‘WINGS ON’ USER STORY

**Technical Assessment**

|  |  |
| --- | --- |
|  | |
| Customer Name | Travix |
| Project/Sub-Project | Technical Assessment documentation |
| File Name | Wings Documentation.docx |
| Version | 0.1 |
| Compatibility Version | 1.0 |
| Author | Mohamed Salah |
| Publish Date | 15-JUN-2019 |
| Reviewer |  |
| Review Date |  |

Disclaimer

© 2019 All rights reserved. This document has been created for use by Travix International. These rights are not transferable to any third party without the prior written consent of the author.

Table of contents

[Record of Changes 5](#_Toc11687399)

[Approvals 6](#_Toc11687400)

[Distribution List 6](#_Toc11687401)

[Documents References 7](#_Toc11687402)

[1 Sum of Multiple 8](#_Toc11687403)

[1.1 Problem statement 8](#_Toc11687404)

[1.2 Assumption 8](#_Toc11687405)

[1.3 Solution Approaches 8](#_Toc11687406)

[1.3.1 Approach A 8](#_Toc11687407)

[1.3.2 Approach B 9](#_Toc11687408)

[1.3.3 Approach C 10](#_Toc11687409)

[1.4 Design Pattern 11](#_Toc11687410)

[1.4.1 Template Method Pattern 11](#_Toc11687411)

[1.4.2 Dependency Injection 12](#_Toc11687412)

[1.5 Statistical Running 12](#_Toc11687413)

[2 Sequence Analysis 13](#_Toc11687414)

[2.1 Problem Statement 13](#_Toc11687415)

[2.2 Assumption 13](#_Toc11687416)

[2.3 Solution Approaches 13](#_Toc11687417)

[2.3.1 Approach A 13](#_Toc11687418)

[2.3.1.1 Pseudocode and Complexity 13](#_Toc11687419)

[2.3.2 Approach B 13](#_Toc11687420)

[2.3.2.1 Pseudocode and Complexity 14](#_Toc11687421)

[2.4 Design patterns 15](#_Toc11687422)

[2.4.1 Factory Method Pattern 15](#_Toc11687423)

[2.4.2 Dependency Injection 16](#_Toc11687424)

[2.5 Statistical Running 16](#_Toc11687425)

[3 Testing Solution 18](#_Toc11687426)

# Record of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Change Reference |
| 0.1 | 19-Jun-2019 | Mohamed Salah | Initial version |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Approvals

This document has been approved by the following people

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Function | Date of Approval | Signature |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Distribution List

This document has been distributed to:

|  |  |
| --- | --- |
| Name | Function |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Documents References

|  |  |  |
| --- | --- | --- |
| Document Name | Document Reference | Version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 

# Purpose

The purpose of this document is presenting the **design** and **development** approaches to achieve the requirement in the technical assignment.

I did my best to cover most common topic in design Restful API.

# Problem statement

As a Front-End Developer, I need a REST-ful Web API for my ticketing website, so that I can access and manage information related to the passengers.

Website need the following:

1. Endpoint that returns a person by Id.

2. Endpoint that returns all passengers on the flight by number for example ‘PZ696’.

3. Endpoint that updates a person’s email address.

4. Endpoint that creates a booking of an existing flight for a new passenger.

5. Endpoint that lists all the male passengers.

# Endpoint Table

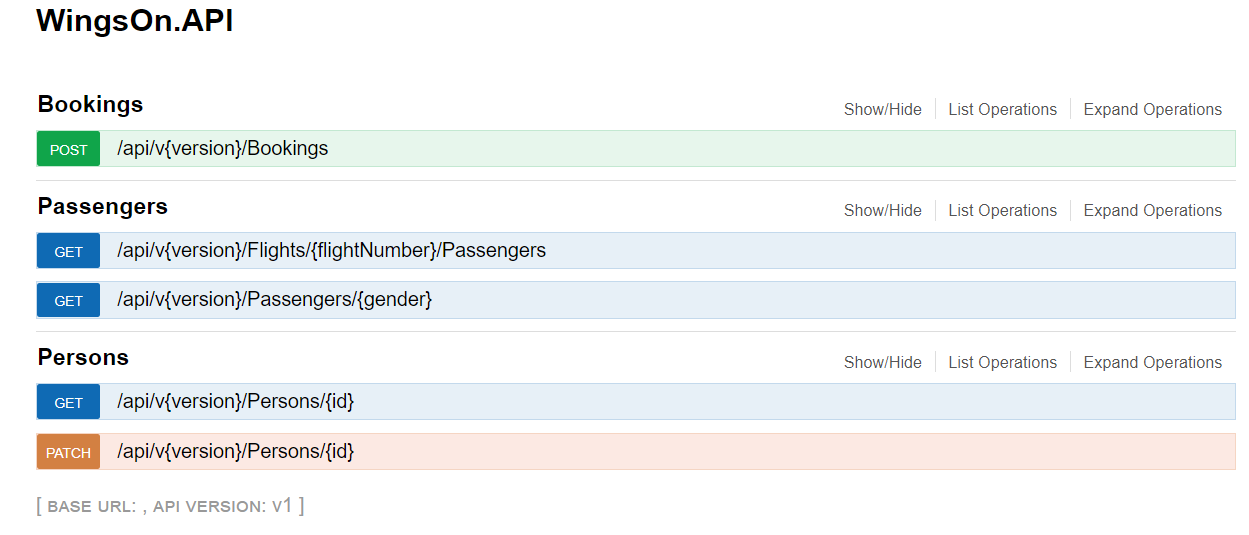
The following is the Endpoint Table

|  |  |  |
| --- | --- | --- |
|  | Endpoint | Usage |
| POST | /api/v1/Bookings | Create new Booking with new Passenger |
| GET | /api/v1/Flights/{flightNumber}/Passengers | Get All passenger in specific gender |
| GET | [/api/v1/Passengers/{gender](http://localhost:63312/swagger/ui/index#!/Passengers/Passengers_Get)} | Get All Passenger with specific gender |
| GET | [/api/v1/Persons](http://localhost:63312/swagger/ui/index#!/Persons/Persons_Get) | Get a Person in specific Id |
| PATCH | [/api/v1/Persons](http://localhost:63312/swagger/ui/index#!/Persons/Persons_Get) | Update Person mail |

Also, you can find this using Swagger url

http://<hostname>:<port>/swagger/ui/index

<http://localhost:63312/swagger/ui/index>



# Assumption

The following assumption will apply for the different approaches

* This approach is focus of build restful API but not deeply focus of business validation for example available capacity of the flight and the departure date of the flight
* No authentication required for that test as per email sent from travix

# Covered Topics

The goal of this project is mainly to focus on some topics the following are the topics that covered in this project

1. *SOLID* Principals
2. Separation of concern
3. Simplicity, readability and Clean code

# Best Practice followed for Building REST API

I follow the next point in building the APIs

### Use nouns but no verbs

For an easy understanding use this structure for every resource:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | GET read | POST create | PUT update | PATCH  Update Partially |
| /Bookings |  | Create a new Booking |  |  |
| /Flights/{flightNumber}/Passengers | Returns Passengers in specific flight number |  |  |  |
| [/Flights/Passengers/{gender](http://localhost:63312/swagger/ui/index#!/Passengers/Passengers_Get)} | Returns Passengers in specific gender (male or female) |  |  |  |
| [/Persons](http://localhost:63312/swagger/ui/index#!/Persons/Persons_Get)/{Id} | Returns a specific person |  |  |  |
| [/Persons](http://localhost:63312/swagger/ui/index#!/Persons/Persons_Get)/{Id} |  |  |  | Update person Email |

And avoid to use verbs for example:

/getAllPassenger/

/CreateNewBooking

### GET method and query parameters should not alter the state

Use **PUT**, **POST** and **DELETE** methods instead of the **GET** method to alter the state.

Do not use **GET** for state changes:

GET [/Persons/77/UpdateEmail/test@test.com](mailto:/Persons/77/UpdateEmail/test@test.com)

### Use plural nouns

Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.

/Persons instead of /Person

/Bookings instead of /Booking

### Use sub-resources for relations

If a resource is related to another resource use sub resources.

GET /Flights/BB124/Passengers Returns a list of Passengers for flight BB124

### Provide sorting, field selection and paging for collections

Passengers/0?page=1&pageCount=5&sidx=Id&sord=asc

### Version your API

Make the API Version mandatory and do not release an unversioned API. Use a simple ordinal number and avoid dot notation such as 1.

### Handle Errors with HTTP status codes

It is hard to work with an API that ignores error handling. Pure returning of a HTTP 500 with a stack trace is not very helpful.

### Use HTTP status codes

The HTTP standard provides over 70 status codes to describe the return values. We don’t need them all, but there should be used at least amount of 10.

200 – OK – Everything is working

400 – Bad Request – The request was invalid or cannot be served. The exact

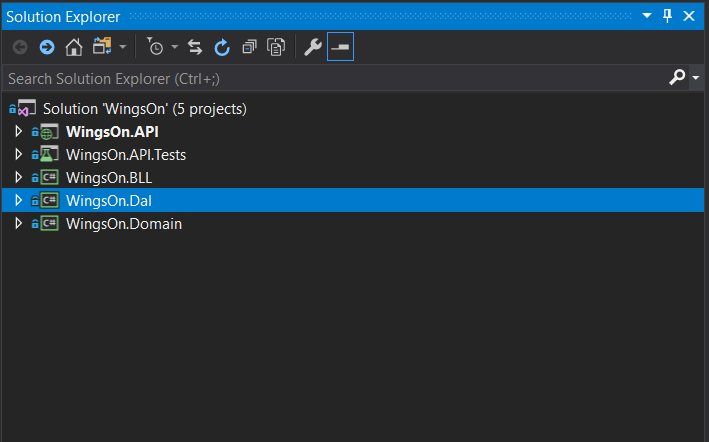
# Solution Structure

To prove the solution for both problem and according to what is minimum required.

New Solution “WingsOn” contains the following projects

Using Visual studio 17 and .Net Framework 4.7

|  |  |
| --- | --- |
| Project Name | Usage |
| WingsOn.API | API Tier |
| WingsOn.API.Tests | Test project for the API tier |
| WingsOn.BLL | Business Logic tier |
| WingsOn.Dal | Data Access tier |
| WingsOn.Domain | Domain Tier |



**Note**

Adding Business Logic Layer (BLL) Due to ***concept separation of Concern***

This gives capability to

1. separate the API layer from being tightly coupled with data base layer DAL
2. Also contains the algorithms which operate on the data in response to user input
3. BLL can contains any business rules or integration with external services

# Used Packages

The following tables show and used packages

|  |  |
| --- | --- |
| Package Name | Usage |
| AutoMapper | Automapper is an object to the object mapper. Automapper can map properties of one object of one type, to the properties of another object. The automapper is widely used in the cases where DTO (Data transfer object) are used. By this, object properties can be assigned very easily from View object to DTO object and DTO object to Domain model |
| log4net | use logging as a way to store information about performed operations. Such data is very useful when you have to diagnose an issue |
| Swashbuckle | Swagger basically is a framework for describing, consuming, and visualizing RESTful APIs. The nice thing about Swagger that it is really keeps the documentation system, the client, and the server code in sync always, in other words the documentation of methods, parameters, and models are tightly integrated into the server code. |
| Unity.WebApi | Unity.WebAPI is a library that allows the simple integration of Microsoft's Unity **IoC** container with ASP.NET Web API. |
| System.Linq.Dynamic | helper library that allows you to express LINQ queries using extension methods that take string arguments instead of type-safe language operators |

# Spot on Code

There are 3 parts on the code need to put spot on it.

### SOLID

As mention in covered topic solution focus in applying the ***SOLID*** principal specially the second principal ***O*** (open for extension close for modification)

So, in tier “WingsOn.Dal” when I need to add more features for RepositoryBase

I add extension for it RepositoryBaseExtension and avoid modify the RepositoryBase

Then I got the following features without touch RepositoryBase

/// adding where

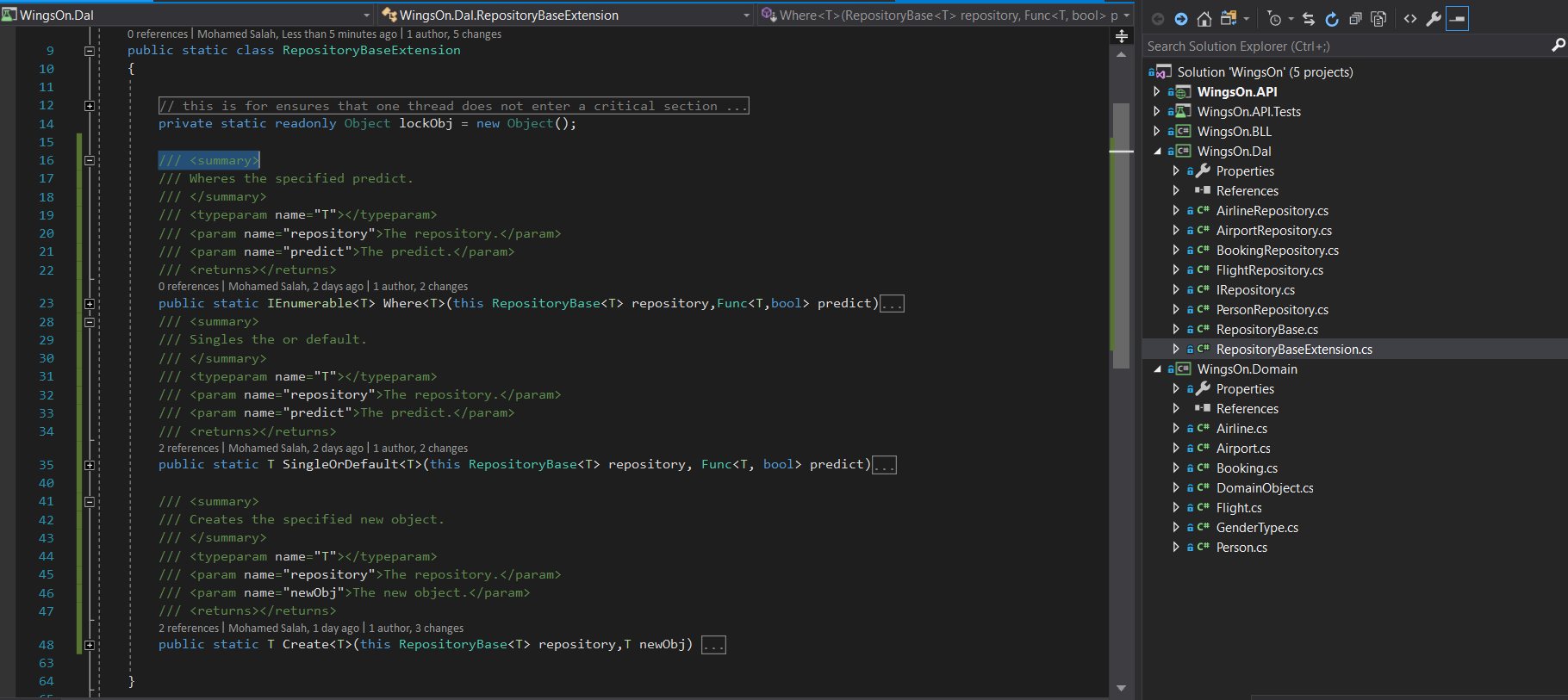
public static IEnumerable<T> Where<T>(this RepositoryBase<T> repository,Func<T,bool> predict)

/// adding singleOrDefault

public static T SingleOrDefault<T>(this RepositoryBase<T> repository, Func<T, bool> predict)

/// create new object with new Unique Id

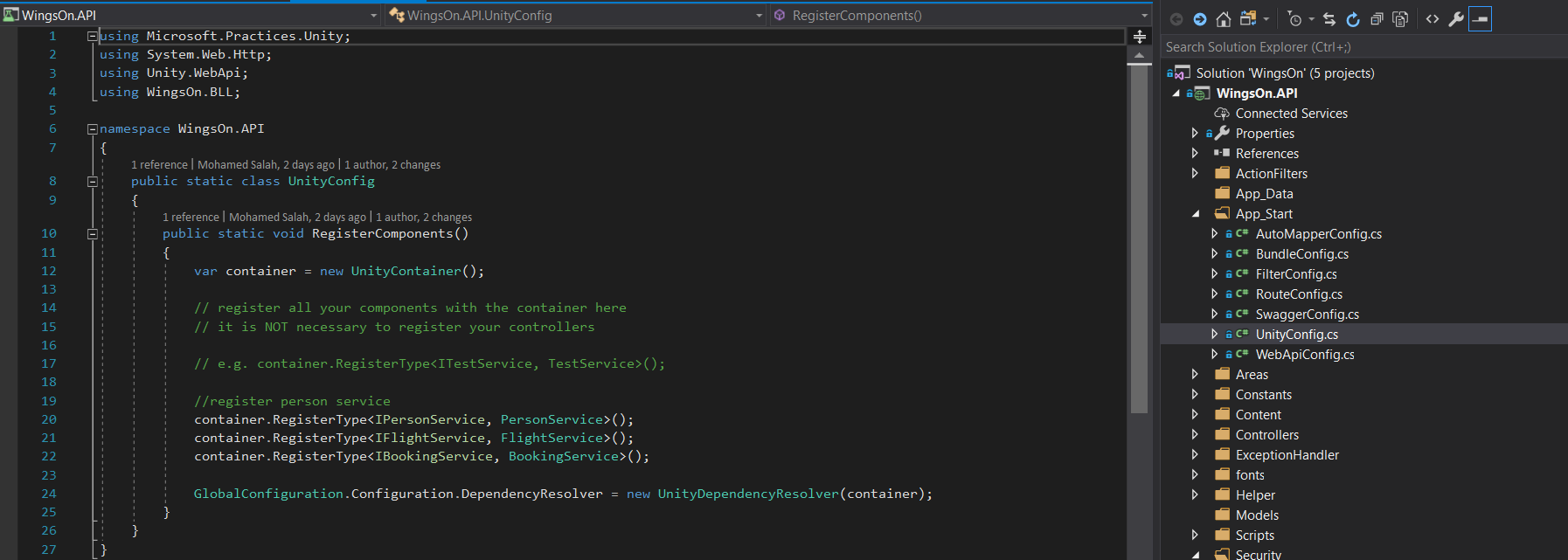
public static T Create<T>(this RepositoryBase<T> repository,T newObj)



### IoC

For applying the IoC project used **unity.webApi** packge and the configuration was put on file

Project uses the **unity.webApi** to inject the services tier



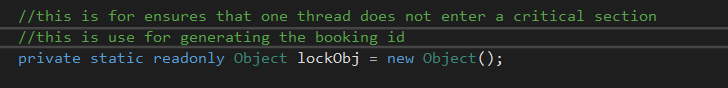
### Generating unique ID for Booking

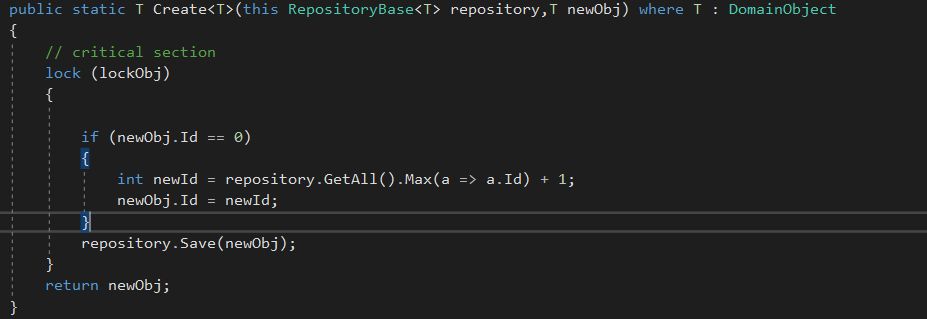
To generate the new Unique ID and to avoid to give the same Id for multiple threads

So the solution took the approach to make thread safe by using ***lock***.

The lock statement acquires the mutual-exclusion lock for a given object, executes a statement block, and then releases the lock. While a lock is held, the thread that holds the lock can again acquire and release the lock. Any other thread is blocked from acquiring the lock and waits until the lock is released

Alternative solution is to use ***Monitor***





# The Future Covered topics

Due to the interview will not go further solution was planned to add the following features

* Authentication and Authorization
* Add docker to the project
* Deploy on Azure
* Build CI/CD to complete the SDLC’
* Localization messages to support Multilanguage