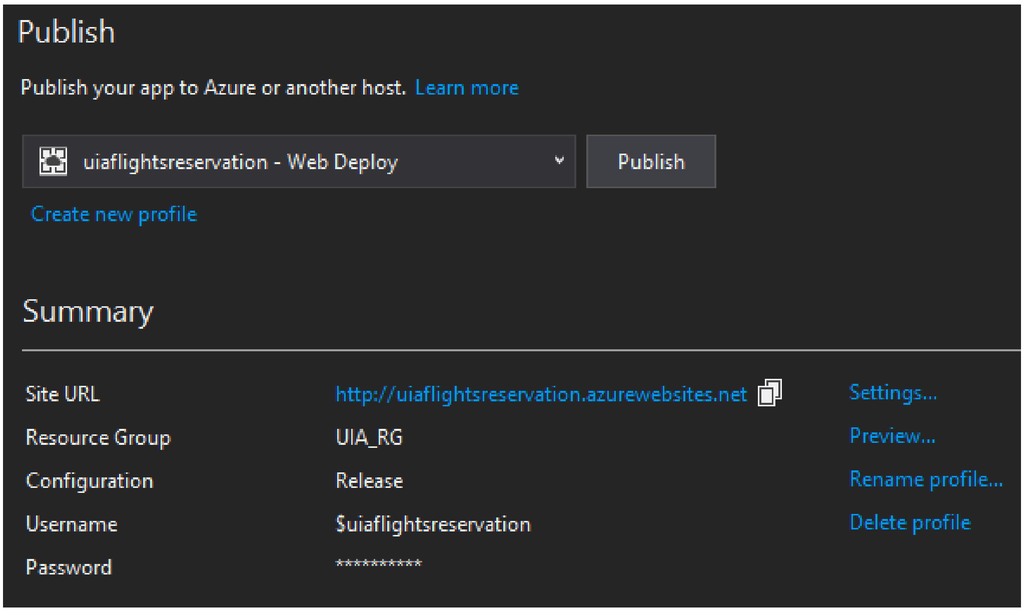


Figure 4.1.6 Visual Studio Enterprise Publish Window

While publishing to the targeted profile, Entity Framework prepares the necessary migration codes to establish the required tables in the SQL Database which is reflective of the models in the solution. Four tables, Airports, Flights, Planes, Tariffs have to be fed with the prepared datasets which was done by executing INSERT INTO queries targeting these tables. The site is tested with the prepared test cases upon completion of the queries.

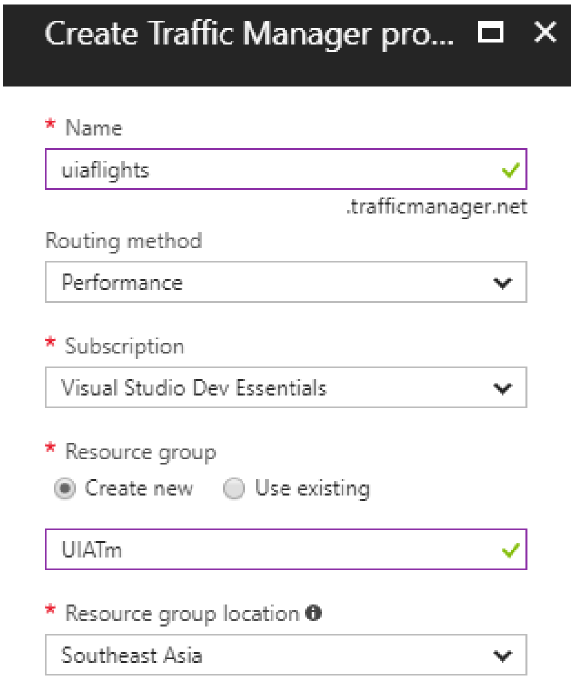


Figure 4.1.7 Traffic Manager Profile Creation on Azure Portal

The Azure Traffic Manager is used to control the distribution of requests from web clients to Azure App Service web apps and monitor continuously the endpoints' health to provide automatic failover as required (Lin & Che, 2016). Endpoints of two web apps, hosted in different regions, were added to the Azure Traffic Manager profile and tracked in terms of its status (running, stopped, or deleted). The Performance routing method was selected where the closest web app in terms of the lowest network latency among web apps in different geographic locations is used. Other routing methods include;

- Geographic – users are directed to specific web apps based on the DNS query origin geographical location.
- Priority – a primary web app serves all traffic and in the event of failure, backups are used.

- Weighted – requests are evenly, or weight distributed across a collection of web apps.

The implementation of traffic manager helps UIA to improve the availability of its Online Flight Booking System in which the endpoints are monitored, and automatic failover occurs when an endpoint stops. In addition, it improves the responsiveness of the application by having it hosted in data centres around the world, thus directing client traffic to the endpoint with lowest network latency. Instead of accessing the site independently with its URL (uiaflightsreservation.azurewebsites.net / uiaflightsreservations.aszurewebsites.net), users can access the site via the DNS level URL (uiaflightsreservation.trafficmanager.net) which redirects users according to the Performance routing method selected.

4.2. Application Scaling

The web app that we published to the App Service runs in an app service plan in a certain region which defines a collection of resources for the app's execution in that region and can be shared with more than one app. These set of resources and features allocated for the App Service plans differ according to the pricing tier which are mainly categorised into shared and dedicated compute.

In shared compute which provides the Free and Shared base tiers, apps share the same Azure VM with apps of other customers. CPU quotas are allocated to apps of these tiers running on shared resources that are cannot scale out. In dedicated compute on the other hand, tiers such as Basic, Standard, Premium, and PremiumV2 have the apps running on dedicated Azure VMs where only apps of the same App Service plan share the compute resources. The pricing is based on the tier chosen and the size and number of running instances. The table 4.2.1 below shows a comparison between dedicated compute tiers (Microsoft, n.d.).

Tiers	Basic	Standard	Premium
Apps	Unlimited	Unlimited	Unlimited
Disk space	10 GB	50 GB	250 GB
Max instances	Up to 3	Up to 10	Up to 20
SLA	99.95%	99.95%	99.95%
Always On	X	X	X
Auto Scale		X	X
Backup/Restore		X	X
Custom Domains	X	X	X
SSL (IP/SNI)	X	X	X
Traffic Manager		X	X

Table 4.2.1 Comparison between Dedicated Compute App Service Pricing Tiers

The Basic service plan is conceptualised for application with lower traffic requirements without the need for advanced auto-scale and traffic management capabilities. However, it does come built-in with automatic network load-balancing support across instances.

INSTANCE	CORES	RAM	STORAGE	PRICE / month
B1	1	1.75 GB	10 GB	~RM248.31
B2	2	3.50 GB	10 GB	~RM496.62
B3	4	7 GB	10 GB	~RM993.24

The Standard service plan on the other hand is designed to run production workloads with built-in network load-balancing support that automatically distributes traffic across instances. In addition, auto-scale functionality is made available in which the number of virtual machine instances running can be automatically adjusted to match the workloads.

INSTANCE	CORES	RAM	STORAGE	PRICE / month
S1	1	1.75 GB	50 GB	~RM331.08
S2	2	3.50 GB	50 GB	~RM662.16
S3	4	7 GB	50 GB	~RM1,324.32

Lastly, The Premium service plan is designed for production apps that requires enhanced performance by introducing faster processors, SSD storage and double memory-to-core ratio in comparison with the Standard service plan. In addition, scaling capabilities are boosted via increased instance count, while still providing all the advanced capabilities available on the Standard plan.

INSTANCE	CORES	RAM	STORAGE	PRICE / Month
P1v2	1	3.50 GB	250 GB	~RM993.24
P2v2	2	7 GB	250 GB	~RM1,986.48
P3v2	4	14 GB	250 GB	~RM3,972.96

The S1 Standard pricing tier was selected when creating the App Service plan in section 4.1 due to the optimum balance of;

- Price – The estimated monthly charges of RM331.08 per month is within the budget of the project as the proof of concept would only require the service to run for a few days. The cost for Premium plans would be too expensive.
- Performance – A single core with 1.75 GB of RAM is sufficient for the application and can be scaled as required after conducting performance testing either to scale up or scale out.
- Scalability – The ability to scale up to 10 instances compared to 3 instances for the Basic plan indicates that the plan is more scalable and can accommodate more load in comparison with the Basic plan.

- Features (Traffic Manager) – Features such as Traffic Manager and Auto Scale are other reasons the Standard plan premier over the Basic plan.

However, upon conducting performance tests that simulated concurrent user loads as described in section 5.3, the monitoring mechanisms of the web app (section 4.3) suggested that the percentage of CPU usage had reached up to 90.9%. Thus, the application should be analysed and scaled accordingly in order to cope with the spike in workload.

There are two ways in which the App Service can be scaled, namely Up and Out (Wasson, 2016). Scaling up refers to increment of the instance size which in turn determines the number of cores, memory, and storage. This can be performed by changing the size of the instance or the plan tier. On the other hand, scaling out involves adding instances to cope with the increase in load. This can be performed by changing the count of the instances or to enable auto scaling for automatic increase or decrease in instances according to a schedule and/or performance metrics.

In reference to the scenario faced, scale out is the optimum solution to handle the increased load by spreading the incoming requests to the application across all available instances. A higher number of instance would be able to reduce the consistent high levels of CPU usage on a single instance. Besides, the application must handle high request loads which can be shared by multiple instances. However, as these increase in resource usage occurs during peak traffic hours, rule-based autoscaling is applied to respond to the hike in workload in which CPU usage was used as a metric for autoscale rules.

4.3. Investigate & Analyse Application

Performance issues in the UIA Online Flight Booking System can severely impact business. In the event of promotions or sales where the ticket prices are sold at a cheaper rate, there is bound to be an increase in the concurrent users load on the application due to more visitors and customers interested in searching for flights and booking tickets. However, the application must be able to scale itself to meet these needs for additional resources. Furthermore, the users may discover issues that were not identified in testing and these issues should be alerted to the developers equipped with diagnosis and troubleshooting tools. In addition, issues in the application may be due to the underlying infrastructure on which the applications run. Microsoft Azure provides a collection of tools to identify and resolve such issues (Jr, et al., 2017).

The Azure App Service provides monitoring functions that include metrics indicating the behaviour of the application which allow for alerts and scaling to be configured (Lin & Tardif, 2016). On the overview panel of the App Service, metrics (figure 4.3.1) for the past hour highlight HTTP Server Errors, Data In, Data Out, Requests, and Average Response Time. These allows us to identify behavioural patterns and issues in the app service. The number of HTTP Server Errors may indicate an issue in the application that require further investigation. It may be due to the heavy load of concurrent users trying to search flights at the same time thus there may not be sufficient DTUs to handle the requests. Detailed metrics for additional metrics such as Availability, Percentage of CPU Usage and Percentage of Memory Usage for the past 24 hours are also available through the diagnose panel.

Metrics	Definition
Average Response Time	The average time taken for the app to serve requests in ms.
Data In	The amount of incoming bandwidth consumed by the app in MiBs.
Data Out	The amount of outgoing bandwidth consumed by the app in MiBs.
Http Server Errors	Count of requests resulting in an HTTP status code >= 500 but < 600.
Requests	Total number of requests regardless of their resulting HTTP status code.
Availability	The percentage in which the app is available to serve requests.
CPU Usage %	The percentage of overall CPU usage
Memory Usage %	The percentage of memory used across the instance.

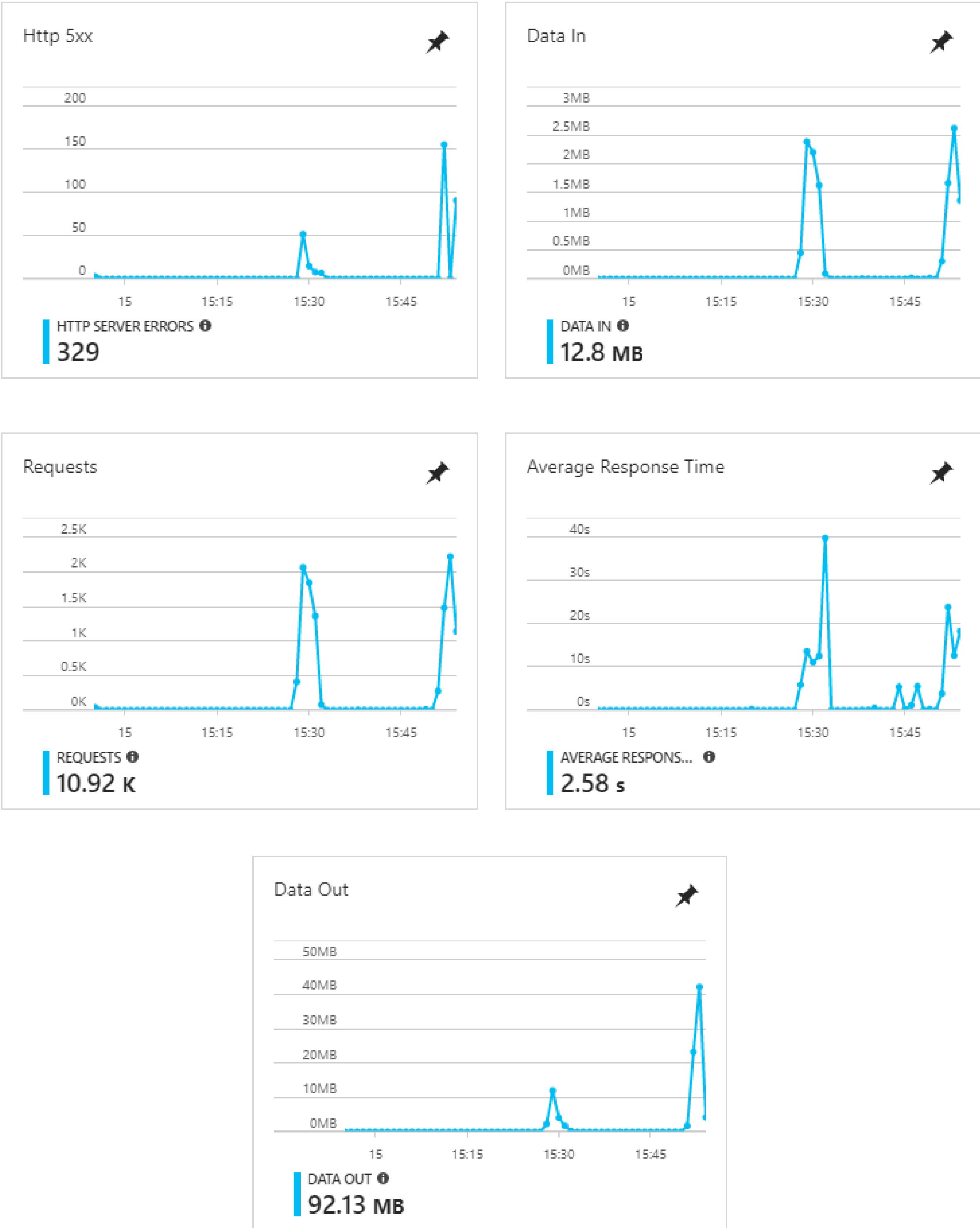


Figure 4.3.1 Instrumentation Data from App Service Overview

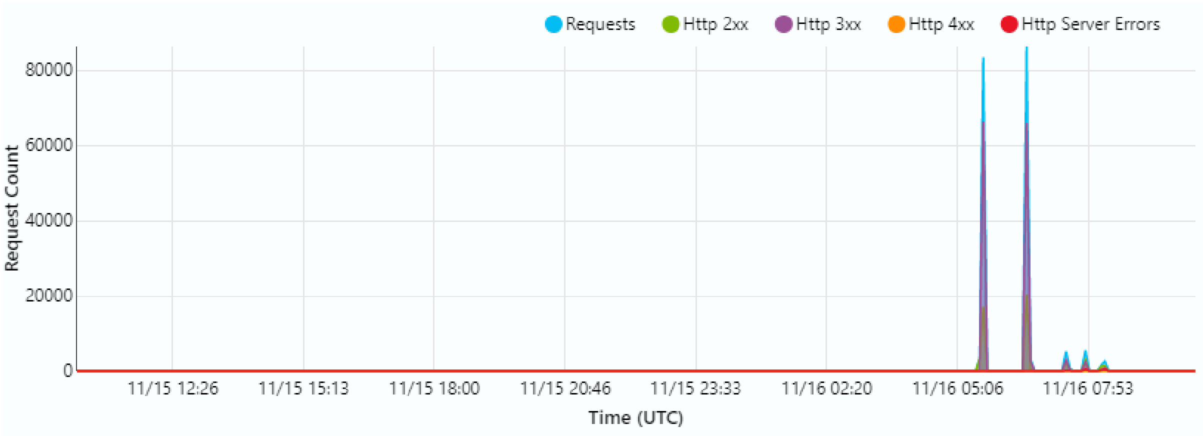


Figure 4.3.2 Time against Request Count

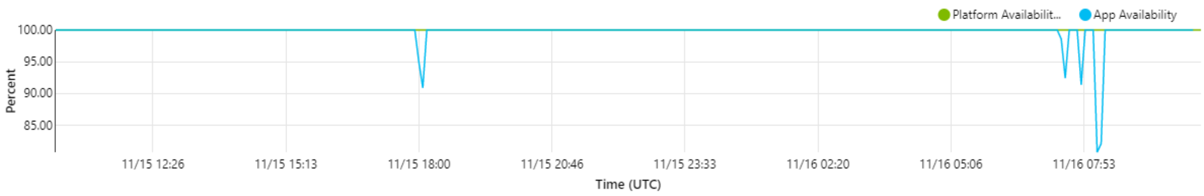


Figure 4.3.3 Time against Percentage of App Availability

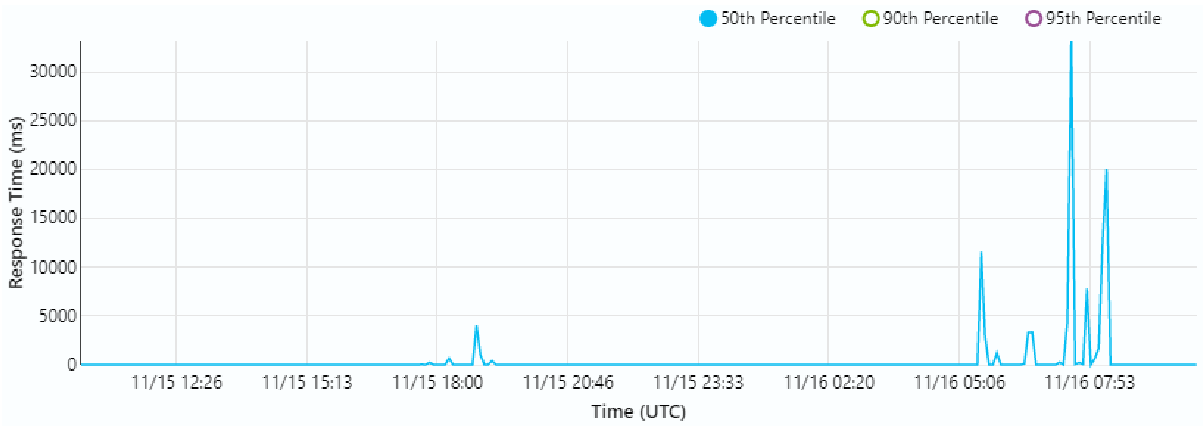


Figure 4.3.4 Time against Response Time

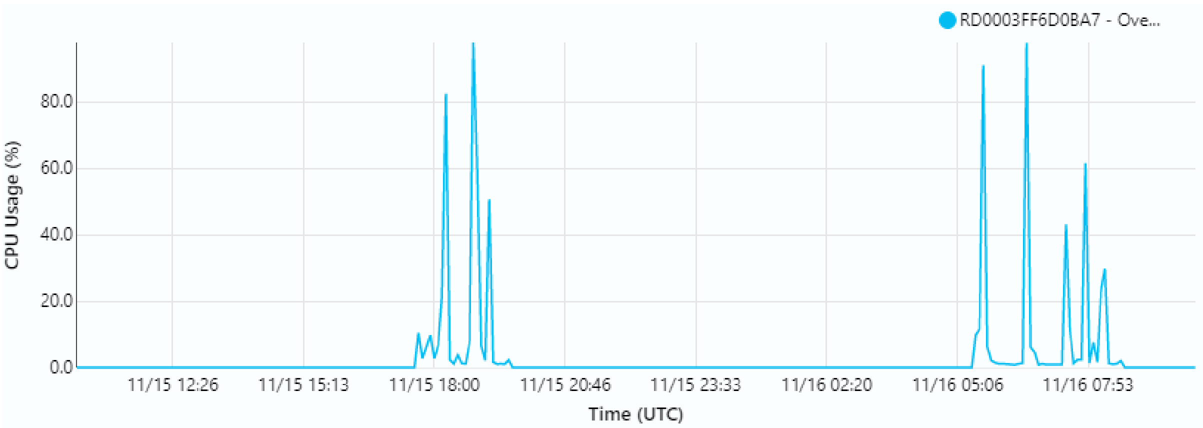


Figure 4.3.5 Time against Percentage of CPU Usage

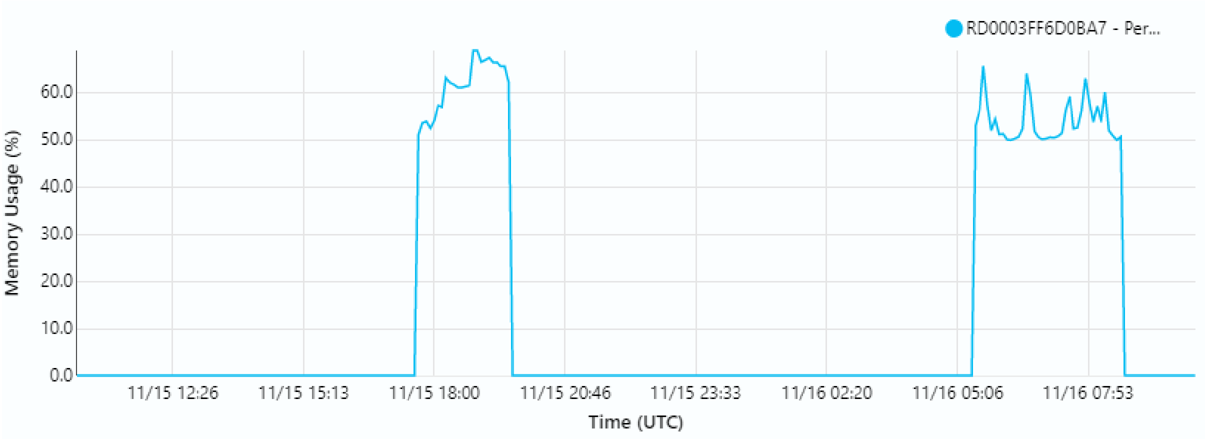


Figure 4.3.6 Time against Percentage of Memory Usage

In addition, Application Insights monitors the performance of the application and user analytics. The figure 4.3.7 below shows one of its monitors which track the application's slowest requests. This monitor allows the developer to identify which request is taking the most time for the application to respond and investigate its cause. In this scenario, the GET Flights/Search request that is taking the most time of up to 10.2 seconds to search for a flight may be due to the performance tests that were conducted with high load of concurrent users thus causing congestion in the application. The other requests involve retrieval of static files and may suggest for the implementation of a cloud design pattern to host static content separately to improve performance.

Slowest Requests (past 24 hours)

REQUEST NAME	DURATION (95TH)
GET Flights/Search	10.2 sec
GET /images/banner1.jpg	1.66 sec
GET /images/card4.jpg	570 ms
GET /images/banner2.jpg	569 ms

Figure 4.3.7 Slowest Requests for the past 24 hours

4.4. Implementation & Discussion of Managed Databases

Azure SQL Database is a cloud database service built for app developers. It's the only database as a service that scales on the fly without downtime and helps you to deliver multi-tenant apps efficiently, ultimately giving you more time to innovate and accelerating your time to market. SQL Database's built-in intelligence quickly learns your app's unique characteristics and dynamically adapts to maximise performance, reliability and data protection (Microsoft, n.d.).

A SQL Database, UIAFlightsReservation was created on Microsoft Azure as illustrated in figure 4.4.1 and a new server was created in the Southeast Asia region as a logical server in which the database resides (figure 4.4.2). The Standard S0 pricing tier was selected with 10 DTUs and 250 GBs of storage as illustrated in figure 4.4.3.

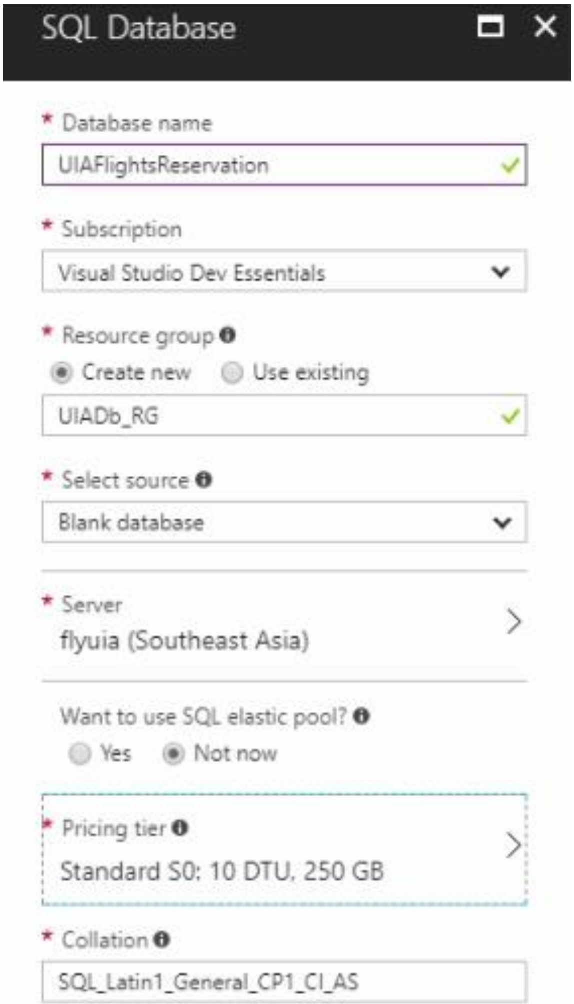


Figure 4.4.1 SQL Database Creation on Azure Portal

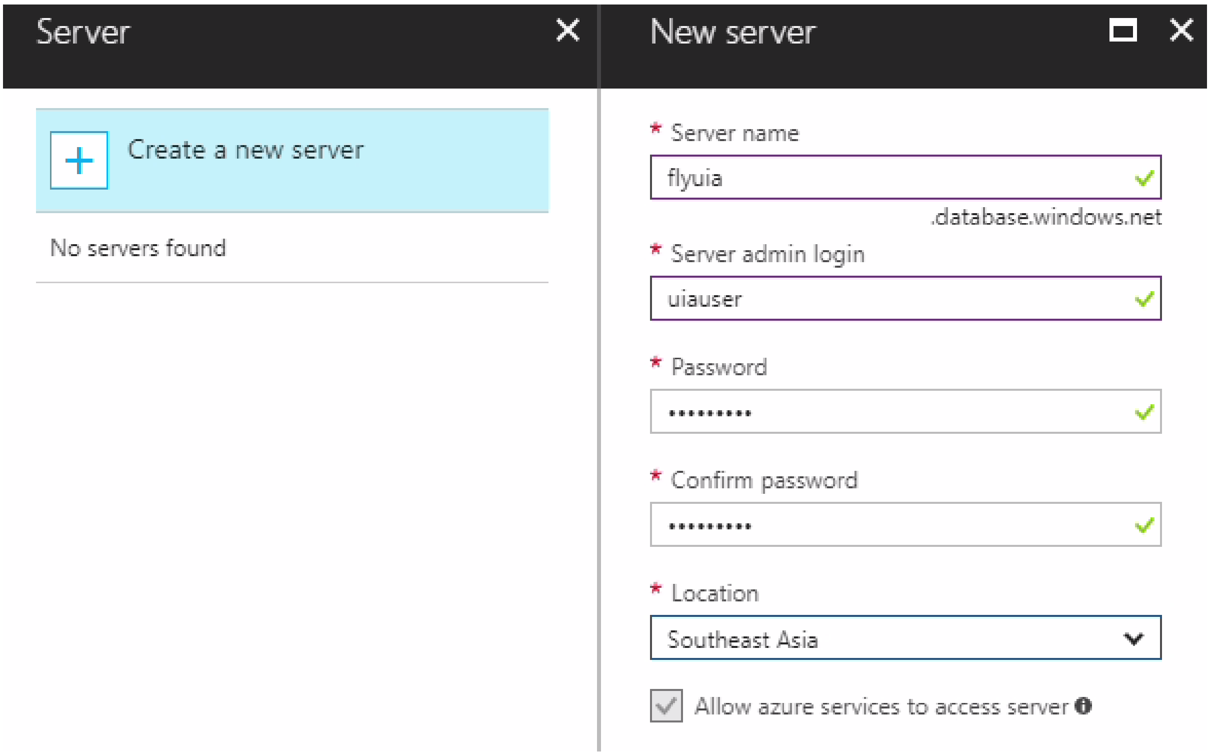


Figure 4.4.2 SQL Logical Server Creation on Azure Portal

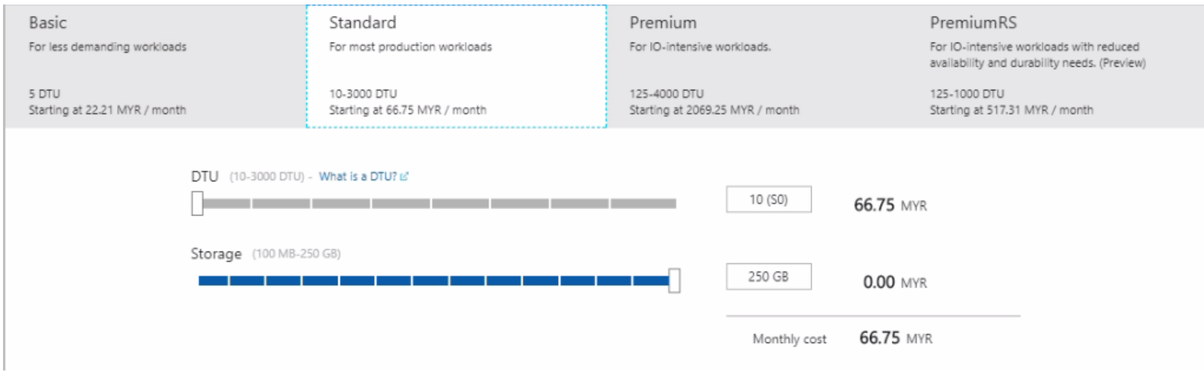


Figure 4.4.3 SQL Database Pricing Tier Selection

Table 4.4.1 details the comparison between single databases of the Basic, Standard, and Premium pricing tier (Microsoft, n.d.). These completely isolated databases are best for predictable performance workloads and can be scaled for the required performance and features as required. The performance is mainly measured in Database Transaction Units (DTUs).

Plan	Basic	Standard	Premium
Target workload	Development and production	Development and production	Development and production
Uptime SLA	99.99%	99.99%	99.99%
Backup retention	7 days	35 days	35 days
CPU	Low	Low, Medium, High	Medium, High
IO throughput	Low	Medium	Order of magnitude higher than Standard
IO latency	Higher than Premium	Higher than Premium	Lower than Basic and Standard
Columnstore indexing and in-memory OLTP	N/A	N/A	Supported

Table 4.4.1 Comparison between Single Databases of Basic, Standard, and Premium

TIERS	DTUs	INCLUDED STORAGE	MAX STORAGE	PRICE FOR DTUs AND INCLUDED STORAGE
S0	10	250 GB	250 GB	~RM66.76/month
S1	20	250 GB	250 GB	~RM133.50/month
S2	50	250 GB	250 GB	~RM333.84/month
S3	100	250 GB	1 TB	~RM667.50/month

Table 4.4.2 Standard Single Database Pricing Tiers

After the database has been successfully deployed, a secondary readable database is established in the West US 2 region using the Active Geo-Replication function as illustrated in figure 4.4.4. As it is in a separate region, a new server was created (figure 4.4.5). The secondary database provides the functionalities for querying and manual failover in the event of an outage or failure of the primary database. It operates in snapshot isolation model so that the replication of the primary database updates does not get delayed by queries executing on the secondary database.

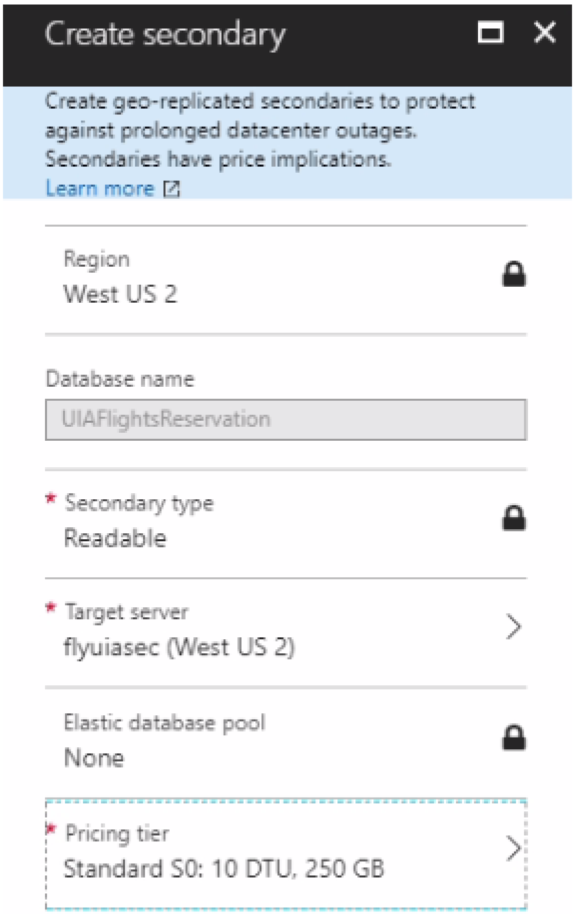


Figure 4.4.4 Secondary SQL Database Creation

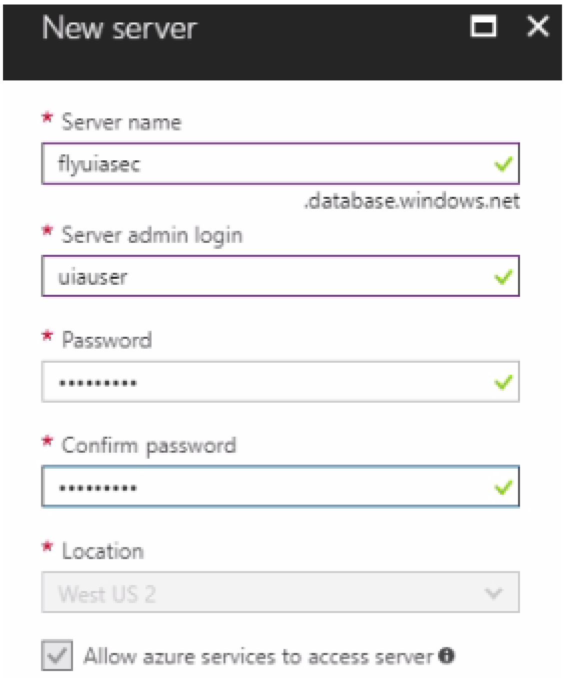


Figure 4.4.5 Secondary Database Server Creation

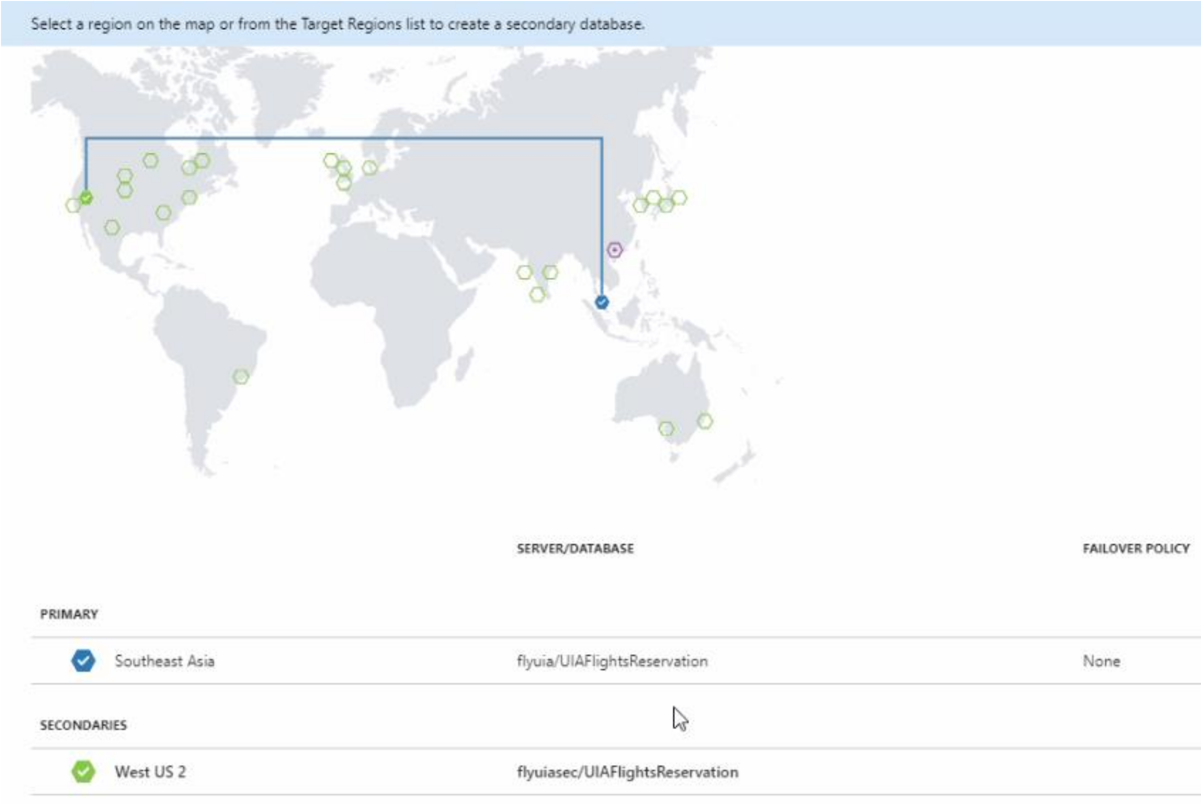


Figure 4.4.6 Active Geo-Replication Visualisation on Azure Portal

5.0 Test Plan & Testing Discussion

5.1. Unit Testing

Unit testing were conducted between 23rd October 2017 and 30th October 2017 for the following units based on the test case developed.

1. Login
2. Registration
3. Change Password
4. Search Flights
5. Book Flight
6. Edit Ticket
7. Cancel Ticket
8. Cancel Booking

All these modules were tested within its own unit to ensure that they function as expected and outlined in the test results before being integrated into a full system.

5.1.1. Unit Testing for Login

Test Case ID: UIA_01

Test Priority (Low/Medium/High): High

Module Name: Account

Test Title: Verify login with username and password

Description: Test the system login page

Test Designed by: Tai Wai Jin

Test Designed Date: 3 September 2017

Test Executed by: Tai Wai Jin

Test Execution Date: 23 October 2017

Pre-conditions: User has valid username and password

Post-conditions: User is validated with database and successfully login to account.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to login page		Login page is displayed	Login page is displayed	Pass	
2	Provide username	Use username that has been registered				
3	Provide password	Use password that has been registered				
4	Click on Login button		User should be able to login	User is navigated to home page with successful login	Pass	

5.1.2. Unit Testing for Registration

Test Case ID: UIA_02	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): Medium	**Test Designed Date:** 4 September 2017
Module Name: Account	**Test Executed by:** Tai Wai Jin
Test Title: Register as new user	**Test Execution Date:** 24 October 2017
Description: Test the user registration page	

Pre-conditions: None

Post-conditions: New user is created in the database and auto login to account.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Registration page		Registration page is displayed	Registration page is displayed	Pass	
2	Input user information					
3	Click on Register button		User is logged in automatically and redirected.	User is logged in automatically and redirected	Pass	

5.1.3. Unit Testing for Change Password

Test Case ID: UIA_03	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): Low	**Test Designed Date:** 5 September 2017
Module Name: Account	**Test Executed by:** Tai Wai Jin
Test Title: Verify change password page	**Test Execution Date:** 25 October 2017
Description: Test the change password page	

Pre-conditions: User must be logged into the system.

Post-conditions: User record is updated in the database.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Profile page		Profile page is displayed	Profile page is displayed	Pass	
2	Select Change Password		Change Password page is displayed	Change Password page is displayed	Pass	
3	Provide old and new password					
4	Click Change Password button		Page refresh and alert message inform of change	Page refresh and alert message inform of change	Pass	

5.1.4. Unit Testing for Search Flights

Test Case ID: UIA_04	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): High	**Test Designed Date:** 6 September 2017
Module Name: Flights	**Test Executed by:** Tai Wai Jin
Test Title: Search for flights	**Test Execution Date:** 24 October 2017
Description: Test the search flights page	

Pre-conditions: Database populated with data (Airports, Flights, Planes, Tariffs)

Post-conditions: None

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Search Flight page		Search Flight page is displayed	Search Flight page is displayed	Pass	
2	Input search filter	Origin: Destination: Date:				
3	Click on Search button		Search Flight page is displayed with list of flights	Filtered flights are displayed as per conditions.	Pass	

5.1.5. Unit Testing for Book Flight

Test Case ID: UIA_05	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): High	**Test Designed Date:** 7 September 2017
Module Name: Booking	**Test Executed by:** Tai Wai Jin
Test Title: Book a flight	**Test Execution Date:** 27 October 2017
Description: Test the book flight page	

Pre-conditions: User must be logged into the system and existing flight data available in the system.

Post-conditions: Booking record is created in the database.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Click Book on an existing flight		Book Flight page is displayed	Book Flight page is displayed	Pass	
2	Input Passenger Details					
3	Click Book button		Book Flight page refreshes, and alert confirm booking has been made	Book Flight page refreshed and alert confirming booking has been made	Pass	

5.1.6. Unit Testing for Edit Ticket

Test Case ID: UIA_06

Test Priority (Low/Medium/High): Medium

Module Name: Ticket

Test Title: Edit a ticket in an existing booking

Description: Test the edit ticket page

Test Designed by: Tai Wai Jin

Test Designed Date: 7 September 2017

Test Executed by: Tai Wai Jin

Test Execution Date: 28 October 2017

Pre-conditions: User must be logged into the system and existing booking has been made.

Post-conditions: Ticket is updated in the database.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Edit Booking page		Edit Booking page is displayed	Edit Booking page is displayed	Pass	
2	Click Edit on a ticket record		Edit Ticket page is displayed	Edit Ticket page is displayed	Pass	
3	Change existing ticket record					
4	Click on Save button		Edit Booking page is displayed	Edit Booking page is displayed	Pass	

5.1.7. Unit Testing for Cancel Ticket

Test Case ID: UIA_07	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): Medium	**Test Designed Date:** 8 September 2017
Module Name: Ticket	**Test Executed by:** Tai Wai Jin
Test Title: Cancel a ticket in an existing booking	**Test Execution Date:** 28 October 2017
Description: Test the cancel ticket page	

Pre-conditions: User must be logged into the system and existing booking has been made.

Post-conditions: Ticket is removed from the database.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Edit Booking page		Edit Booking page is displayed	Edit Booking page is displayed	Pass	Select an existing booking
2	Click Cancel on a ticket record		Confirmation page is displayed	Confirmation page is displayed	Pass	
3	Click on Cancel		Edit Booking page is displayed	Edit Booking page is displayed	Pass	

5.1.8. Unit Testing for Cancel Booking

Test Case ID: UIA_08	**Test Designed by:** Tai Wai Jin
Test Priority (Low/Medium/High): Medium	**Test Designed Date:** 8 September 2017
Module Name: Booking	**Test Executed by:** Tai Wai Jin
Test Title: Cancel Existing Booking	**Test Execution Date:** 30 October 2017
Description: Test the cancel booking page	

Pre-conditions: User must be logged into the system and existing booking has been made.

Post-conditions: Booking is removed from the database.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status	Notes
1	Navigate to Bookings page		Booking page is displayed	Booking page is displayed	Pass	
2	Click Cancel on a booking record		Confirmation page is displayed	Confirmation page is displayed	Pass	
3	Click on Cancel		Booking page is displayed	Booking page is displayed	Pass	

5.2. Performance Testing

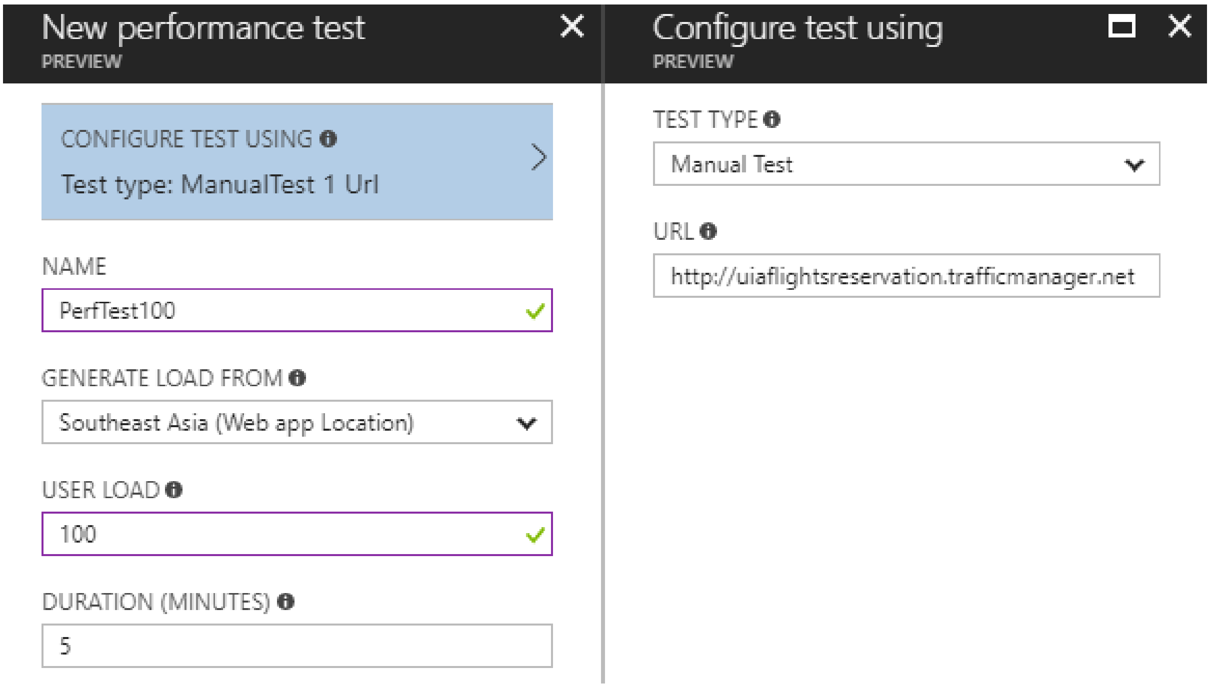


Figure 5.3.1 Initialising App Service Performance Test on Microsoft Azure Portal

The performance test was executed on two test targets with five iterations each to simulate concurrent user loads of 100, 250, 350, 450, 500 from the region of South East Asia. The first test target was the home page of the website, accessed through the URL http://uiaflightsreservation.trafficmanager.net and five tests were conducted for 5 minutes each. The results of the tests were tabulated in table 5.3.1.

User Load (Concurrent)	100	250	350	450	500
# Successful Requests	101968 (100%)	58559 (100%)	100582 (100%)	118731 (100%)	121905 (100%)
# Failed Requests	0	0	0	0	0
Average Response Time (secs)	0.42	2.77	2.34	2.55	2.87
Requests/sec	339.89	195.20	335.27	395.77	406.35

Table 5.3.1 Home Page Performance Test Results

The second test target was to simulate user requests searching for flights that involves communication between the App Service and SQL Database through the URL http://uiaflightsreservation.trafficmanager.net/Flights/Search?origin=Glasgow+International+Airport&dest=Dubai+International+Airport&date=2017-11-18. Five tests were conducted for 3 minutes each. The results of the tests were tabulated in table 5.3.2.

User Load (Concurrent)	**100**	**250**	**350**	**450**	**500**
# Successful Requests	5975 (99.5%)	8184 (97.84%)	8310 (99.11%)	7185 (96.72%)	4536 (89.79%)
# Failed Requests	30 (0.5%)	181 (2.16%)	75 (0.89%)	244 (3.28%)	516 (10.21%)
Average Response Time (secs)	3.55	5.27	7.9	11.09	17.31
Requests/sec	33.36	46.47	46.58	41.27	28.07

Table 5.3.2 Search Flight Page Performance Test Results

Based on the results tabulated in table 5.3.1 and 5.3.2, it can be observed that the Average Response Time increases as the Number of Concurrent User Load increases because the application has to deal with higher amount of workload at the same time. In addition, the number of failed requests in table 5.3.2 is higher because of the increase in computations performed by the function and the communication with the database. In order to improve the performance and availability of the system, upgrades can be performed to both the app service and database. The app service can implement autoscale function that scales up or out according to Percentage of CPU Usage rule automatically. Thus, the application would be able to deal with the spikes in requests more seamlessly and reduce or maintain average response time in the event of increase in concurrent user load. The SQL database can scale as well in terms of DTUs which increase the processing power and thus performance of the database.

6.0 Conclusion

This project involves the design and development of an Online Flight Booking System to manage the flight booking process of Ukraine International Airlines (UIA). After analysing the business case which outlines the scope and objectives of the project, a requirements specification was drawn up. These inputs drove the design process through design considerations which resulted in an Architectural Diagram and Modelling Diagrams that include the Use Case Diagram, Class Diagram, and Sequence Diagram.

Implementing the solution involves implementing the primary and secondary Azure SQL Databases in two different regions, publishing the application to Azure App Service in active and secondary regions, and creating a Traffic Manager profile with the two app services in different regions as end points. Subsequently, performance testing was conducted, and the monitoring mechanisms of the app service were analysed. This led to the decision to implement rule-based auto scale to improve the performance and availability of the application as and when required.

This project was a great lesson in terms of designing and developing applications on cloud and it would be a valuable skill and experience as the developer enter the software industry. The developer learnt the importance of a balanced architecture by considering required resources with the allocated budget to present the best solution. In addition, the implementation should be well thought of to consider performance, availability and scalability among other crucial factors. On top of that, the developer learnt to conduct performance tests on the cloud which allows for the simulation of high numbers of concurrent users accessing the application and would reflect the applications resilience to these stress loads. Furthermore, this project enriches not just implementing the solution on cloud, but also enhance the developer's programming skills.

All in all, cloud computing is the new trend that organisations are migrating to due to its speed, agility and global reach. Its pay-as-you-use model provides flexibility in terms of resources usage while not having to worry about the infrastructure. However, it may not be the best solution for all applications and an analysis should be conducted prior to adopting cloud based solutions.

The problems faced throughout execution of the project was time management and budget constraints. However, as these are real life problems bound to face in the industry, it serves as a good learning experience.

7.0 References

Jr, R. B., Kemnetz, J., Kodagoda, K. & Kamstra, D., 2017. *Overview of Monitoring in Microsoft Azure.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview
[Accessed 22 October 2017].

Lin, C. & Che, J., 2016. *Controlling Azure web app traffic with Azure Traffic Manager.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/app-service/web-sites-traffic-manager
[Accessed 24 October 2017].

Lin, C. & Tardif, B., 2016. *How to: Monitor Apps in Azure App Service.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/app-service/web-sites-monitor
[Accessed 20 October 2017].

Microsoft, n.d. *App Service pricing.* [Online]
Available at: https://azure.microsoft.com/en-gb/pricing/details/app-service/
[Accessed 6 October 2017].

Microsoft, n.d. *Azure SQL Database Documentation.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/sql-database/
[Accessed 4 November 2017].

Microsoft, n.d. *SQL Database pricing.* [Online]
Available at: https://azure.microsoft.com/en-gb/pricing/details/sql-database/
[Accessed 26 October 2017].

Wasson, M., 2016. *Basic web application.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/app-service-web-app/basic-web-app
[Accessed 20 September 2017].

8.0 Appendix

Stream Video Presentation : https://web.microsoftstream.com/video/b3850a2e-30c5-40cc-9ca9-2086db57a311

GitHub Repository : https://github.com/jtaiwaijin/DDAC-UIA-Online-Flights-Reservation