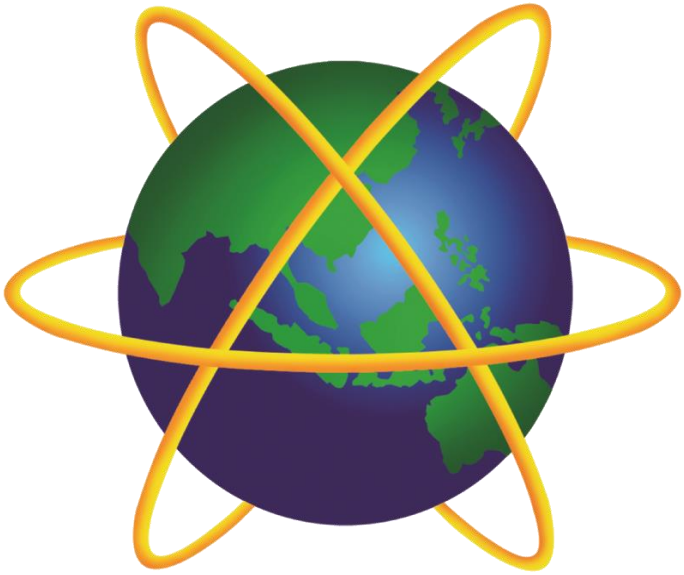


A · P · U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Module Code	**:**	CT071-3-3-DDAC - DESIGNING & DEVELOPING CLOUD APPLICATIONS
Intake Code	**:**	UC3F1702SE
Student Name	**:**	CHEE KAH SHING
TP No	**:**	TP033934
Lecturer Name	**:**	DR. KALAI ANAND A/L RATNAM
Hand in Date	**:**	20 NOVEMBER 2017
Assignment Title	**:**	UIA Online Flight Booking System

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to express my special thanks of appreciation to those who have been supporting and mentoring me throughout the progress of accomplish the report. First, I would like to thank Dr. Kalai for giving this golden opportunity in developing and deploying online web application with integration of Microsoft Azure services. My gratitude is also extended to his valuable guidance and expertise in the field of cloud computing. Without his supervision and encouragement, the process would not be going smooth when writing the report.

Moreover, I would like to thank all my friends who encouraged me a lot in finishing this report and the project within the limited time. I am thankful to them and appreciated it a lot. I would also like to thank my university, APU for the resources supporting all the times.

Lastly, a special thank should be given to my parents and family who have been always supporting me in all way they could.

1 INTRODUCTION

1.1 BACKGROUND INFORMATION

Online shoppers are notoriously fickle. If a website lags for even a few seconds, shoppers are just a couple of clicks away from many more options. Ukraine International Airlines (UIA) is the flagship carrier and largest airline in Ukraine. It operates domestic and international passenger flights and cargo services to Europe, the Middle East, the United States, and Asia.

The airline is eager to expand into new markets, but problems with its website prevented it from adequately serving customers beyond Ukraine. The site experienced severe denial-of-service (DOS) attacks, which hurt site performance and reliability, and it did not have the performance needed to host visitors from many parts of the world.

UIA has long used technology to reduce costs, innovate, and improve customer service. It has gone to a paperless cockpit and uses sophisticated software for analyzing fuel economy. The airline decided that it once again needed to innovate its way out of its web challenges. Dmitriy Prudnikov, Chief Information Officer at Ukraine International Airlines, realized that migrating the website out of UIA data centers into a public cloud could solve all these problems.

Ukraine International Airlines (UIA), is looking at designing and developing an Online Flight Booking System. UIA looked at both Microsoft Azure and Amazon Web Services and chose Azure. Azure was also very compatible with open source software, which didn't surprise Prudnikov.

1.2 OBJECTIVES

This project is aimed to design and develop a single tenant web solution that meets the following objectives as below to further expand the markets of UIA:

1. Able to create customer profile.
2. Able to manage entire booking process.
3. Able to view the information about the flight bookings made my customers.

1.3 SCOPES

The online flight booking system for UIA will be design and develop as a single tenant web application using Microsoft ASP.NET CORE and hosted on Microsoft Azure as an App Service (Web App). The hosted web application will consume Azure SQL Database to store all the related information to support the business activities of UIA like customer profile, flight details and flight bookings. The source code of the web application will be place in Github repository which provide convenient source control management services.

1.4 DELIVERABLES

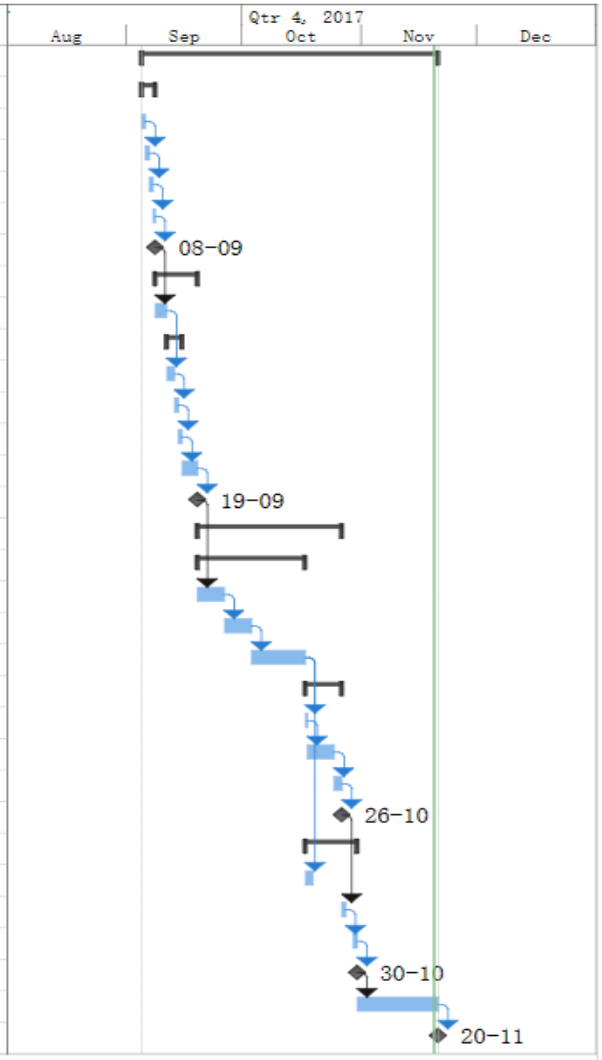
The online flight booking system developed in this project will include the following functionalities as below:

1. User allow to register account under the system.
2. User allow to sign in with registered account or social login account like Facebook and Twitter.
3. User allow to view their profile and edit their details after login to the system.
4. User allow to search flight by providing information about origin, destination and other related information.
5. User allow to book the flights that has selected from the flight searching result.
6. User allow to view all their flight booking details that have been made.

The overall functionalities of the flight booking system are further illustrated in the **Chapter 3 Modelling** with use case diagram, and sequence diagram.

2 PROJECT PLAN

ID	Task Name	Duration	Start	Finish	Predecessors
1	**UIA Online Flight Booking System**	**55 days**	**Tue 05-09-17**	**Mon 20-11-17**	
2	**Introduction**	**3.5 days**	**Tue 05-09-17**	**Fri 08-09-17**	
3	Project Background	1 day	Tue 05-09-17	Tue 05-09-17	
4	Objective	1 day	Wed 06-09-17	Wed 06-09-17	3
5	Scope	1 day	Thu 07-09-17	Thu 07-09-17	4
6	Deliverables	0.5 days	Fri 08-09-17	Fri 08-09-17	5
7	Introduction Completed	0 days	Fri 08-09-17	Fri 08-09-17	6
8	**Design**	**7 days**	**Fri 08-09-17**	**Tue 19-09-17**	
9	Design Consideration	1 day	Fri 08-09-17	Mon 11-09-17	7
10	Design Modelling	**4 days**	**Mon 11-09-17**	**Fri 15-09-17**	
11	Interface Design	2 days	Mon 11-09-17	Wed 13-09-17	9
12	Use Case Design	1 day	Wed 13-09-17	Thu 14-09-17	11
13	Database Design	1 day	Thu 14-09-17	Fri 15-09-17	12
14	Design Architectural Diagrams	2 days	Fri 15-09-17	Tue 19-09-17	13
15	Design Completed	0 days	Tue 19-09-17	Tue 19-09-17	14
16	**Implementation**	**27.5 day**	**Tue 19-09-17**	**Thu 26-10-17**	
17	Development	**20 days**	**Tue 19-09-17**	**Tue 17-10-17**	
18	Web Page	5 days	Tue 19-09-17	Tue 26-09-17	15
19	User Account Functions	5 days	Tue 26-09-17	Tue 03-10-17	18
20	Flight Funtions	10 days	Tue 03-10-17	Tue 17-10-17	19
21	Deployment	**7.5 days**	**Tue 17-10-17**	**Thu 26-10-17**	
22	Update Source Code to GitHub	0.5 days	Tue 17-10-17	Tue 17-10-17	20
23	Integrate Azure Services	5 days	Wed 18-10-17	Tue 24-10-17	22
24	Deploy to Azure	2 days	Wed 25-10-17	Thu 26-10-17	23
25	Implementation Completed	0 days	Thu 26-10-17	Thu 26-10-17	24
26	**Testing**	**9.5 days**	**Tue 17-10-17**	**Mon 30-10-17**	
27	Prepare Test Plan	2 days	Tue 17-10-17	Thu 19-10-17	20
28	Conduct Unit Testing	1 day	Fri 27-10-17	Fri 27-10-17	25
29	Conduct Performance Testing	1 day	Mon 30-10-17	Mon 30-10-17	28
30	Testing Completed	0 days	Mon 30-10-17	Mon 30-10-17	29
31	Project Final Documentation	15 days	Tue 31-10-17	Mon 20-11-17	30
32	System Completed	0 days	Mon 20-11-17	Mon 20-11-17	31



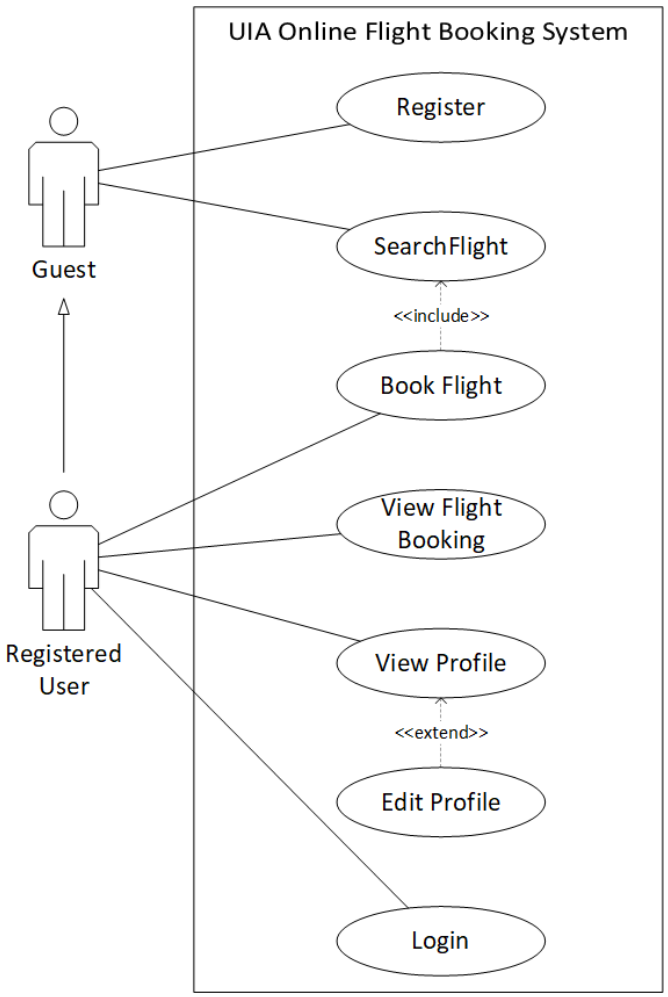
3 DESIGN

3.1 DESIGN CONSIDERATIONS

Before starting to design the online flight booking web application for UIA, several assumptions and considerations are made to fulfil the goal of UIA which is expanding their markets globally. The developed web application need to come with high performance and reliability to serve as many visitors as possible at instant of time. It should also be considered that for development and proof of concept, the developer has been given RM 150 per month of Azure credits and is required to work within the budget.

3.2 MODELLING

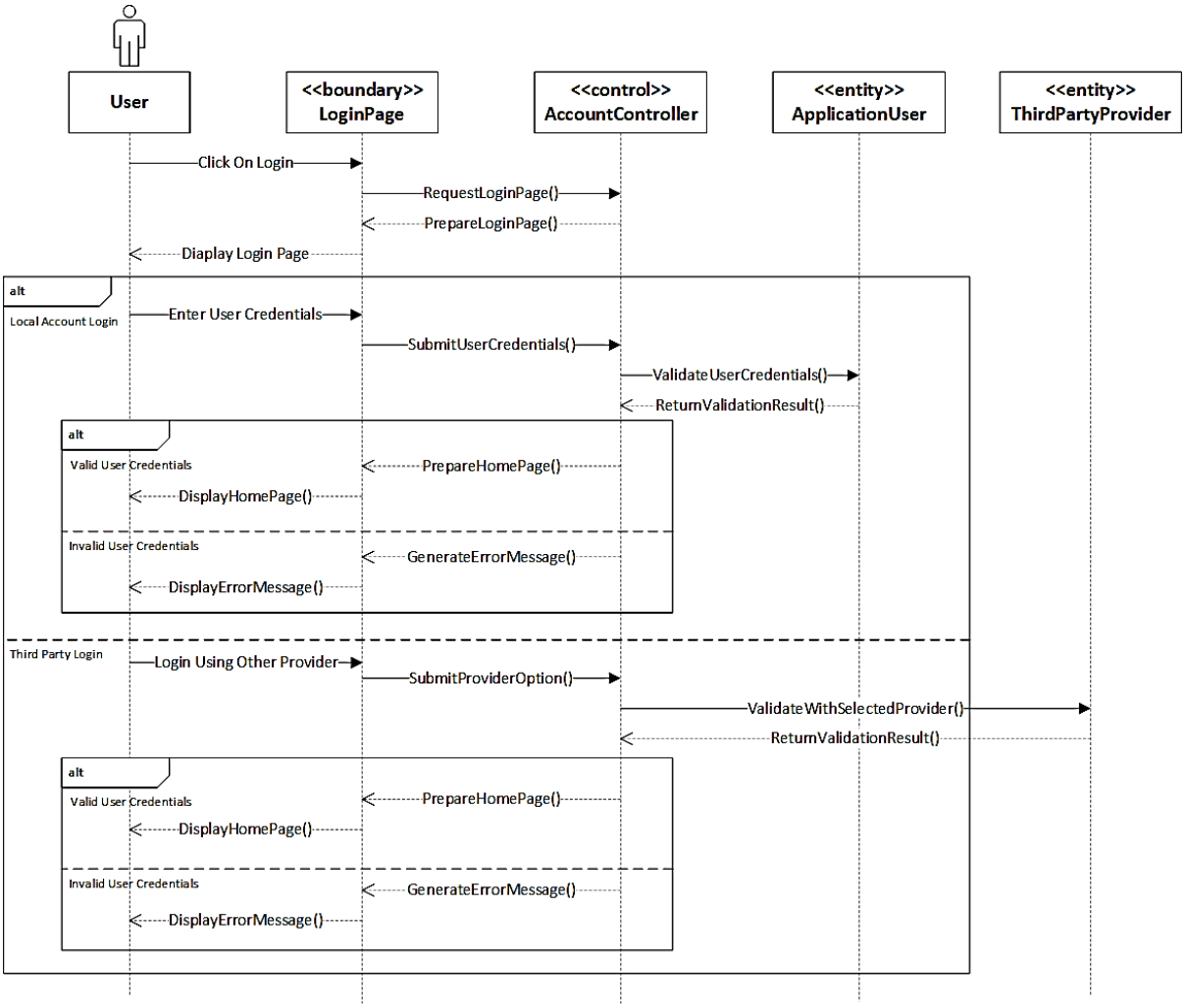
3.2.1 Use Case Diagram



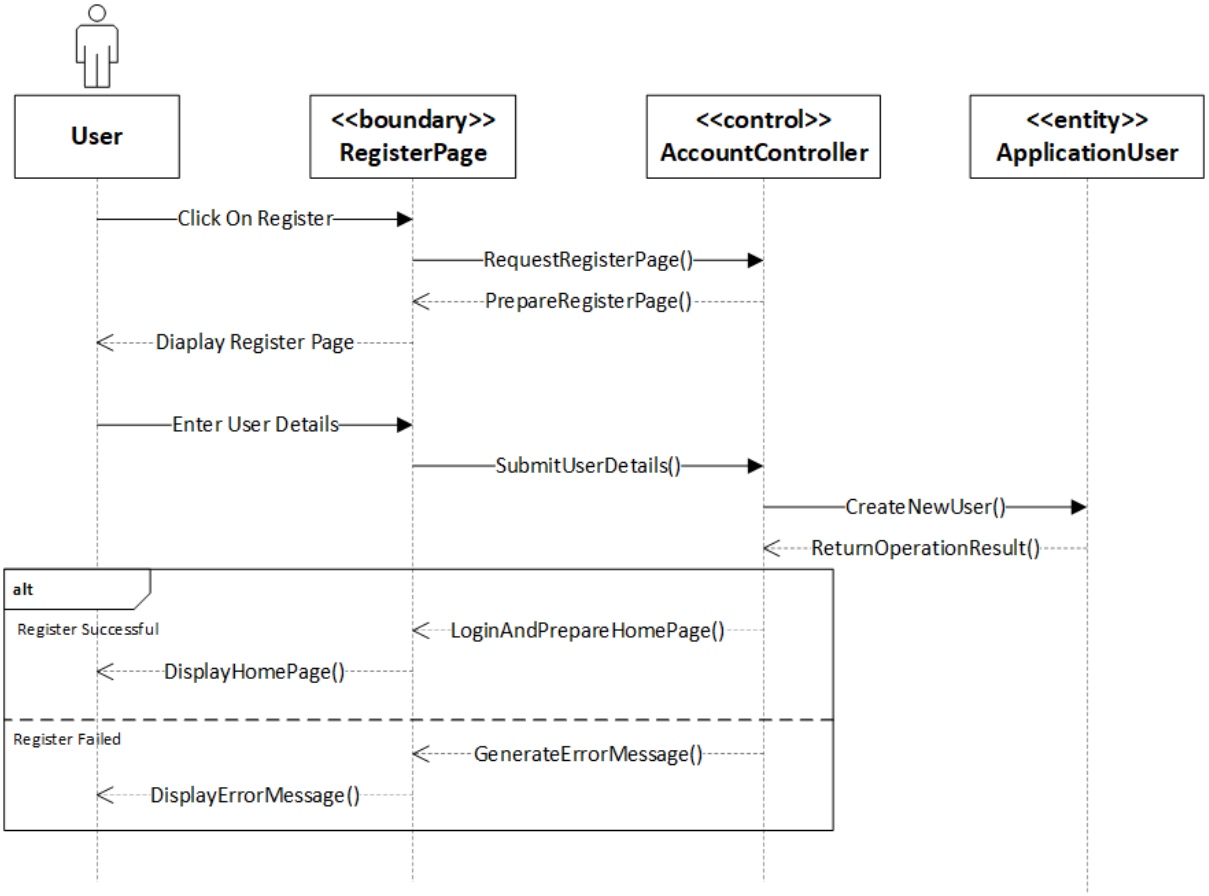
3.2.2 Sequence Diagram

The sequence diagrams illustrate below will show how the web application is supposed to perform each of the use cases drawn in the use case diagram above.

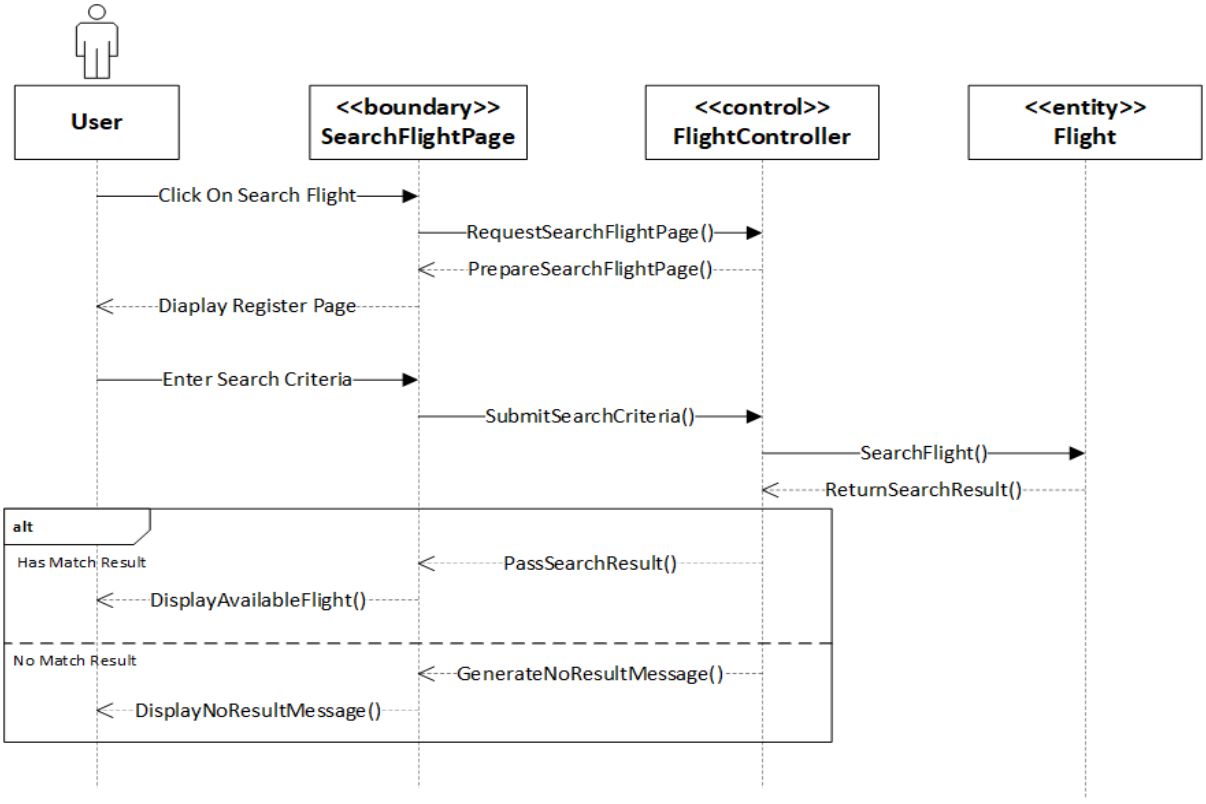
3.2.2.1 Login



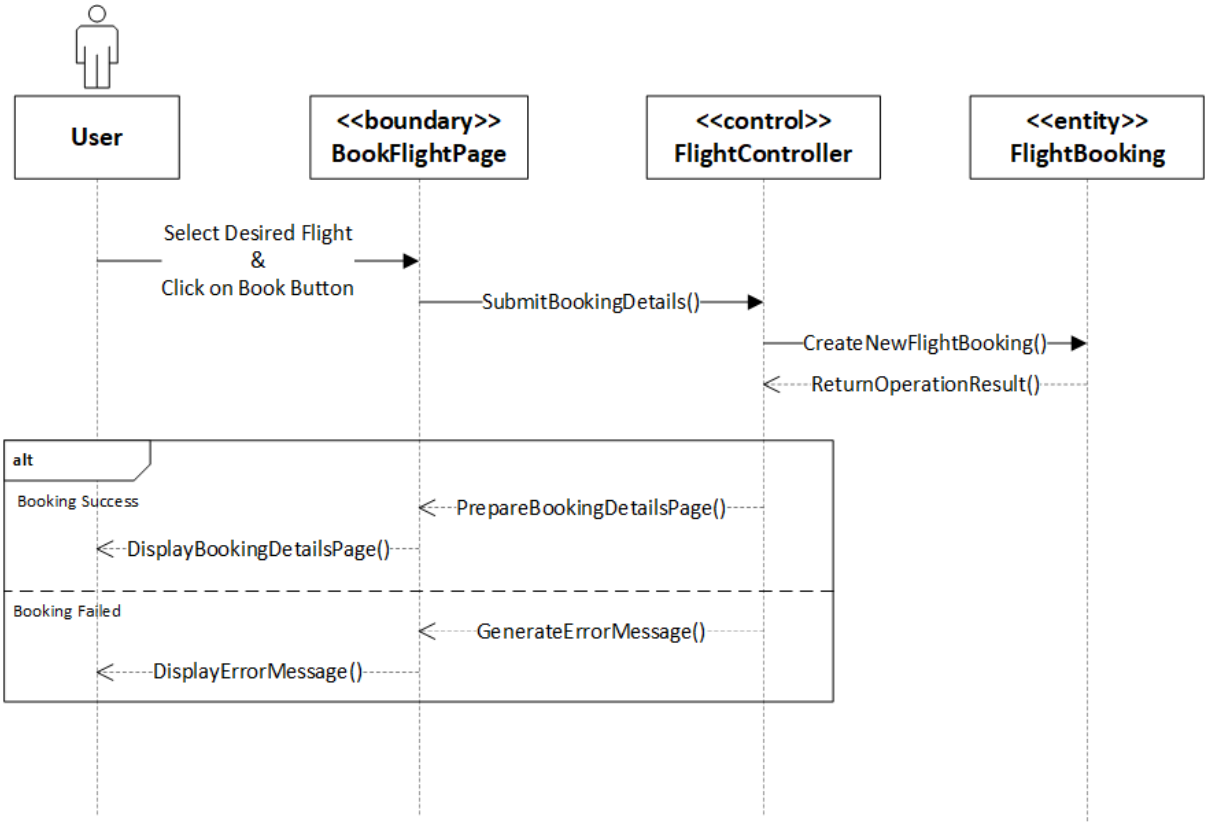
3.2.2.2 Register



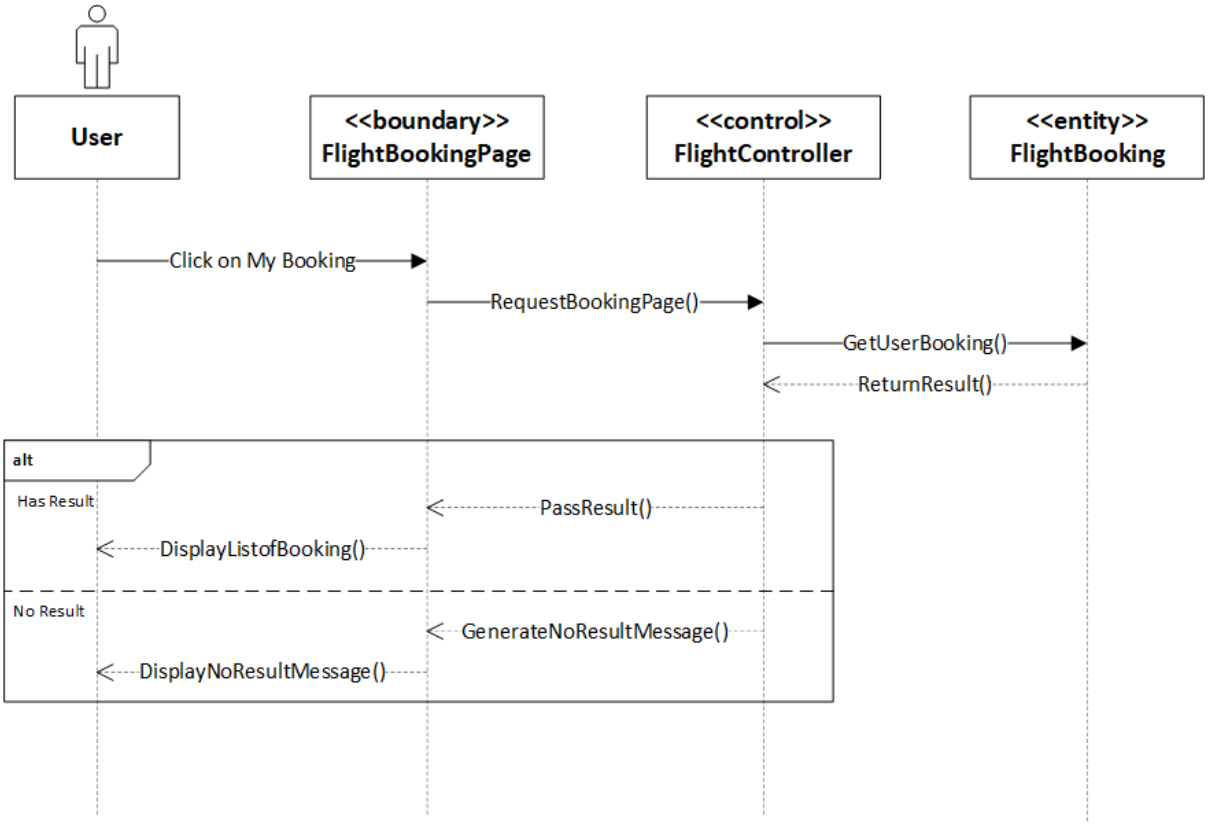
3.2.2.3 Search Fight



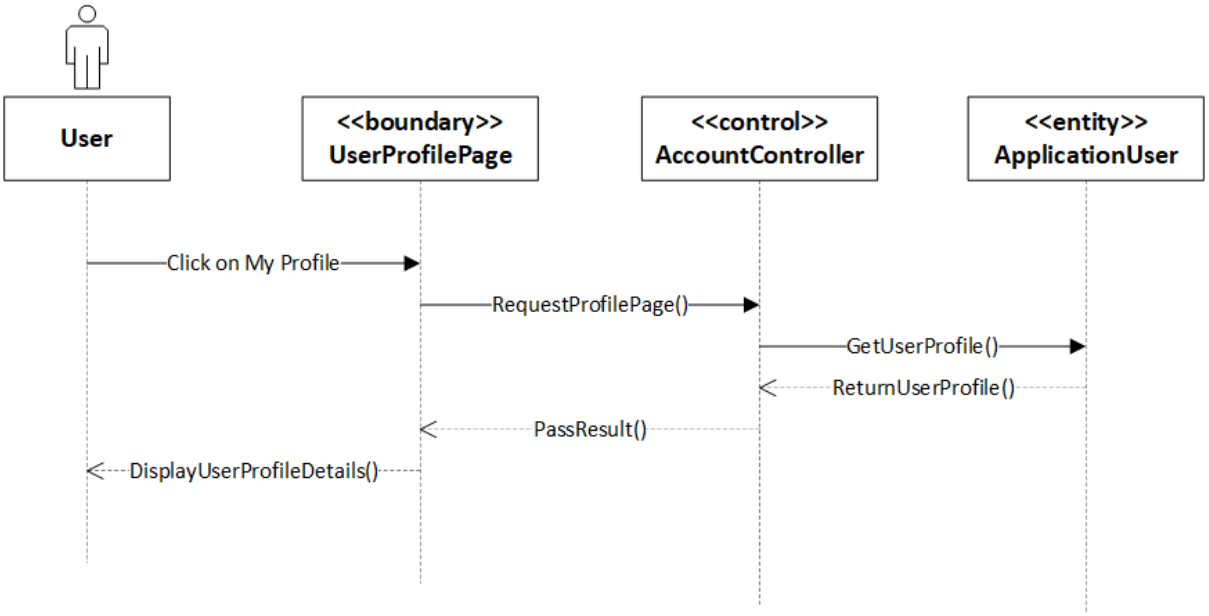
3.2.2.4 Book Flight



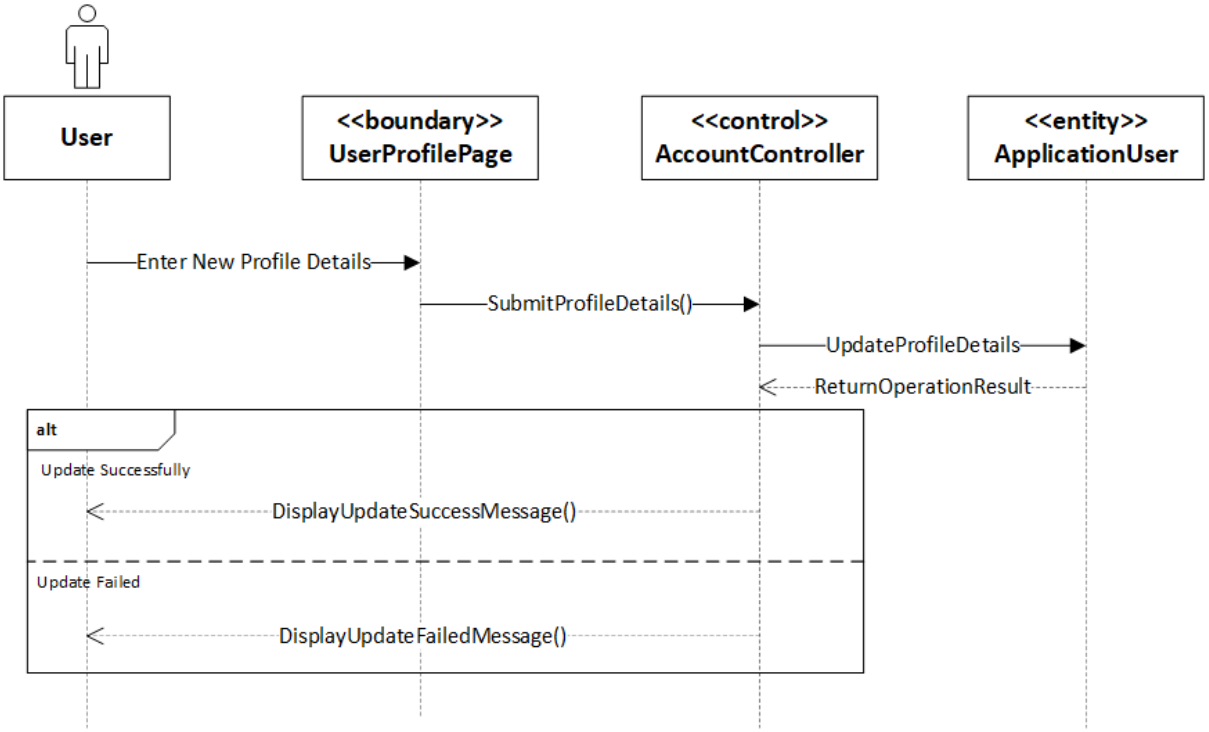
3.2.2.5 View Flight Booking



3.2.2.6 View Profile



3.2.2.7 Edit Profile



3.2.3 Entity Relationship Diagram

The database model used for the web application will be relational SQL database which has more benefits for storing business transaction records like flight booking. There are three main entities which are Application User, Flight and Flight Booking. Their relationship and attributes are illustrated as the entity relationship diagram below.

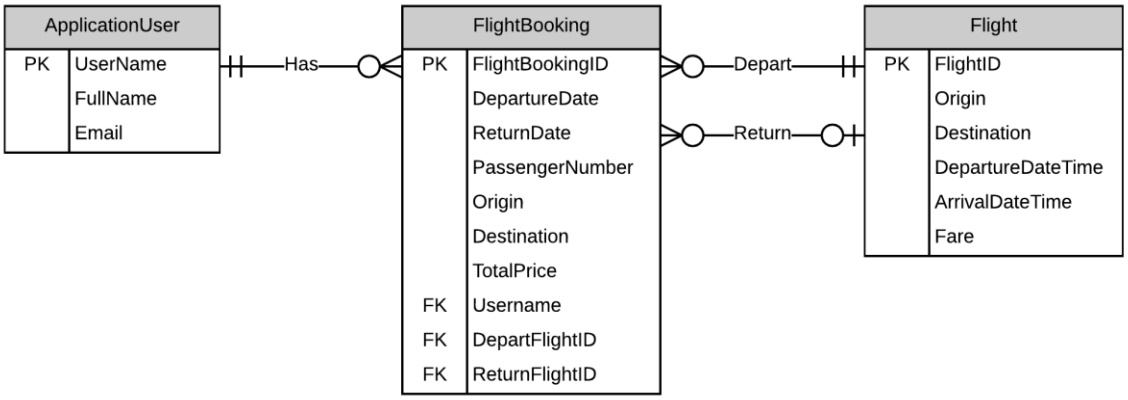


Diagram: Entity Relationship Diagram of UIA online flight booking system

3.2.4 Site Map

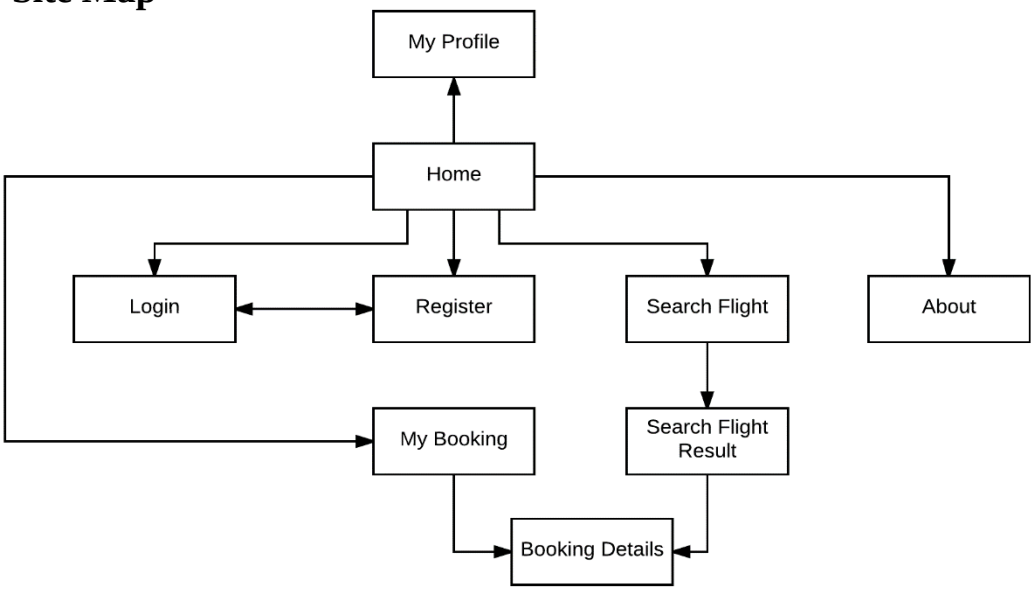


Diagram: Site Map of UIA online flight booking system

Diagram above show all the possible navigation link between each page in the web application. The web application will begin and display with Home page when user first entering the website.

3.3 CLOUD ARCHITECTURAL DIAGRAM

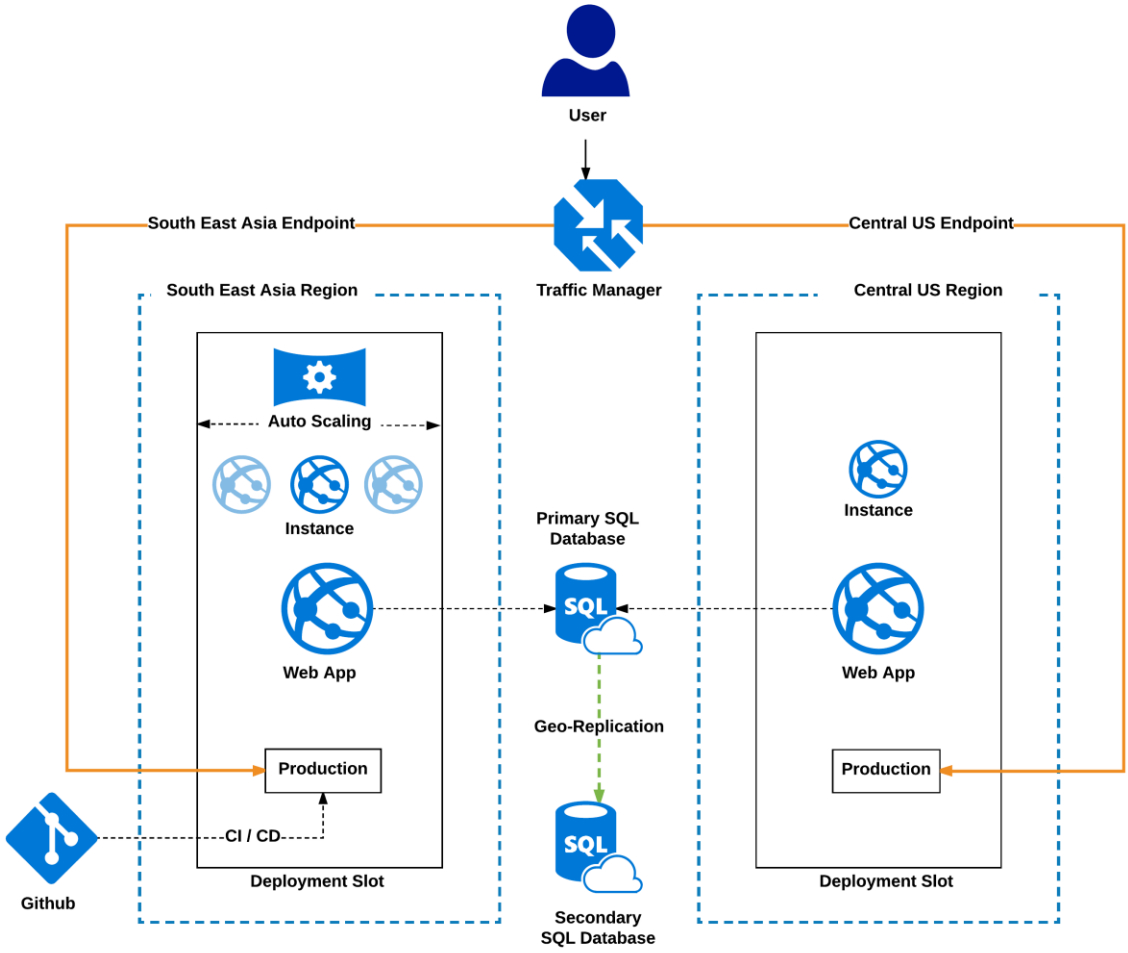


Diagram: Cloud architectural diagram for UIA online flight booking application

The cloud architectural diagram above is designed for deploying the UIA online flight booking web application to the Microsoft Azure cloud platform. As discussed in the design considerations, UIA is planning to expand their market wisely. So, the web application developed will be deploy in two different regions which are South East Asia (SEA) and Central US. This is to allow users from different regions be route to the nearest endpoint for faster response time and shorter loading time when browsing the websites. The primary web app is decided to place in SEA region while Central US region will be the secondary instance due to limited budget constraints and the developer believed that possible users of UIA mostly come from SEA region.

As the web application was written using Microsoft ASP.NET CORE, SQL server database is selected to have better compatibility and performance. Both regions that has selected to deploy the web application are sharing the same SQL database to ensure that both region has to same data to operate. A secondary database which is only readable is created via active geo replication from the primary database. This allow the web application to have an up-to-date backup that always ready to guarantee that the data can be recovered when needed. Besides, it also allows the secondary database to switch over with primary database if there is any problem or error for the primary database to make sure that the web application can continue to function properly.

Moreover, in the primary SEA region, auto scaling is enabled to scale in and out the web application according to the current situation. This not only guarantee the performance of the application, but also ensure the limited budget is fully utilised. During peak hours where more requests are coming in to the web application, it will automatically scale out to increase more application instance to handle the heavy load. While the requests or load on the application are drop, it will automatically scale in again to decrease the application instance to make sure the resources are not wasted. This can maximise the utilisation with the given budget.

Lastly, the cloud architecture for the web application has demonstrated the use of continuous delivery in the SEA region. This feature allows the web application to be build and deploy to the production automatically whenever there are any new changes in the pre-configured source code management repository. The UIA web application has utilised the GitHub repository to place and manage the source code of the application. As a result, the developer can easily push a new fix or modification to the production site by just updating the application source code in GitHub repository.

4 IMPLEMENTATION

4.1 APPLICATION DEVELOPMENT

The online flight booking application for UIA is written using C# with Microsoft ASP.NET Core which is a popular framework for developing modern and high-performance web application (Daniel Roth et al., 2017). The application is designed and developed by following the well-known structure which are Model, View and Controller (MVC) structure. Based on Microsoft (2017), MVC based web application allow high degree of control over the application behaviour but still relatively easy to manage complexity by dividing an application into the three different concerns accordingly which can provide better support for test-driven development (TDD).

One of the main function in the application is creating the customer profile which required for user authentication. This functionality is done through the ASP.NET Core Identity which allow to create user and add login functionality to the application easily as well as integrating with an external login provider such as Facebook, Google, Microsoft Account, Twitter or others (Pranav Rastogi et al., 2017). The code for adding the authentication service using ASP.NET Core Identity is shown as below which located in Startup.cs.

```
services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();

services.Configure<IdentityOptions>(options =>
{
    // Password settings
    options.Password.RequireDigit = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.Password.RequiredUniqueChars = 0;

    // User settings
    options.User.RequireUniqueEmail = true;
});

services.AddAuthentication().AddFacebook(facebookOptions =>
{
    facebookOptions.AppId = Configuration["Authentication:Facebook:AppId"];
    facebookOptions.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
}).AddTwitter(twitterOptions =>
{
    twitterOptions.ConsumerKey = Configuration["Authentication:Twitter:ConsumerKey"];
    twitterOptions.ConsumerSecret = Configuration["Authentication:Twitter:ConsumerSecret"];
});
```

Diagram: Adding the authentication service using ASP.NET Core Identity

After adding the services provided by ASP.NET Core Identity, the application can have sign in functionality ready easily by using the SignInManager instance to validate whether the login credentials of a user. The code for login is shown as below.

```
var result = await _signInManager.PasswordSignInAsync(userName, model.Password,
  model.RememberMe, lockoutOnFailure: false);
if (result.Succeeded)
{
    _logger.LogInformation("User logged in.");
    return RedirectToLocal(returnUrl);
}
```

Diagram: User login

Besides, the application developed is using Microsoft SQL Server database to store the related data as it has better compatibility and performance with ASP.NET Core. The database and table creation are well managed using Entity Framework Core which is an object-relational mapper (O/RM) that enables developers to work with a database using code first approach (Tom Dykstra and Rick Anderson, 2017). This has greatly eliminated the need for most of the data-access code that developers usually need to write. The developer only required to define the entities with all its attribute in a simple model class and add in to the database context as shown in the code below.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    0 references | shingz96, 16 hours ago | 1 author, 1 change | 0 exceptions
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    0 references | shingz96, 16 hours ago | 1 author, 1 change | 0 exceptions
    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        // Customize the ASP.NET Identity model and override the defaults if needed.
        // For example, you can rename the ASP.NET Identity table names and more.
        // Add your customizations after calling base.OnModelCreating(builder);
    }

    13 references | shingz96, 15 hours ago | 1 author, 1 change | 0 exceptions
    public DbSet<Flight> Flights { get; set; }
    2 references | shingz96, 15 hours ago | 1 author, 1 change | 0 exceptions
    public DbSet<FlightBooking> FlightBookings { get; set; }
}
```

Diagram: Add entities into Database Context

After that, by using the database context, it can read the entity in the database as shown in the code below which search for available flights.

```
private readonly ApplicationDbContext _context;
var departFlight = from f in _context.Flights
                   where f.Origin == model.Origin
                   && (f.Destination == model.Destination)
                   && (f.DepartureDateTime.Date == model.DepartureDate)
                   select f;
```

Diagram: Search available flights

With the use of ASP.NET Core MVC, it also allows to run some C# code directly in the cshtml (like html but with C# code support) that represent the View. For example, the code below shown using the SignInManager instance to validate whether the current visitor of the application has sign in or not to decide whether there is a need to render certain component for login user.

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

@if (SignInManager.IsSignedIn(User))
{
    <li><a asp-area="" asp-controller="Flights" asp-action="BookedFlights">My Booking</ 
      a></li>
}
```

Diagram: Run C# code in cshtml

A full demo of the application can be viewed in the attached cd. The source code of the application has published to GitHub repository and available to view at https://github.com/shingz96/UIAWebApp.

4.2 AZURE PUBLISHING

After the application has successfully developed in the local environment, it was then published to the cloud with Azure platform. As discussed before, the application will be deployed in SEA region and Central US region. So, two App service plan for the web app will be required and each located at the two regions accordingly.

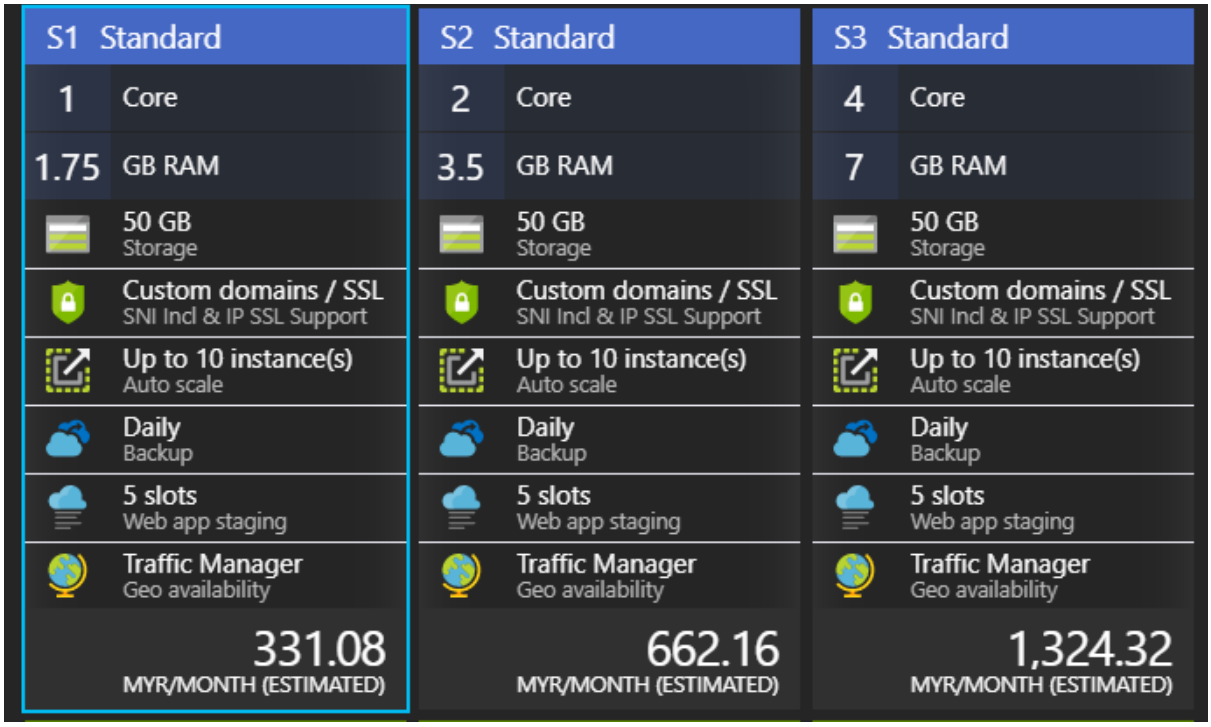


Diagram: Standard pricing tier for App Service (Microsoft Azure, 2017)

The selected pricing tier for App Service will be standard plan as it contains all the essential features like auto scale, daily backup and traffic manager to ensure the reliability and performance of the web application. So, with consideration of limited budget and needed features, a balance selection was made which is the S1 tier which is the minimum requirement to have all those important features. However, when the application is approved and ready for daily production use, the application may require upgrading the App Service plan to S3 or even higher to cope with more user especially for SEA region. This can be easily done using the Azure portal.

Besides, the SQL database provided by Azure should be added to the deployment as well. As the application using SQL Server database, Azure SQL Database is selected for the application as it not only same as normal Microsoft SQL server database but also with built-in intelligence that can quickly learns the deployed app's unique characteristics and dynamically adapts to maximize performance, reliability, and data protection (Microsoft Azure, 2017).

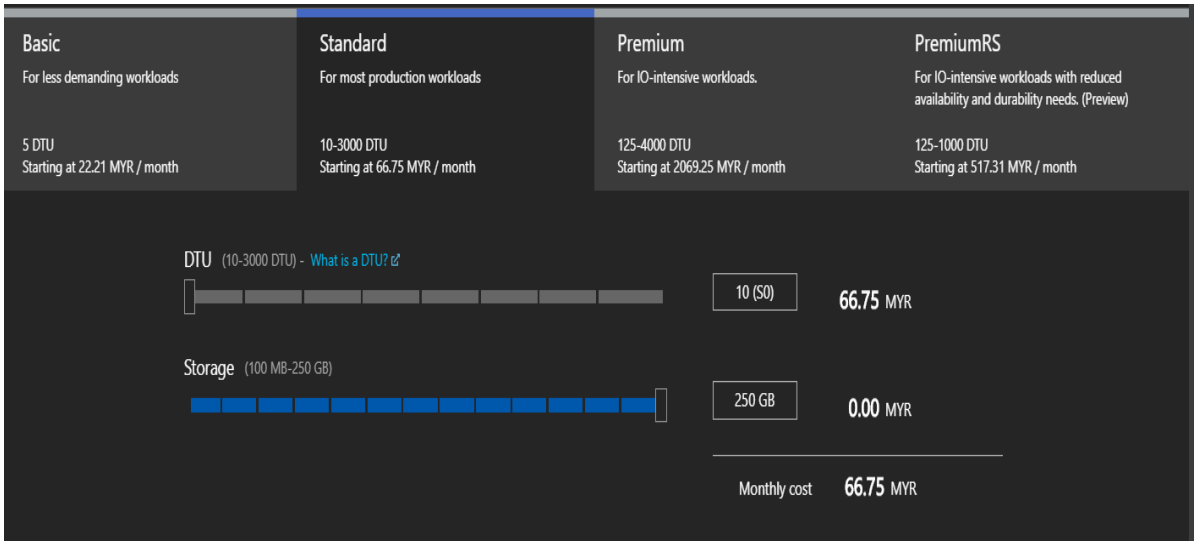


Diagram: Standard pricing tier for App Service (Microsoft Azure, 2017)

The selected pricing tier for the SQL database will be standard plan which support most of the production workloads with high storage up to 250 GB. S0 standard plan is the final decision for the developed application which can have up to 10 database transaction unit (DTU) which represent the performance level of the database. This option has better balanced between price and performance which is sufficient for running the database transaction processing task of the flight booking web application. The location selected for the SQL database will be the primary SEA region. So that, the primary site in SEA region can have better access time. Web app deployed in Central US region will have to share the same SQL database to ensure both region has operated on the same data set. Although this will cause users in Central US region suffer decreased performance but the requests coming from that region are usually lower as compare to SEA region. So, it should be acceptable within the budget constraints.

After created all the needed App Service and SQL database, the final step is to fully publish the developed application to the given App Service and integrated with the cloud SQL database of Azure. This required to configured the App Setting of the App Service with all the setting attributes such as database connection strings and external login provider integration credentials that previously located locally in app.setttings.json or secrets.json. The diagram below shows the App settings of the App Service for deploying the web application.

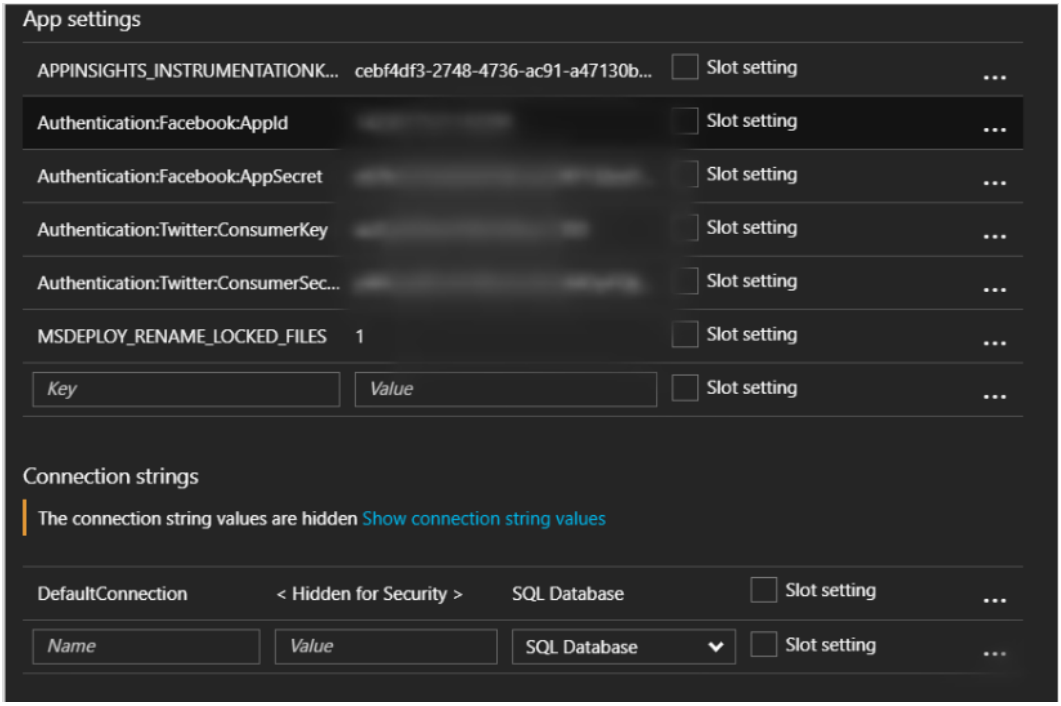
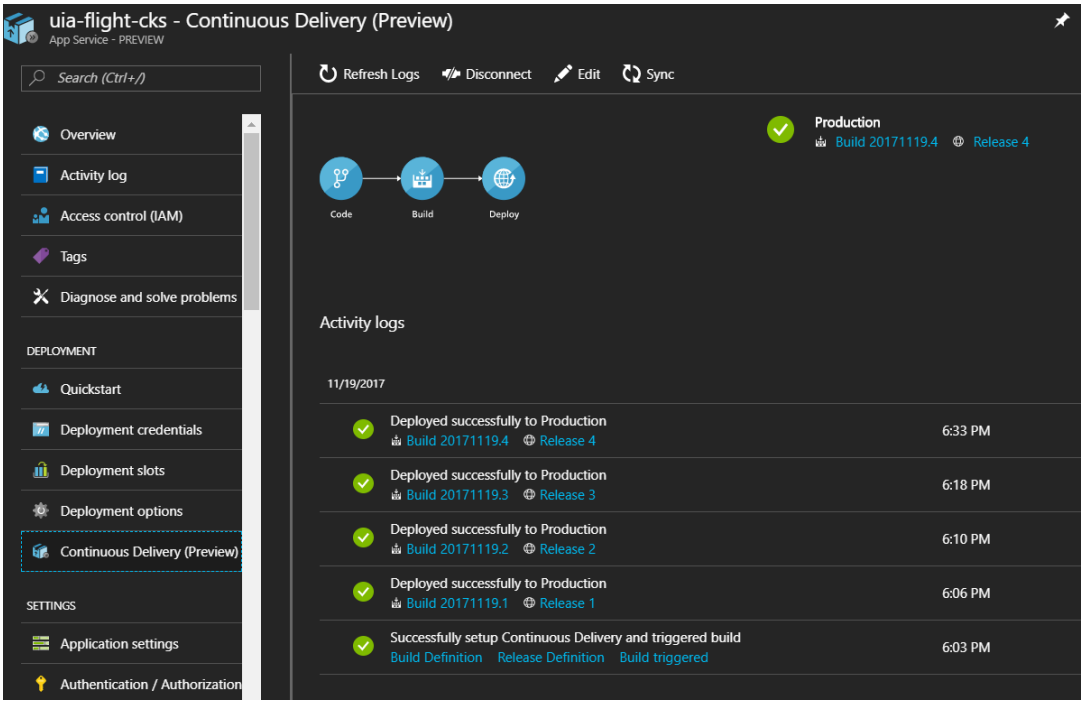


Diagram: App Settings of App Service



Furthermore, the web application that deployed in the SEA region is configured with continuous delivery which keep track on the GitHub repository that contain the source code of the application. Continuous Delivery (CD) is the process to build, test, configure and deploy from a build to a production environment with goal to keep production fresh by achieving the shortest path from the availability of new code in version control or new components in package management to deployment (Sam Guckenheimer, 2017). Whenever there are any new changes made and GitHub repository, the web application hosted in Azure will be automatically build with new source code and deploy to the production site. This is known as DevOps which unify the development and operation of software (Sam Guckenheimer, 2017).

4.3 APPLICATION SCALING

Azure App service (web app) allow to scale out easily either with manual or auto scale to increase or decrease the number of instances (Microsoft Azure, 2017). Manual scale allows to configure the web app to run with a defined number of instances while auto scale allows to increase or decrease the number of instance based on certain performance metric such as CPU usage, Memory Usage, HTTP Requests and so on (Microsoft Azure, 2017). This give the web application to dynamically allocating resources to match with current performance requirements.

For the each of the deployed flight booking web app in both SEA and Central US region, it has manual configured with 3 instances to ensure the reliability and performance of the application. The primary SEA region has setup with auto scale to automatically scale out or scale in based on the current load as SEA region is assumed to be the largest customer base. An additional instance for SEA region web app will be added (scale out) whenever the server average CPU utilization has increases more than 70% while an instance will be removed when it drops below 30%. The configurations are shown in the diagram below.

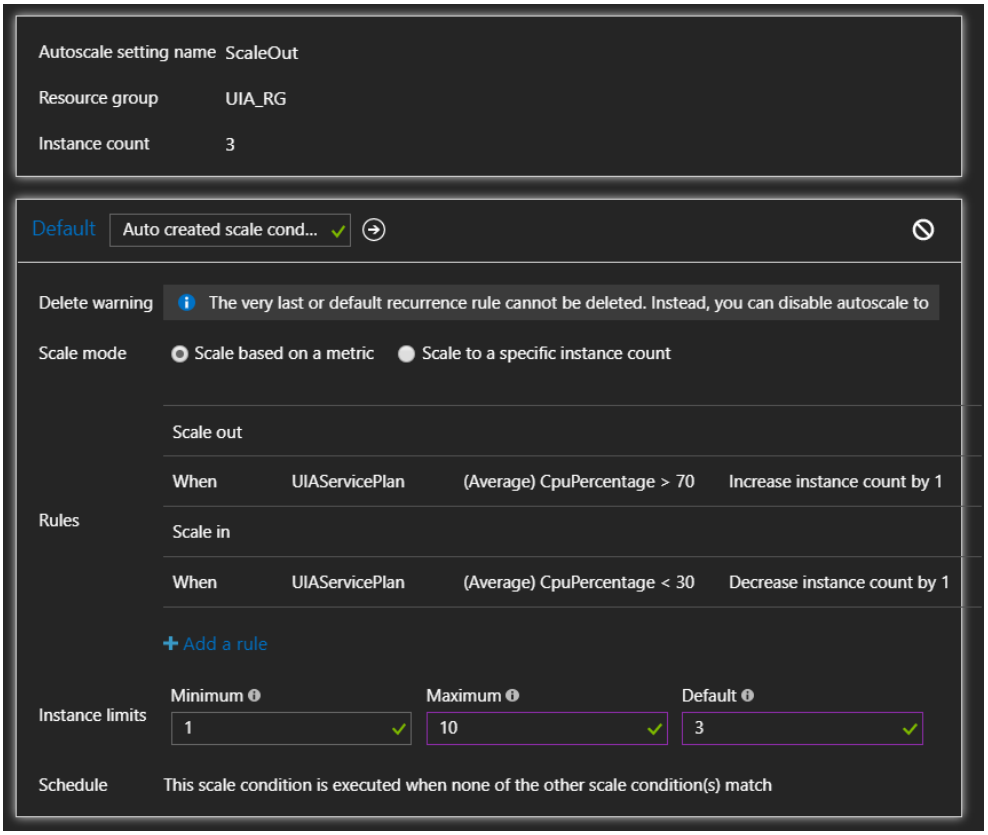


Diagram: Web application scaling for UIA flight booking system

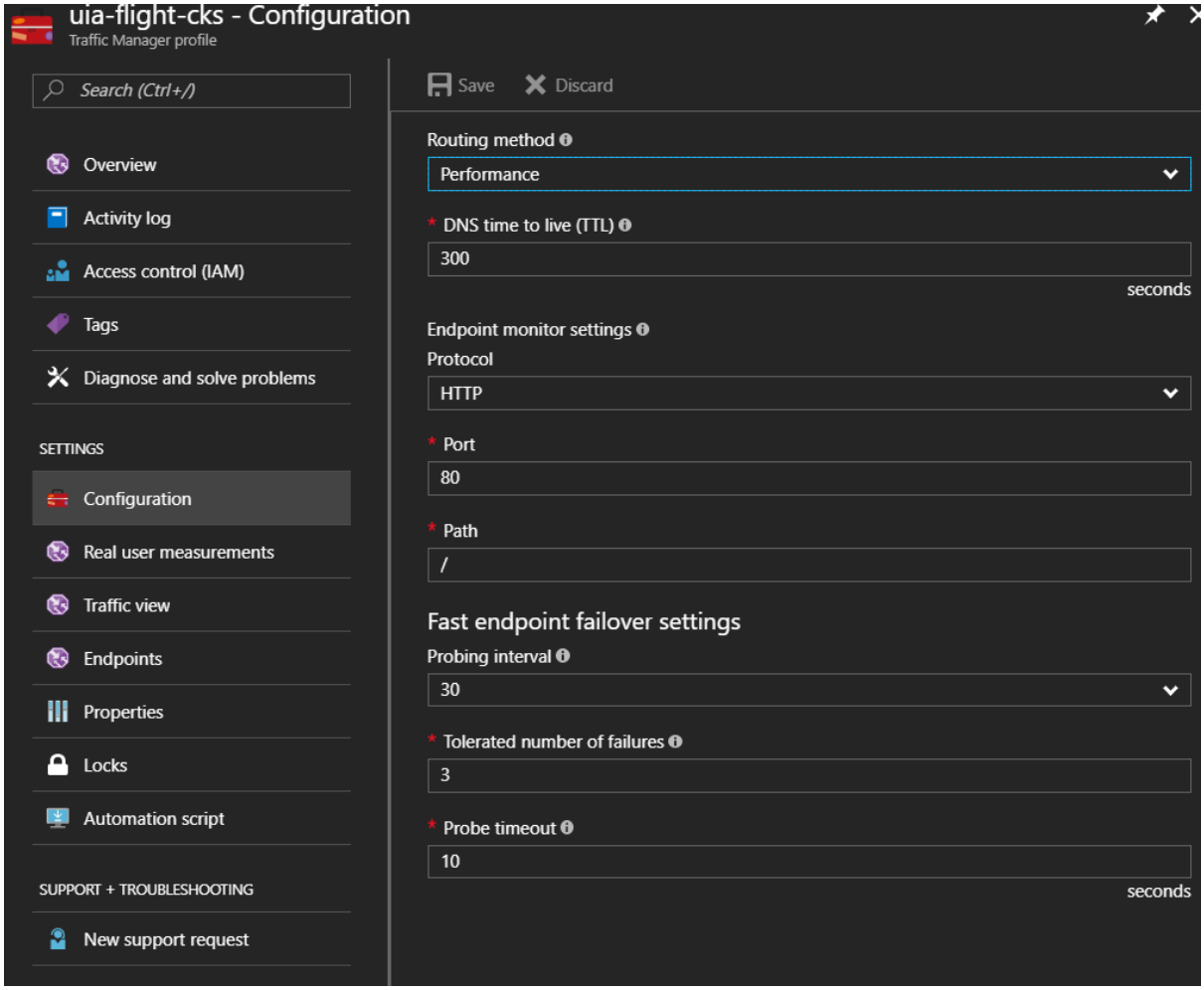


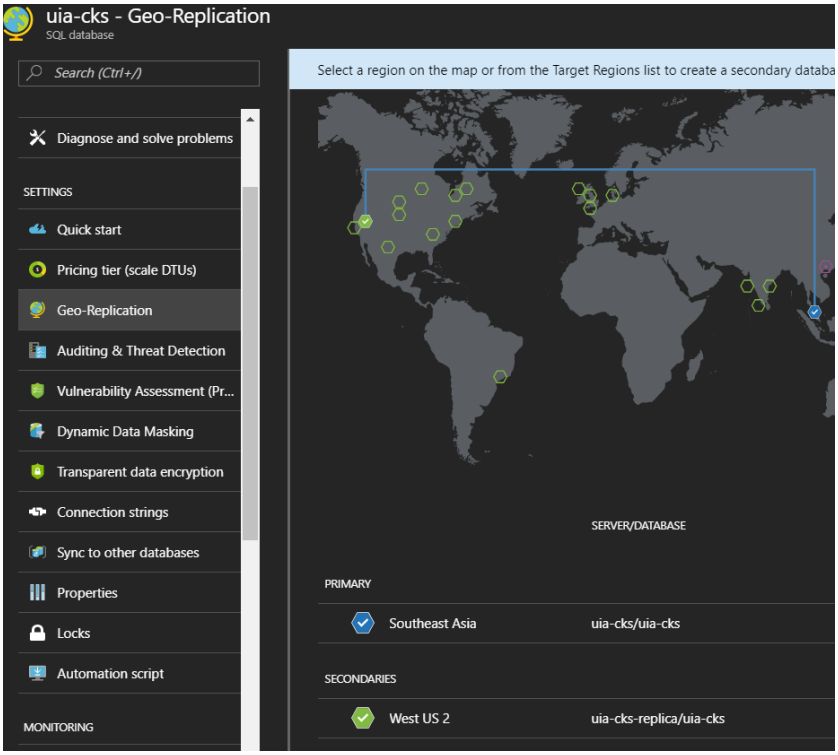
Diagram: Traffic Manager configuration

Besides, there will also be two web applications deployed in different geographical regions which are SEA region and Central US region. A traffic manager provided by azure is used and configured to direct the traffic in those regions based on the performance measure as shown in the diagram above. So that, user will be directed to their nearest location instead of requesting service from a server in another region. This also ensure that there is a backup application available if any one of the region's servers are down. For example, if the web app in SEA region is down, the traffic manager will redirect the traffic to the Central US server instead. Therefore, the user can continue to use the application while waiting another web application to be fix.

4.4 MANAGED DATABASE

Platform as a service (PaaS) is a complete solution which combine of both development and deployment environments in the cloud with resources readily available to deliver everything from simple cloud-based apps to complex, cloud-enabled enterprise applications (Microsoft Azure, 2017). All the development or deployment resources that needed can be purchase from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection (Microsoft Azure, 2017). This allows to avoid the expense and complexity of buying and managing software licenses, the underlying application infrastructure and middleware or the development tools and other resources. Once the resources are purchase from the cloud service provider, the developer just need to focus on managing the applications and services develop, while the rest will be well managed by the cloud service provider.

A SQL database native to the cloud, also known as a platform as a service (PaaS) database or a database as a service (DBaaS) that provide compatibility with most SQL Server feature (Microsoft Azure, 2017). This is what the Azure has provide with its cloud SQL database. It is built on standardized hardware and software that is owned, hosted, and maintained by Microsoft. The developed flight booking web application has took the benefits of Azure SQL database to develop directly on the service using built-in features and functionality. Not only that, it also come with pay-as-you-go options to easily scale up or out for greater power with no interruption.



Besides, the application using Azure SQL database also allow to perform active geo-replication which can create a secondary read only database to serve as a backup database for the primary database at a different location. The secondary database will constantly replicate and sync the data from primary database. This allow the application to switch over the backup database easily to ensure the business operation of UIA will not be interrupt even the primary database failed to function. In the flight booking application, a secondary database that located in West US 2 is replicated via the active geo-replication for the primary database that located in Southeast Asia region. This platform as a service allow developer to eliminate the complexity and efforts on managing database for active replication. All the developer need to do is just configured through the azure portal to have this useful feature enabled, so that the developer can focus more on application development.

5 TESTING

5.1 UNIT TESTING

Unit testing is carried out during the local development of the web application to ensure each of the individual unit of the application are work as intended. There are several unit to be test which are register page, login page, user profile page, search flight page, search flight result page, user flight booking page. Test plan for carry put the unit testing are prepared before and follow when performing the testing. All the testing result are written into test cases and shown in the following tables.

Register Page

Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-R-001	Register	FullName	1) Do not enter FullName 2) Press on Register button	Display error message "The Full Name field is required."	Error message of The Full Name field is required is displayed	Pass	Low
TC-R-002	Register	Username	1) Do not enter Username 2) Press on Register button	Display error message "The Username field is required"	Error message of The Username field is required is displayed	Pass	Low
TC-R-003	Register	Username	1) Enter duplicate Username 2) Press on Register button	Display error message "Username is taken"	Error message of username is taken is displayed	Pass	High

TC-R-004	Register	Password, ConfirmPassword	1) Do not enter Password and Confirm Password 2) Press on Register button	Display error message "The Password Field is Required."	Error message of password field is required is displayed	Pass	Low
TC-R-005	Register	Password, ConfirmPassword	1) Enter either Password or Confirm Password. 2) Press on Register button	Display error message "The password and confirmation password do not match."	Error message of the password and confirmation password do not match is displayed	Pass	High
TC-R-006	Register	Email	1) Do not enter Email 2) Press on Register button	Display error message "The Email Field is Required."	Error message of the email field is displayed	Pass	Low
TC-R-007	Register	Email	1) Enter invalid Email 2) Press on Register button	Display error message " Invalid email format."	Error message of the invalid email format is displayed	Pass	Low
TC-R-008	Register	FullName, Username, Email, Password, ConfirmPassword,	1) Enter valid details for all the fields. 2) Press on Register button	A new user profile is registered and automatically login and redirect to login screen.	A new user profile is registered successfully and login to the system and redirect to home page.	Pass	High

Login Page

Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-L-001	Login	Username, password	1) Enter invalid username. 2) Enter valid password. 3) Press on Login button	User should not able to login and display error message "Invalid login attempt."	Error message of Invalid login attempt.is displayed.	Pass	High
TC-L-002	Login	Username, password	1) Enter valid username. 2) Enter invalid password. 3) Press on Login button	User should not able to login and display error message "Invalid login attempt."	Error message of Invalid login attempt is displayed.	Pass	High
TC-L-003	Login	Username, password	1) Enter valid username. 2) Do not enter password. 3) Press on Login button	User should not able to login and display error message "The Password field is required"	Error message of the password field is required is displayed.	Pass	High
TC-L-004	Login	Username, password	1) Enter valid username. 2) Enter valid password. 3) Press on Login button	User should successfully login.	User has successfully login and redirect to Home page.	Pass	High

User Profile Page

Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-P-001	ViewProfile	None	1) Click on my name after login.	User profile details of the current user should be show	Display User Profile page and the user profile details of the current user is shown	Pass	High
TC-P-002	UpdateProfile	Email	1) Enter valid email. 2) Press on Save button	Message of "Profile updated successfully" should be show	Message of "Profile updated successfully" is displayed	Pass	High
TC-P-003	UpdateProfile	PhoneNumber	1)Enter valid PhoneNumber. 2) Press on Save button	Message of "Profile updated successfully" should be show	Message of "Profile updated successfully" is displayed	Pass	High
TC-P-004	EditPassword	CurrentPsw, NewPsw, ConfirmPsw	1) Enter valid CurrentPsw. 2) Enter valid NewPsw. 3) Enter valid ConfirmPsw. 4) Press on Change button	User password should be updated.	Message of "Password updated successfully" is displayed	Pass	High

Search Flight Page

Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-SF-001	SearchFlight	None	1) Click on search flight.	Display Search Flight page along with a search flight form	Search Flight page is display along with search flight form	Pass	High
TC-SF-002	SearchFlight	Origin, Destination, DepartureDate, PassengerNum, RoundTrip	1) Select Origin 2) Select Destination 3) No Select Departure Date 4) No check round trip 5) No enter Passenger Num 5)Press on Search button	Display appropriate error message to indicate Departure Date, Passenger Num fields are required.	Appropriate error message is displayed to indicate Departure Date, Passenger Num fields are required.	Pass	High
TC-SF-003	SearchFlight	Origin, Destination, DepartureDate, PassengerNum, RoundTrip, ReturnDate	1) Select Origin 2) Select Destination 3) Select Departure Date 4) Check round trip 5) No select Return Date 6) Enter Passenger Num 7)Press on Search button	Show Return Date column and Display appropriate error message to indicate Return Date field is required.	Return Date column is showed, and error message is displayed to indicate Return Date field is required.	Pass	High

TC-SF-004	SearchFlight	Origin, Destination, DepartureDate, PassengerNum, RoundTrip, ReturnDate	1) Select Origin 2) Select Destination 3) Select Departure Date 4) Check round trip 5) Select Return Date 6) Enter Passenger Num 7)Press on Search button	Redirect to Search Flight Result page and show list of available depart flight and return flight based on the search criteria	Search Flight Result page is displayed and showed with a list of available depart flight and return flight based on the search criteria	Pass	High
TC-SF-005	SearchFlight	Origin, Destination, DepartureDate, PassengerNum, RoundTrip, ReturnDate	1) Select Origin 2) Select Destination 3) Select Departure Date 4) No Check round trip 5) Enter Passenger Num 6)Press on Search button	Redirect to Search Flight Result page and show list of available depart flight based on the search criteria	Search Flight Result page is displayed and showed with a list of available depart flight based on the search criteria	Pass	High

Search Flight Result Page

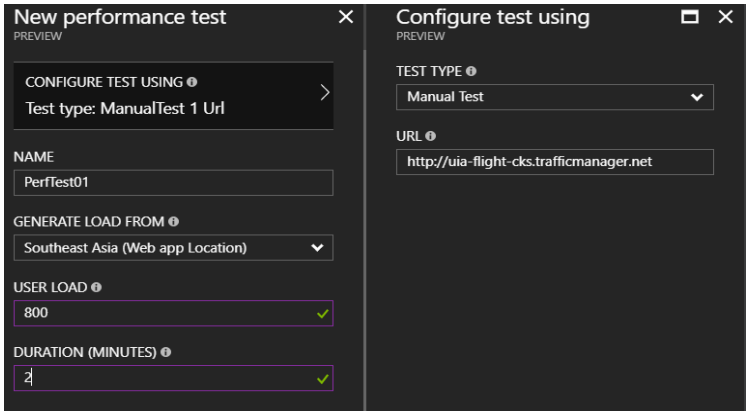
Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-SFR-001	BookFlight	SelectedFlight	1) No login successfully 2) No select any flight. 3) Click on Book Button	Display error message of “Please select your desired flight first”.	Error message of “Please select your desired flight first”. is displayed	Pass	High
TC-SFR-002	BookFlight	SelectedFlight	1) Has login successfully 2) No select any flight. 3) Click on Book Button	Display error message of “Please select your desired flight first”.	Error message of “Please select your desired flight first”. is displayed	Pass	High
TC-SFR-003	BookFlight	SelectedFlight	1) No Login 2) Has select flight. 2) Click on Book Button	Display error message of “Please login before booking a flight”.	Error message of “Please login before booking a flight”. is displayed	Pass	High
TC-SFR-004	BookFlight	SelectedFlight	1) Has Login 2) Has select flight. 2) Click on Book Button	Save the new flight booking by the user and redirect to the Booking details page show with the booked flights details	A new flight booking by the user is saved and redirect to the Booking details page show with the booked flights details	Pass	High

User Flight Booking Page

Test Case ID	Test Function	Test Parameters	Test Description	Expected Result	Actual Result	Status	Priority
TC-P-001	ViewBookedFlight	None	1) Has login successfully 2) Press on My Booking	Redirect to My Booking page and display a list of flight booking of the current user.	My Booking page is displayed with a list of flight booking of the current user.	Pass	High
TC-P-002	ViewBookingDetails	None	1) Click on the details of a flight booking record.	Redirect to Booking details page show with the details of the selected flight	Booking details page is displayed with the details of the selected flight.	Pass	High

5.2 PERFORMANCE TESTING

Azure Web App come with functionality called performance test to check the deployed app's performance before launching it or deploying updates to production. This can help to better assess whether the app is ready for releases and give more confident that the app can handle the traffic during peak hours. Therefore, additional consideration or implementation can be made to further improve the reliability and performance of the application when it goes live.



The deployed flight booking web application in the SEA region is tested with four test cases which will test the user loads on requesting the home page of the application with 200 concurrent users to 800 concurrent users within 2 minutes. Each of the test will increment by 200 concurrent users to simulate more user load. All the user loads will be come from South East Asia to simulate the real-world production scenario as UIA is assumed to have more customer base in SEA region. The final results of those tests are gathered include of average response time, requests per second, successful requests, and failed requests as shown in the table below.

User Load	200	400	600	800
Successful Request	50718 (100%)	44868 (100%)	2811 (99.86%)	9728(95.99%)
Failed Request	0 (0%)	0 (0%)	4 (0.14%)	406 (4.01%)
Request Per Second	422.65/s	373.9/s	23.46/s	84.45/s
Average Response Time (second)	0.93 s	2.39 s	15.51 s	20.98 s

Table: Performance Test Result

5.3 ANALYSIS

Based on the result of performance test above, it clearly shown that the web application located in SEA region with S1 standard plan is cable to handle concurrent user requests up until 400 with 0 failed request. However, the application starts to have failed requests after 600 concurrent users load. The average response time also has gradually increase with more concurrent users load from 200 until 800. Since the application begin to fail when having concurrent users more than 600, corresponding actions may require for scaling up or out the application to support more user loads at the same time especially during the peak hours.

Auto scaling is suggested to configured for the web application in the primary SEA region to automatically scale out based on current performance requirements. For example, the web application should be increase number of instance when having average CPU utilisation or memory usage more than 60%. Besides, scale up the web application is application as well. The pricing tier can be upgraded to S3 or even premium plan to support more user load with higher reliability and performance.

6 CONCLUSION

In conclusion, the development of a cloud based flight booking application for Ukraine International Airlines (UIA) was challenging and exciting. Throughout the project, it is not only focus on building the application but also deploying an application that fit well for the cloud. This is important as it allow the application to fully utilize and exploit all the features and possibilities that the cloud has offer. However, it is not easy and may require extra efforts from developer to always up-to-date with new and relevant skills in the context of modern informational technology industry. In the project, cloud design architectures and cloud resource provisioning as well as the considerations needed to design an appropriate cloud based application are well studied and explored such as automatic scaling. Undeniably, the used of cloud has become more popular where many industries started to migrate or involve themselves to the cloud. Therefore, with this hand on experience on the project, the developer has gained more knowledges and experiences in developing the cloud based application especially with the use of Azure platform by Microsoft which is one of the top cloud service provider in the current market.

REFERENCES

Daniel Roth et al. (2017). Introduction to ASP.NET Core. Microsoft. [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/ [Accessed on: 10 November 2017]

Microsoft. (2017). ASP.NET MVC Overview. Microsoft MSDN. [Online] Available at: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx [Accessed on: 10 November 2017]

Pranav Rastogi 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 [Accessed on: 10 November 2017]

Tom Dykstra and Rick Anderson. (2017). Getting started with ASP.NET Core MVC and Entity Framework Core using Visual Studio [Online] Available at: https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro [Accessed on: 10 November 2017]

Microsoft Azure. (2017). What are Azure SQL Database service tiers. Microsoft. [Online] Available at: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers [Accessed on: 10 November 2017]

Microsoft Azure. (2017). Azure App Service plan overview. [Online] Available at: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers [Accessed on: 10 November 2017]

Sam Guckenheimer. (2017). What is Continuous Delivery? Microsoft. [Online] Available at: https://www.visualstudio.com/learn/what-is-continuous-delivery/ [Accessed on: 10 November 2017]

Microsoft Azure. (2017). How to configure auto scaling for a Cloud Service in the portal [Online] Available at: https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-how-to-scale-portal [Accessed on: 10 November 2017]

Microsoft Azure. (2017). How to configure auto scaling for a Cloud Service in the portal [Online] Available at: https://docs.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling [Accessed on: 10 November 2017]

Microsoft Azure. (2017). How to configure auto scaling for a Cloud Service in the portal [Online] Available at: https://azure.microsoft.com/en-us/overview/what-is-paas/ [Accessed on: 10 November 2017]

Microsoft Azure. (2017). How to configure auto scaling for a Cloud Service in the portal [Online] Available at: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-paas-vs-sql-server-iaas [Accessed on: 10 November 2017]