4/4/2016

Design Document

Group 5

Ang Wei Jian
Bernard Chew Yu Lu
Nicholas Lim Wei Meng
Winnie Lew Ming Hui

TABLE OF CONTENTS

1. Introduction

1.1 Summary of ParkWhere

The name of the application is “ParkWhere”, and the reason why the team has decided to implement this project is because most of the time, drivers will have difficulties finding carparks to park their cars at certain places because of insufficient slots. They will then have to find directions to other car parks that are near their intended destination. There are also instances where the drivers’ gas is running low and they have to find the nearest gas station to top up their fuel before moving off to another destination. Weather is also another factor that drivers will usually consider before driving to some locations.

1.2 Objectives

The aim of this project is to provide useful information such as car park and weather information that drivers can use to help them plan their journeys in advance and also to provide them with information of petrol stations that they can utilize in the event that they are running low on gas.

1.3 Definitions

The definition of searching for a car park is when the user types in the location that they want to search for in the textbox. Alternative car parks refers to the car parks that are nearby the user's intended location in the event that the intended location's car park is unavailable. Weather information refers to the real time weather situation updates that are retrieved from the National Environment Agency (NEA). Google maps is a web mapping service that is developed by Google that provides satellite images and street maps of all places.

1.4 ParkWhere Website

ParkWhere is currently hosted in Microsoft Azure Web Services. The website can be accessed by the following link: http://parkwhere05.azurewebsites.net/

2. DETAILED ARCHITECTURE DESIGN

This section will describe the architecture of the system at different levels of abstraction.

High-Level Architecture

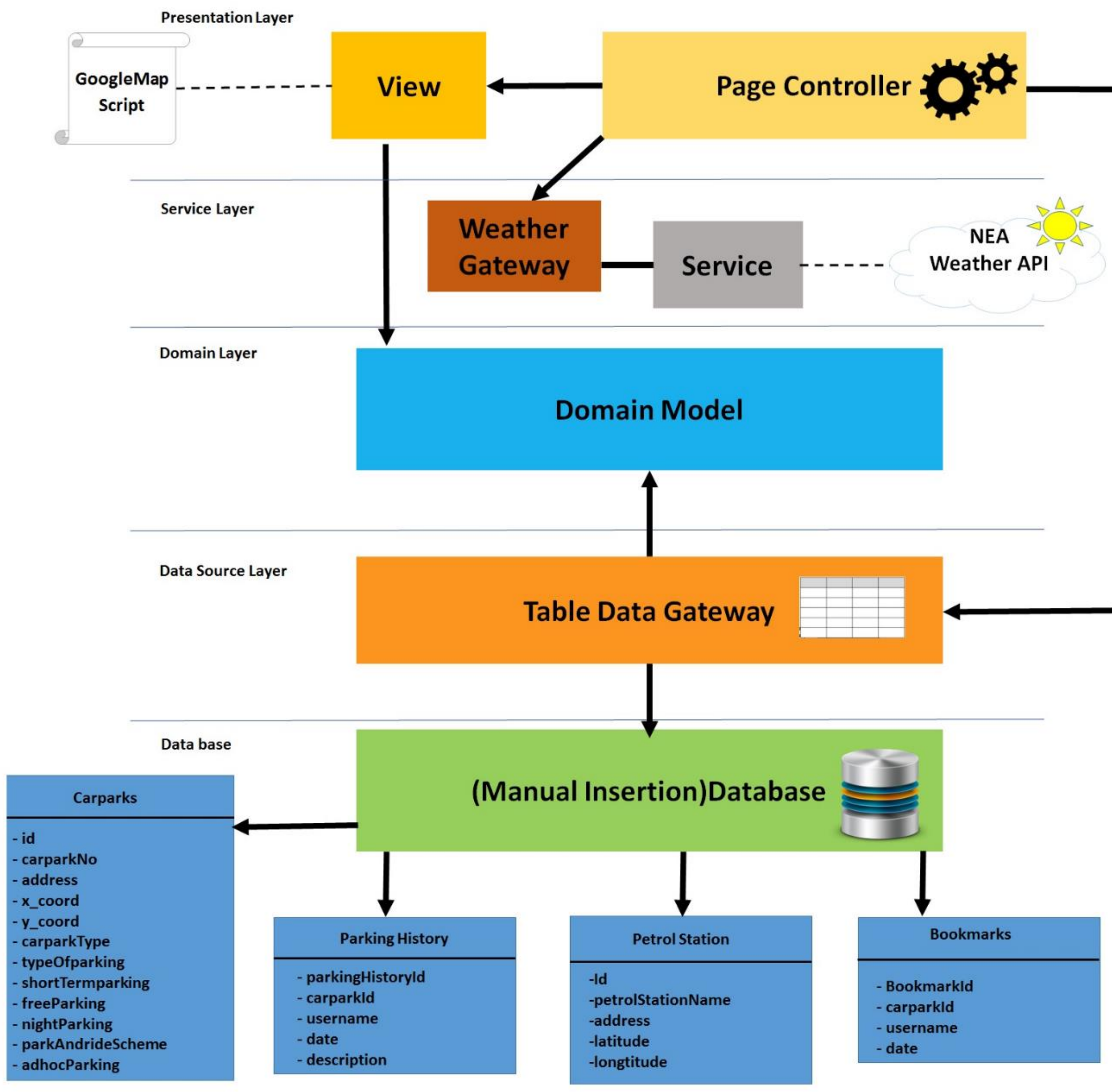

Figure 1 – High-Level Architecture Diagram

Fig 1 shows the overview of our system architecture for ParkWhere and the different design patterns implemented at each layer. The sections below will explain about the design and how different elements in your architecture work with one another.

2.1 PRESENTATION LAYER

Presentation layer displays how the View and Controller are interacting within the system.

View
The View will be in charge of displaying the model onto the User Interface. It will also be using GoogleMaps Script to display the locations of Carparks and Petrol stations all around Singapore. This is to show a geographical representation of the locations in form of map pins to the user.

Controller
The Controller will be handling user inputs to manipulate the model for the View to reflect the changes. The pattern used in the controller is Page Controller, which means that each controller will handle a specific page action on ParkWhere website.

2.2 SERVICE LAYER

The Service used in ParkWhere is the Weather API from NEA. This service will provide real-time weather information of different areas all around Singapore. This is so that it will aid drivers to be more prepared and informed about the weather conditions before heading out on a trip.

2.3 DOMAIN LAYER

The pattern used in this layer is Domain model and it is used to describe the data and business logic of our system ParkWhere. The Domain layer will handle the translation of the data into business/domain models that hold any business logic.

2.4 DATA SOURCE LAYER

The data source layer will separate between the domain and database. It will handle all the loading and storing between both the domain model and database, while allowing them to vary independently.

The pattern used in this layer is Table Data Gateway, which will work to interact with the Domain Model where there is an equivalent class in the domain model to map the data.

2.5 DATABASE LAYER

This layer is to represent the database and data tables being used in the system, the Fig 2 below is to show the entity relationship diagram of the database and the relations in ParkWhere.

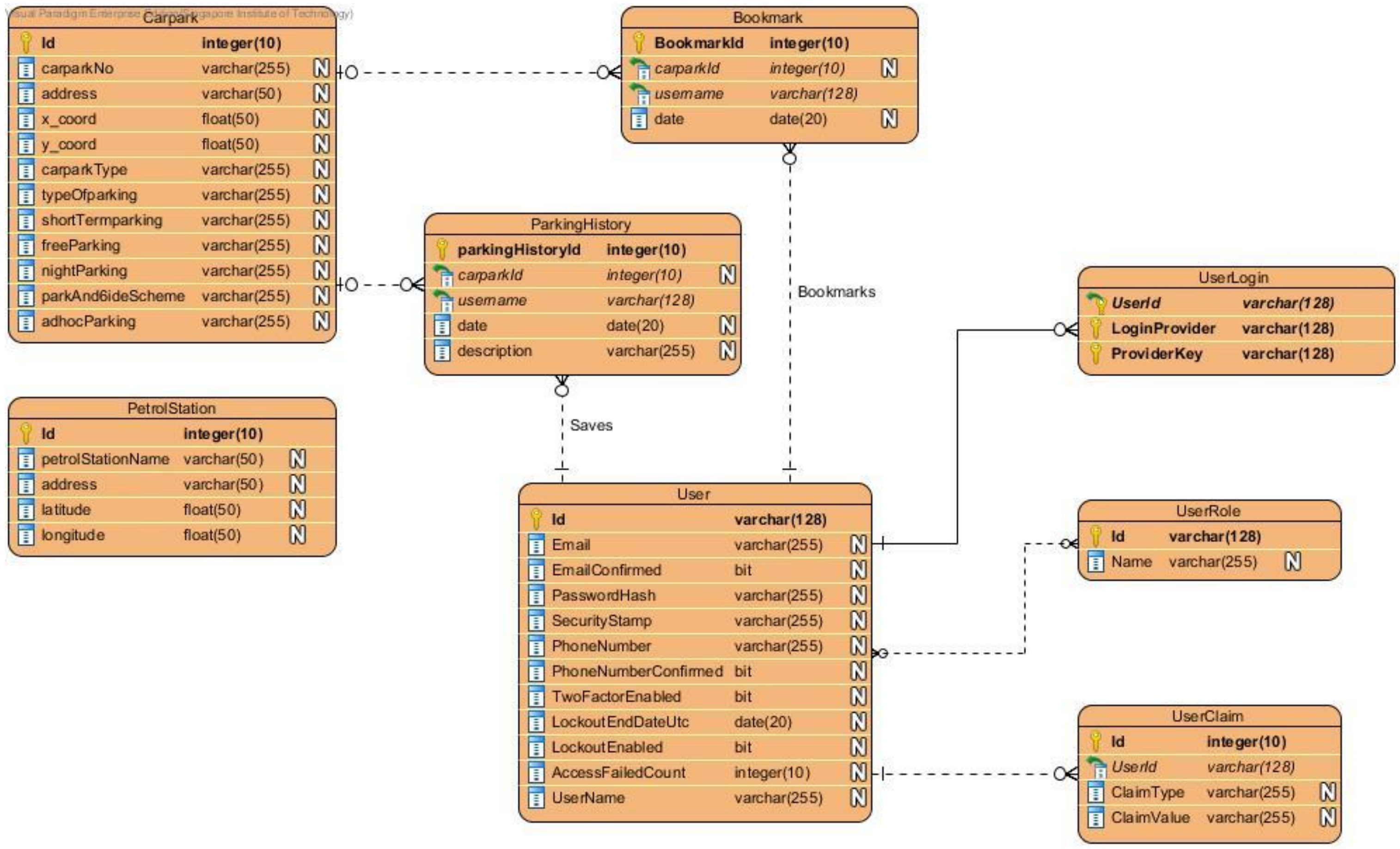

Figure 2 – ER Diagram

This section describes about the tables in the Entity Relationship Diagram as shown in Figure 2.

2.5.1 CARPARK TABLE

Table Name	**Carparks**	
Description	This table contains all the details of car parks.	
Attributes	id (integer) **Primary Key**	Unique id for Carpark table. Auto-increment primary key for the data table.
	carkparkNo (varchar)	Holds an unique code for the car park location
	address (varchar)	Contains the address of the car park.
	x_coord (float)	Contains the x coordinates of the car park location
	y_coord (float)	Contains the y coordinates of the car park location
	carparkType (varchar)	Identify if the car park is a multi-storey car park or surface car park
	typeOfparking (varchar)	Identify if the car park uses electronic parking or coupon parking
	shortTermparking (varchar)	Identify if the car park allows short term parking
	freeParking (varchar)	Describes the days and time of free parking
	nightParking (varchar)	Holds a true or false value if there night parking is allowed
	parkAndrideScheme (varchar)	Holds a true or false value if this car park is under the Park & Ride Scheme
	adhocParking (varchar)	Holds a true or false value if there ad hoc parking in this car park
Dataset Source	HDB Dataset	

2.5.2 PetrolStation Table

Table Name	**PetrolStation**	
Description	This table contains all the details of petrol stations.	
Attributes	Id (integer) **Primary Key**	Unique id for PetrolStation table. Auto-increment primary key for the data table.
	petrolStationName (varchar)	Name describing the petrol station location
	address (varchar)	Contains the address of the petrol station
	latitude (float)	Contains the lat coordinates of the car park location
	longitude (float)	Contains the lng coordinates of the car park location
Dataset Source	Shell Station Locator	

2.5.3 Bookmark Table

Table Name	**Bookmarks**	
Description	This table contains all bookmarks made by users. In addition, there will be two foreign keys in the table that will be linked with Carpark Table and User Table.	
Attributes	BookmarkId (integer) **Primary Key**	Unique id for Bookmark table. Auto-increment primary key for the data table.
	carparkId (integer) **Foreign Key**	Foreign key for Bookmark table. Unique id to identify car park. Primary key for Carpark table.
	username (varchar) **Foreign Key**	Foreign key for Bookmark table. Unique username to identify the user. Primary key for User table.
	date (date)	Contains the date of user creates bookmark
Dataset Source	- NA -	

2.5.4 PARKINGHISTORY TABLE

Table Name	**ParkingHistory**	
Description	This table contains all parking history locations made by users. In addition, there will be two foreign keys in the table that will be linked with Carpark Table and User Table.	
Attributes	parkingHistoryId (integer) **Primary Key**	Unique id for ParkingHistory table. Auto- increment primary key for the data table.
	carparkId (integer) **Foreign Key**	Foreign key for ParkingHistory table. Unique id to identify car park. Primary key for Carpark table.
	username (varchar) **Foreign Key**	Foreign key for ParkingHistory table. Unique username to identify user. Primary key for User table.
	date (date)	Contains the date of user creates bookmark
	description (varchar)	Contains a description entered by users for the saved history
Dataset Source	- NA -	

3. Package Diagram

This package diagram is used to show an in-depth view of our system architecture and to better understand how each and every component in the system work together and interact with.

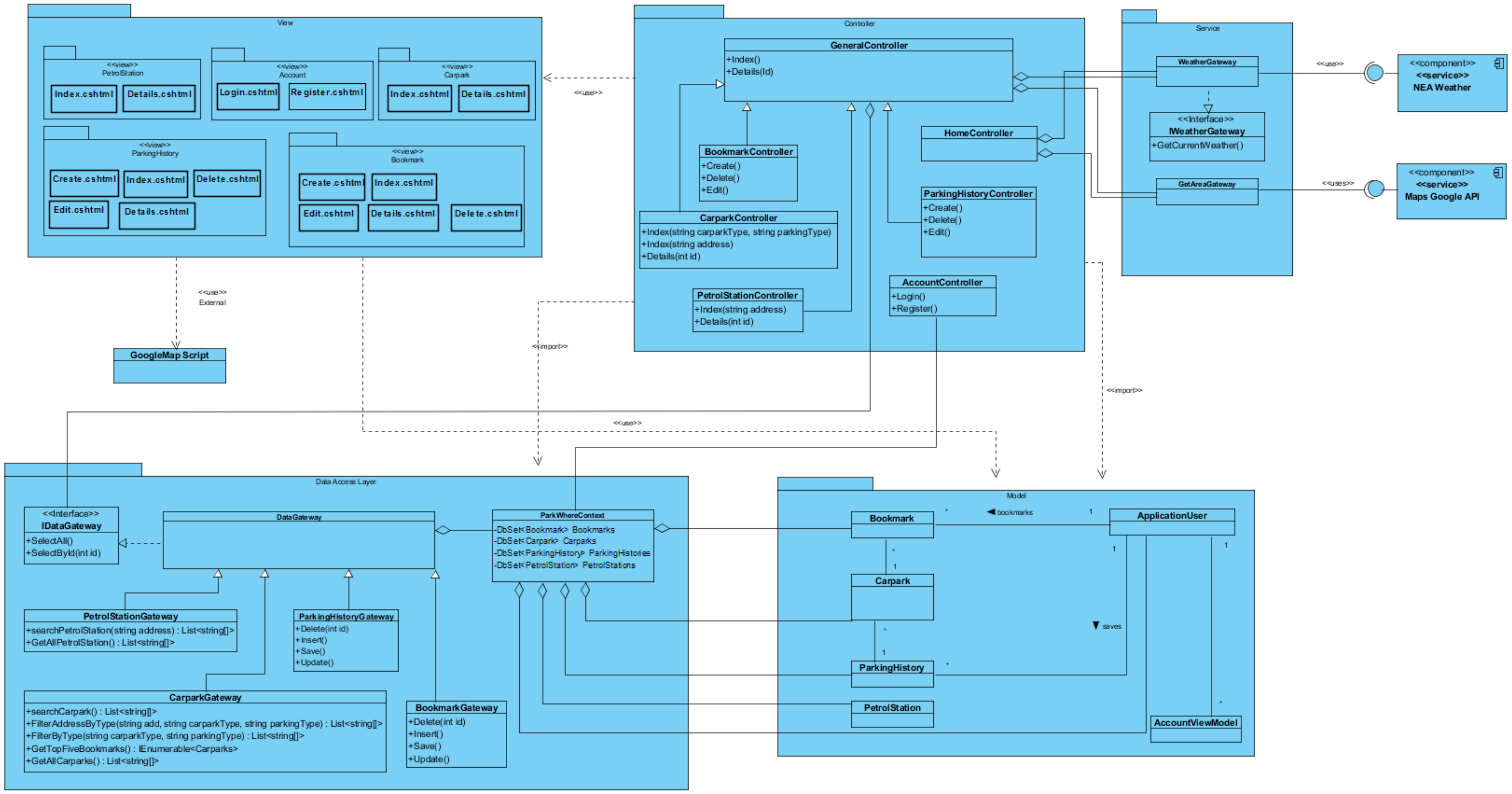

Figure 3 – ParkWhere Package Diagram

3.1 VIEW PACKAGE

The View package is a collection of different view packages and pages in ParkWhere system. The role of the View package in the system is to display the models onto the User Interface. There are a total of five view packages in ParkWhere System which are PetrolStation, Account, Carpark, ParkingHistory and Bookmarks view packages. The weather of the user's current location that is provided by NEA's weather API will be displayed in all the pages of ParkWhere.

3.1.1 PetrolStation

The PetrolStation package consists of two pages, which are the Index and Details page respectively.

Page Views

The Index page is meant to show the location of Petrol Stations near the user's current location and also enable the user to search up the location for a particular Petrol Station's address. The location of the Petrol Stations will be displayed in the form of map pins on a map to the user.

Upon clicking the "Details" link on the index page, the user will be brought to the Details page.

The Details page will display information of a particular Petrol station that was selected in the Index page by the user.

Process Flow

The PetrolStation view package will interact with the PetrolStationController to display the changes made in the model that is being passed back.

Upon loading the Index page it will trigger the "Index()" method from the PetrolStationController that is inherited by the GeneralController to display all the petrol stations stored in the database.

When the search button is clicked on the Index page to search up a Petrol Station's location, it will trigger the "index(string address)"method in the PetrolStationController , to process the search where the address passed into the method is the address that the user has entered in the search field in the Index page.

When the details of a Petrol Station is clicked, the “Details()” method from the PetrolStationController that is inherited from the GeneralController will be triggered to display the information of that Petrol Station in the Details page.

Script

Both the Index and Details page will make use of the GoogleMaps Script to display the map pins of the PetrolStation’s location on a map.

3.1.2 Account

The Account package consists of two pages, which are the Login and Register page respectively.

Page Views

The login page will facilitate users to enter their credentials to access the ParkWhere system to be able to access user authorized features such as bookmarking a carpark location and Parking History.

The register page will allow new users to register themselves to the ParkWhere System.

Process Flow

The Account view package will interact with the AccountController to display the changes made in model that is being passed back.

When the user has entered his/her credentials and clicked the "Login" button, it will trigger the "Login()" method in the AccountController to validate the account and allow authorization to the user authorized features in ParkWhere System.

When the user enter all the required fields to the user registration form and clicks "Submit", it will trigger the "Register()" method in the AccountController to insert a new user record into the database.

3.1.3 Carpark

The carpark package consists of two pages, which is the Index and Details page respectively.

Page Views

The Index page is meant to show the location of Carpark near the user’s current location and also enable the user to search up and filter the location for a particular Carpark’s address. The location of the Carpark will be displayed in the form of map pins on a map to the user.

The Details page will display information of a particular Carpark that was selected in the Index page by the user. It will also display two buttons, which is the “Bookmark” and “Save as Parking History” buttons for the user to save the selected Carpark as a parking history and bookmark.

Process Flow

The Carpark view package will interact with the CarparkController to display the changes made in model that is being passed back.

Upon loading the Index page it will trigger the “Index ()” method from the CarparkController that is inherited from the GeneralController to display all entries of the carpark.

When the search button in the Index page is clicked to search up a Carpark’s location, it will trigger the “index (string address)”method in the CarparkController, to process the search where the address being passed in the method is the address that the user has typed in the search field.

When selecting the different filter types and pressing the “Submit” button in the Index page, it will trigger the “index (string carparkType, string parkingType)” to process the filtering of different category types.

When clicking the details of a Carpark, the “Details ()” method from the CarparkController that is inherited by the GeneralController will be triggered to display the information of that Carpark in the Details page.

When either the “Bookmark” or “Save as Parking History” button in the Details page is clicked, it will pass both the “carparkId” and “address” to BookmarkController and ParkingHistoryController respectively.

Scripts

Both the Index and Details page will make use of the GoogleMap Script to display the map pins of the Carpark’s location on a map.

3.1.4 ParkingHistory

This package contains a total of five pages which is the Index, Create, Edit, Delete and Details page respectively.

Page Views

The Index page will display all of the parking history records of a user, to allow them to view where they have last parked their vehicle.

The Carpark address is being passed from the Carpark's Details pages upon clicking the "Save as Parking History" button as mentioned above. The Create page will then display a form with some preset fields like Carpark address, Date and Username. Upon clicking the "Submit" button, the form will be submitted and a record will be saved to the system.

The Edit page allows the user to edit a record that is selected in the Index page.

The Delete page will display the information of the selected record for the users to delete.

The Details page will display detailed information of a particular record that was selected in the Index page.

Process Flow

The ParkingHistory view package will interact with the ParkingHistoryController to display the changes made in model that is being passed back.

Upon loading the Index page it will trigger the "Index ()" method from the ParkingHistoryController that is inherited from the GeneralController to display all parking history records of a user.

When the submit button is clicked in the Create page, it will trigger the "Create()" method in the ParkingHistoryController to insert the record into the System.

When the delete button is clicked in the Delete page, it will trigger the "Delete()" method in the ParkingHistoryController to delete the selected record.

When the Update button is clicked in the Update page, it will trigger the "Update()" method in the ParkingHistoryController to update the selected record.

When the details button of a Parking History record is clicked, the "Details ()" method from the ParkingHistory Controller that is inherited by the GeneralController will be triggered to display the information of that particular record.

3.1.5 Bookmark

This package contains a total of five pages which is the Index, Create, Edit, Delete and Details page respectively.

Page Views

The Index page will display all of the bookmark records of a user, to allow them to view the user favorite parking location.

The Carpark address is being passed from the Carpark's Details pages upon clicking the "Bookmark" button as mention above. The Create page will then display a form with some preset fields like Carpark address, Date and Username. Upon clicking the "Submit" button, the form will be submitted and a record will be saved to the system.

The Edit page allows the user to edit a record that is selected in the Index page.

The Delete page will display the information of the selected record for the users to delete.

The Details page will display detailed information of a particular record that was selected in the Index page.

Process Flow

The Bookmark view package will interact with the BookmarksController to display the changes made in model that is being passed back.

Upon loading the Index page it will trigger the "Index ()" method from the BookmarksController that is inherited from the GeneralController to display all bookmarked records of a user.

When the "Submit" button is clicked in the Create page, it will trigger the "Create()" method in the BookmarksController to insert the record into the System.

When the "Delete" button is clicked in the Delete page, it will trigger the "Delete()" method in the BookmarksController to delete the selected record.

When the "Update" button is clicked in the Update page it will trigger the "Update()" method in the BookmarksController to update the selected record.

When the details of a Bookmarked record is clicked, the "Details ()" method from the BookmarksController that is inherited by the GeneralController will be triggered to display the information of that record.

3.2 CONTROLLER PACKAGE

There are a total of five controllers in this system. They are GeneralController, CarparkController, PetrolStationController, AccountController, ParkingHistoryController and BookmarksController respectively. The design pattern used for the controller is page controller pattern, which means that each controller handles a request for a specific page or action on the website. The role of the controller package in the system is to take in the user input and manipulate the model to update the view.

3.2.1 GeneralController

The GeneralController is the parent controller that implements two common methods, which is the "Index()" and "Details()" that will be inherited by four of the main controller CarparkController, PetrolStationController, ParkingHistoryController and BookmarksController.

The "Index()" method will call the "SelectAll()" method in the DataGateway to retrieve all the records in the selected database table.

The "Details()" method will call the "SelectById(int id)" method in the DataGateway to retrieve the record of the id.

The GeneralController will interact with WeatherGateway to get the current weather information.

3.2.2 CarparkController

CarparkController will interact with Carpark view package to handle the action for that view, like searching carpark, listing the carparks and the details of each carpark.

The CarparkController will also interact with the CarparkGateway, to call the respective methods in the gateway to retrieve the Carpark model that was queried by the CarparkGateway.

The "index(string address)" method from the controller will call the "searchCarpark()"method in the CarparkGateway to retrieve all the record of that address where the address that is passed into the method is the address that is entered by the user in the search field.

The "index(string carparkType, string parkingType)" method from the controller will call either the "FilterAddressByType()" or "FilterByType()"methods in the CarparkGateway depending on the inputs made by the user. The "FilterAddressByType()" method will query record of the address with the applied filter types and the "FilterByType()" method will query all the record with the applied filter type.

3.2.3 PetrolStationController

The PetrolStationController will interact with petrol station view package to handle the action for that view, like searching Petrol station, listing the Petrol station.

The PetrolStationController will also interact with the PetrolStationGateway, to call the respective methods in the gateway to retrieve the PetrolStation model that was queried by the PetrolStationGateway.

The "index(string address)" method from the controller will call the "searchPetrolStation()"method in the PetrolStationGateway to retrieve all the record of the address that is passed into the method that was entered by the user in the search field.

3.2.4 AccountController

AccountController will interact with Account view package to handle the action for that view, like Register and Login.

The "Login()" method in the AccountController will take in the model from the Account Login view page to validate the user who is logging in the System and authorized the entry.

The "Register()" method in the AccountController will take in the model from the Account Register view page to validate the entry of a register user who is wants to register to the System and save it to the database.

3.2.5 ParkingHistoryController

The ParkingHistoryController will interact with ParkingHistory view package to handle the action for that view, like creating, editing, deleting and listing the parking histories of a user.

The ParkingHistoryController will also interact with the ParkingHistoryGateway, to call the respective methods in the gateway to retrieve the ParkingHistory model that was handled by the ParkingHistoryGateway.

The "Create()"method from the controller will call the "Insert()"method in the ParkingHistoryGateway to insert an record into the ParkingHistory database table , then it will call the " Save()"method to save the changes made in the ParkingHistory database table .

The "Delete()"method from the controller will call the "Delete(int id)" from the controller method in the ParkingHistoryGateway to delete a particular record in the ParkingHistory database table , where a record id matches the id that is been passed. It will then call the "Save()"method to save the changes made in the ParkingHistory database table.

The "Update()" method from the controller will call the "Update()"method in the ParkingHistoryGateway to update a record in the ParkingHistory database table. It will then call the "Save()"method to save the changes made in the ParkingHistory database table .

3.2.6 BookmarksController

The BookmarksController will interact with bookmark view package to handle the action for that view, like creating, editing, deleting and listing the bookmarks of a user.

The BookmarksController will also interact with the BookmarkGateway, to call the respective methods in the gateway to retrieve the Bookmark model that was queried by the BookmarkGateway.

The "Create()"method from the controller will call the "Insert()"method in the BookmarkGateway to insert an record into the Bookmark database table , then it will call the " Save()"method to save the changes made in the Bookmark database table .

The "Delete()"method from the controller will call the "Delete(int id)" method in the BookmarkGateway to delete a particular record in the Bookmark database table , where a record id that matches the id that is been passed. It will then call the "Save()"method to save the changes made in the Bookmark database table .

The"Update()"method from the controller will call the "Update()" method in the BookmarkGateway to update a record in the Bookmarks database table. It will then call the "Save()"method to save the changes made in the Bookmark database table .

3.3 SERVICE PACKAGE

ParkWhere makes use of NEA's Weather Information API to get the current weather status in real time. The GeneralController will interact with the weather service, to get the current weather information.

ParkWhere uses a NEA's Weather REST API, to retrieve real-time weather information in Singapore. With reference t0 the package diagram in Figure 3, in the Services package, the WeatherGateWay will communicate with NEA to extract the weather information of the user' current location and display it to the user in the Index page at the Home View.

GetAreaGateway uses Maps Google JSON API to retrieve location address based on latitude and longitude coordinates provided. Upon retrieving the area name from the location details, the area name will be passed to WeatherGateway to extract weather information.

3.4 Data Access Layer (DAL) Package

The role of the DAL package is to provide simplified access to the data stored in an entity-relational database. The DAL package contains an interface, four table gateways and a ParkWhereContext class file.

Interface
The DAL contains an interface named IDataGateway. The interface will list out the specific operations for the DataGateway without any implementation, and the operations will be then later implemented in the DataGateway. The interface operations consist of "SelectAll()" and "SelectById(int id)" method respectively.

Pattern
The pattern used is Table Data Gateway, act as a gateway to database table and it will handle all the rows in a single table.

Gateway
There are a total of four table data gateways in the system which are, DataGateway, PetrolStationGateway, ParkingHistoryGateway, BookmarkGateway and CarparkGateway.

3.4.1 DataGateway

The DataGateway will implement the common operations in IDataGateway that is being used by the other four gateways.

The "SelectAll()" method will query all the records and return an IEnumerable model back to the respective controllers.

The "SelectById(int id)" method will query a record where it's id that matches the id that has been passed and return the model back to the respective controllers.

3.4.2 PetrolStationGateway

The PetrolStationGateway will inherit the both common operations "SelectAll()" and "SelectById(int id)" methods that is implement by the parent DataGateway. It will also implement "searchPetrolStation(string address)"method in PetrolStationGateway to fulfill the request of PetrolStationController.

The "searchPetrolStation(string address)" method will query the Petrol Stations which address is similar to the user's input address and return the model back to PetrolStationController .

3.4.3 ParkingHistoryGateway

The ParkingHistoryGateway will inherit the both common operations “SelectAll()” and “SelectById()” method that is implement by the parent DataGateway.It will also implement four methods “Delete()”,”Insert()”,”Save()”and”Update()” method that will fulfill the request of ParkingHistoryController.

The “Delete(int Id)” method will execute the SQL to delete the parking history record which id matches the id selected by the user that is being passed, in the ParkingHistory database table. It will call the “Save()”method to the changes made in the ParkingHistory database table and return the model back to ParkingHistoryController.

The “Insert()” method will take in the ParkingHistory model that is pass from the ParkingHistoryController. It executes the SQL to insert a parking history record in the ParkingHistory database table. It will call the “Save()”method to the changes made in the ParkingHistory database table and return the model back to ParkingHistoryController.

The “Update()” method will take in the ParkingHistory model that is passed from the ParkingHistoryController. It executes the SQL to update a parking history record in the ParkingHistory database table and return the model back to ParkingHistoryController.

3.4.4 BookmarkGateway

The BookmarkGateway will inherit the both common operations “SelectAll()” and “SelectById()” method that is implemented by the parent DataGateway. It will also implement four methods “Delete()”, “Insert()”, “Save()” and “Update()” method that will fulfill the request of BookmarksController.

The “Delete(int Id)” method will execute the SQL to delete the bookmark record which id matches the id selected by the user that is being passed, in the Bookmark database table .It will call the “Save()”method to the changes made in the Bookmark database table and return the model back to BookmarksController.

The “Insert()” method will take in the Bookmark model that is pass from the BookmarksController. It executes the SQL to insert a bookmark record in the Bookmark database table. It will call the “Save()” method to save the changes made in the Bookmark database table and return the model back to BookmarksController.

The “Update()” method will take in the Bookmark model that is passed from the BookmarksController. It executes the SQL to update a bookmark record in the Bookmark database table and return the model back to BookmarksController.

3.4.5 CarparkGateway

The CarparkGateway will inherit both the common operations “SelectAll()” and “SelectById()” method that is implemented by the parent DataGateway. It will also implement three methods “searchCarpark()” , “FilterAddressByType()” and “FilterByType()” method that will fulfill the request of CarparkController.

The “searchCarpark (string address)” method will query the Carpark which address is similar to the user’s input address and return the IEnumerable Carpark model back to the CarparkController.

The “FilterAddressByType(string add, string carparkType, string parkingType)” method will query Carpark address that is entered by the user whose carpark type and parking type that matches the input parameters of the method. It will return back an IEnumerable Carpark model back to the CarparkController.

The “FilterByType(string carparkType, string parkingType)” method will query Carparks whose carpark type and parking type that matches the input parameters of the method. It will return back an IEnumerable Carpark model back to the CarparkController.

3.4.6 ParkWhereContext

The context class will get the data set from the database and the moment the application runs, the context class will connect to the database and populate the dbset with the data from the database. ParkWhere have a total of four datasets: Carpark, PetrolStation, Bookmark and ParkingHistory. It also handles Unit of Work to keep track of every action in transaction that affects the database.

3.5 MODEL PACKAGE

The model package shows the different models in ParkWhere system. The pattern used in this package is Domain model and it is used to describe the data and business logic of our system ParkWhere. The model consists of Carparks, Bookmarks, PetrolStations and ParkingHistory. The models associated with Accounts is auto generated by Visual Studio.

4 USE CASE DIAGRAM

This diagram shows the different features the user have access to, there are a total of six features as shown below in figure 4.

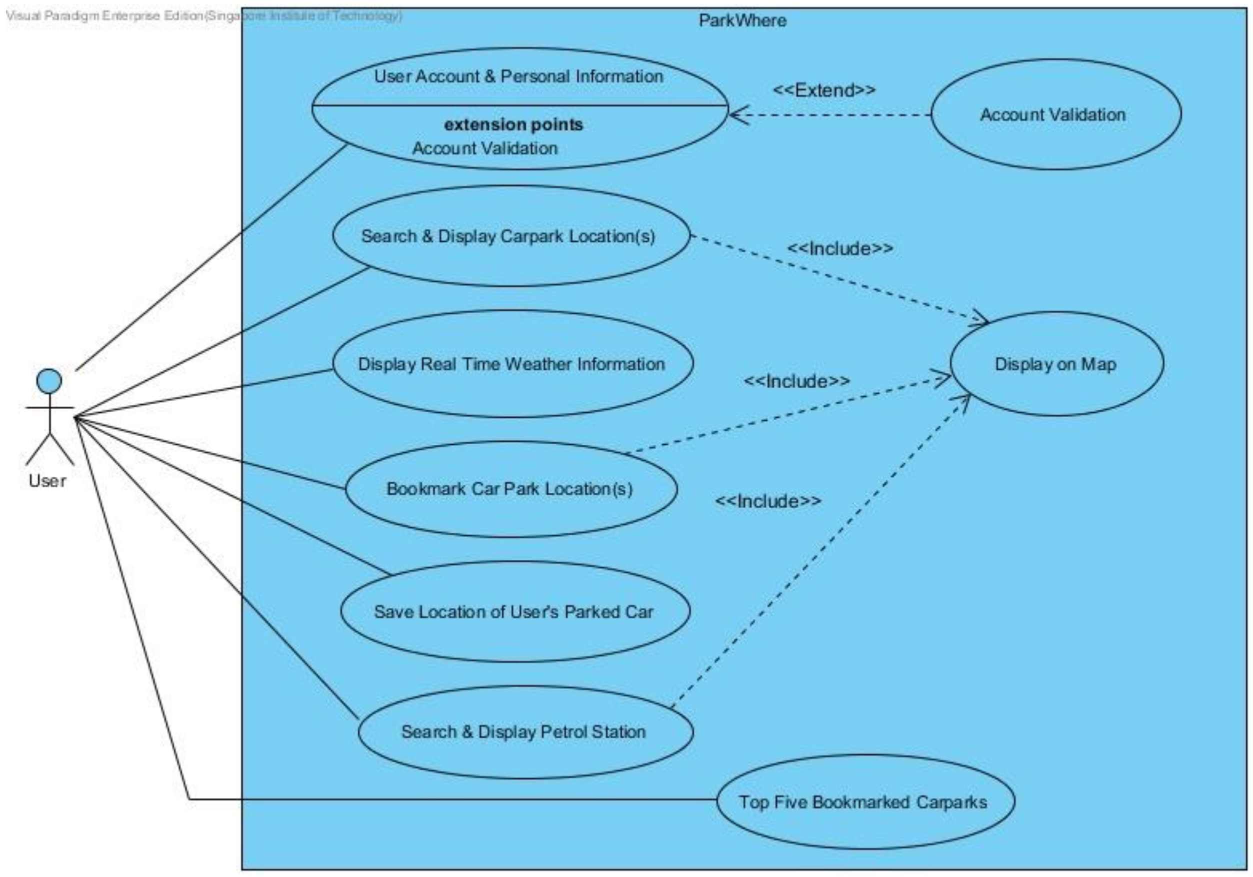
Visual Paradigm Enterprise Edition(Singapore Institute of Technology)
ParkWhere
User Account & Personal Information
extension points
Account Validation
<<Extend>>
Account Validation
Search & Display Carpark Location(s)
<<Include>>
Display on Map
Display Real Time Weather Information
<<Include>>
Bookmark Car Park Location(s)
<<Include>>
Save Location of User's Parked Car
Search & Display Petrol Station
Top Five Bookmarked Carparks
User

Figure 4 – Use Case Diagram

5 SEQUENCE DIAGRAM

Three functions are chosen from the use case diagram above: Search & display car park location, bookmark car park location and save location of user's parked car.

These three functions are chosen to be represented through sequence diagrams to explain the interaction between the different elements in the system.

5.1 SEARCH & DISPLAY CARPARK LOCATION

The sequence in figure 5 shows the flow of when a user search for a carpark, and how it's displayed back to the user.

1. The user first sends a request to the web server to search for a car park
2. It will call the search function and pass in the car park address to CarparkController.
3. The controller will then call searchCarpark method and pass in the car park address to the method in CarparkGateway.
4. The CarparkGateway will then interact with ParkWhereContext to query for the corresponding car parks in the database and return the respective result.
5. IEnumerable Carpark model is returned to CarparkController from CarparkGateway.
6. The controller will pass this back the WebServer.
7. The data model is then displayed to the user.

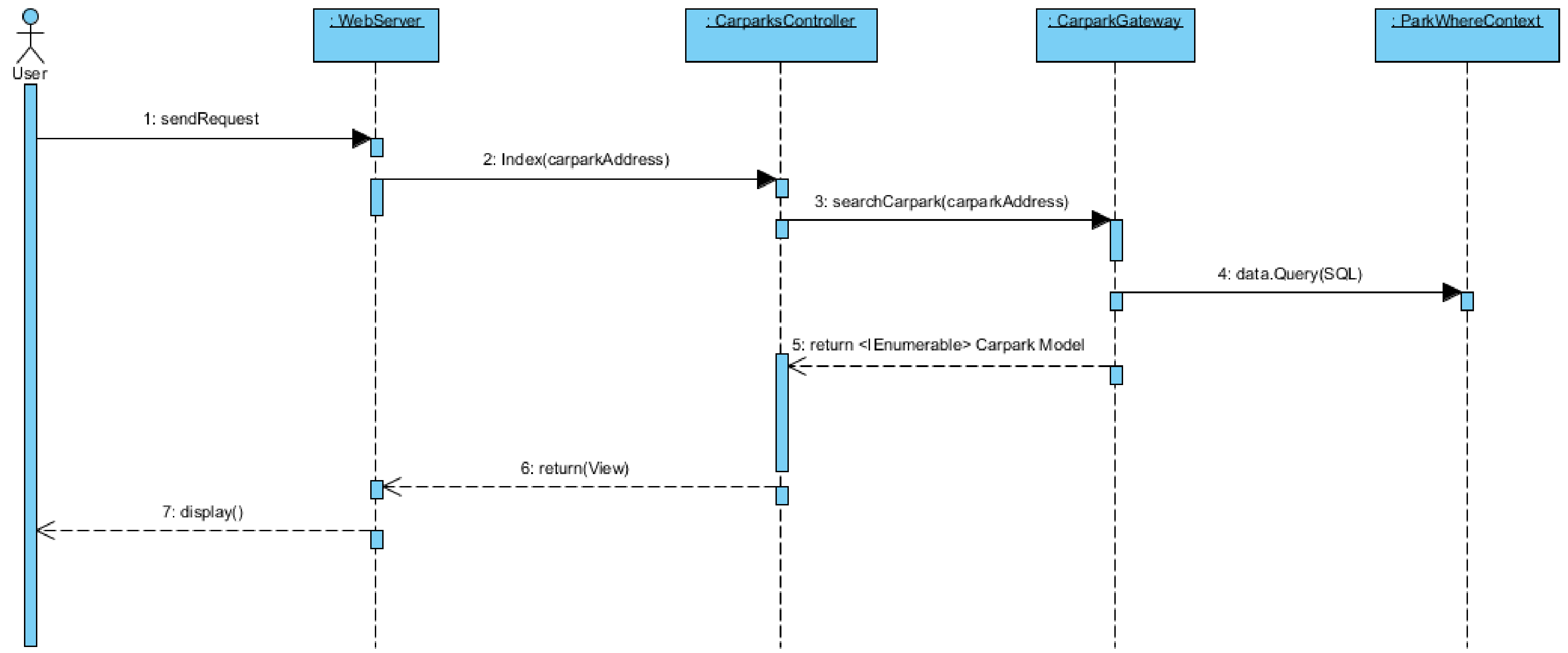

Figure 5 – Search & Display Carpark Location

5.2 BOOKMARK CARPARK LOCATION

The sequence in figure 6 shows the flow of when a user search for carparks, selects a car park and bookmark the car park. It will also show how the result is displayed back to the user.

Pre-requisite: User must be logged in and complete sequence in Search & Display Car Park Locations.

1. After successful completing Search & Display Car Park Locations sequence, the user send a request to the web server to bookmark a selected car park.
2. The WebServer will call create method in BookmarksController, passing together with the necessary information, such as id, date and username.
3. The controller will then create a bookmark object and include all necessary information and pass this model to Insert method in BookmarkGateway
4. BookmarkGateway will make use of ParkWhereContext to add the model using db.add.
5. Upon adding, it will call db.SaveChanges to update the database table.
6. After successful add, BookmarksController will redirect to Index by calling its own method Index.
7. Upon calling Index, it will call SelectAll method in BookmarkGateway to retrieve the updated list of bookmarks.
8. BookmarkGateway will call db.ToList in ParkWhereContext to retrieve all bookmark records.
9. Upon retrieving successfully, it will return IEnumerable Bookmark model back to BookmarksController.
10. BookmarksController will then return view back to the WebServer.
11. The data model is displayed to the user.

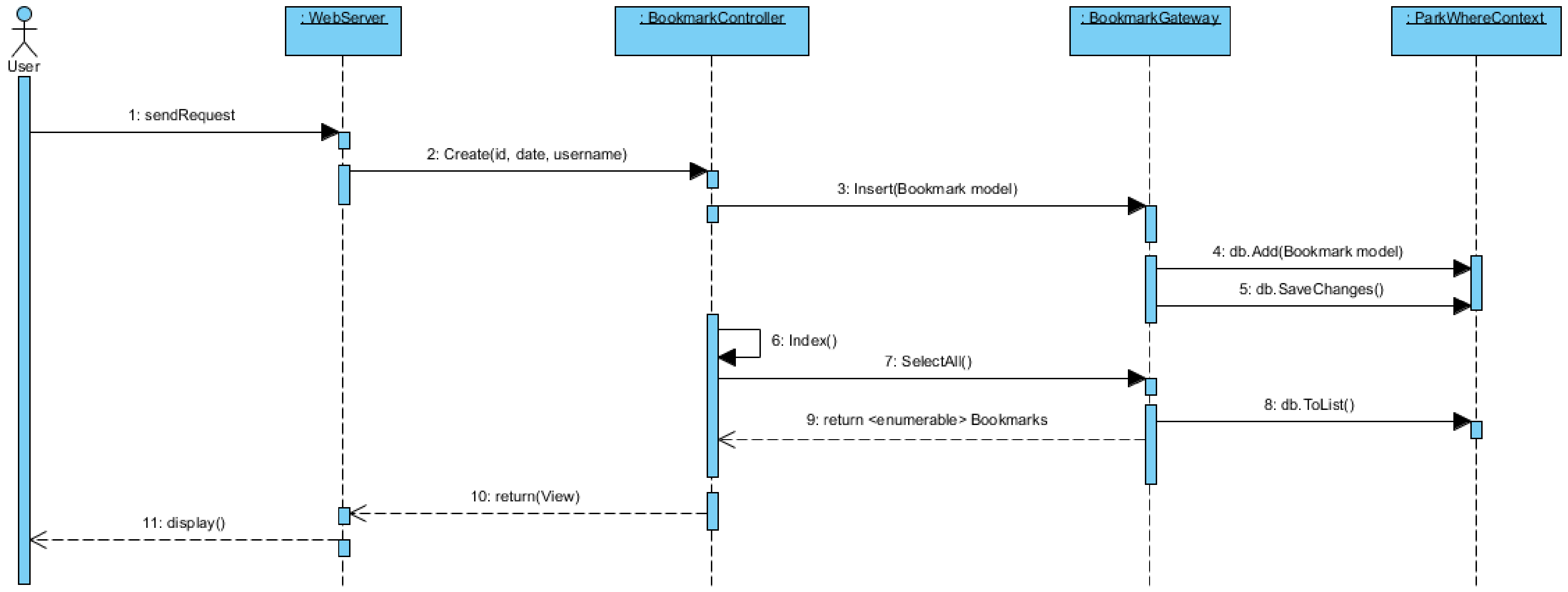

Figure 6 – Bookmark Carpark Location

5.3 SAVE CARPARK LOCATION IN PARKING HISTORY

The sequence in figure 7 shows the flow of when a user search for a car park, and then select a car park to be saved in the Parking History. It will also show how the result is displayed back to the user.

Pre-requisite: User must be logged in and complete sequence 2.3.1 Search & Display Car Park Locations.

1. After successful completing Search & Display Car Park Locations sequence, the user send a request to the web server to save a selected car park in parkingHistory.
2. The WebServer will call create method in ParkingHistoryController, passing together with the necessary information, such as id, date, username and description.
3. The controller will then create a parkingHistory object and include all necessary information and pass this model to Insert method in ParkingHistoryGateway
4. ParkingHistoryGateway will make use of ParkWhereContext to add the model using db.add.
5. Upon adding, it will call db.SaveChanges to update the database table.
6. After successful add, ParkingHistoryController will redirect to Index by calling its own method Index.
7. Upon calling Index, it will call SelectAll method in ParkingHistoryGateway to retrieve the updated list of saved parkingHistory.
8. ParkingHistoryGateway will call db.ToList in ParkWhereContext to retrieve all parkingHistory records.
9. Upon retrieving successfully, it will return IEnumerable ParkingHistory model back to ParkingHistoryController
10. ParkingHistoryController will then return view back to the WebServer.
11. Webserver and data model is displayed to the user.

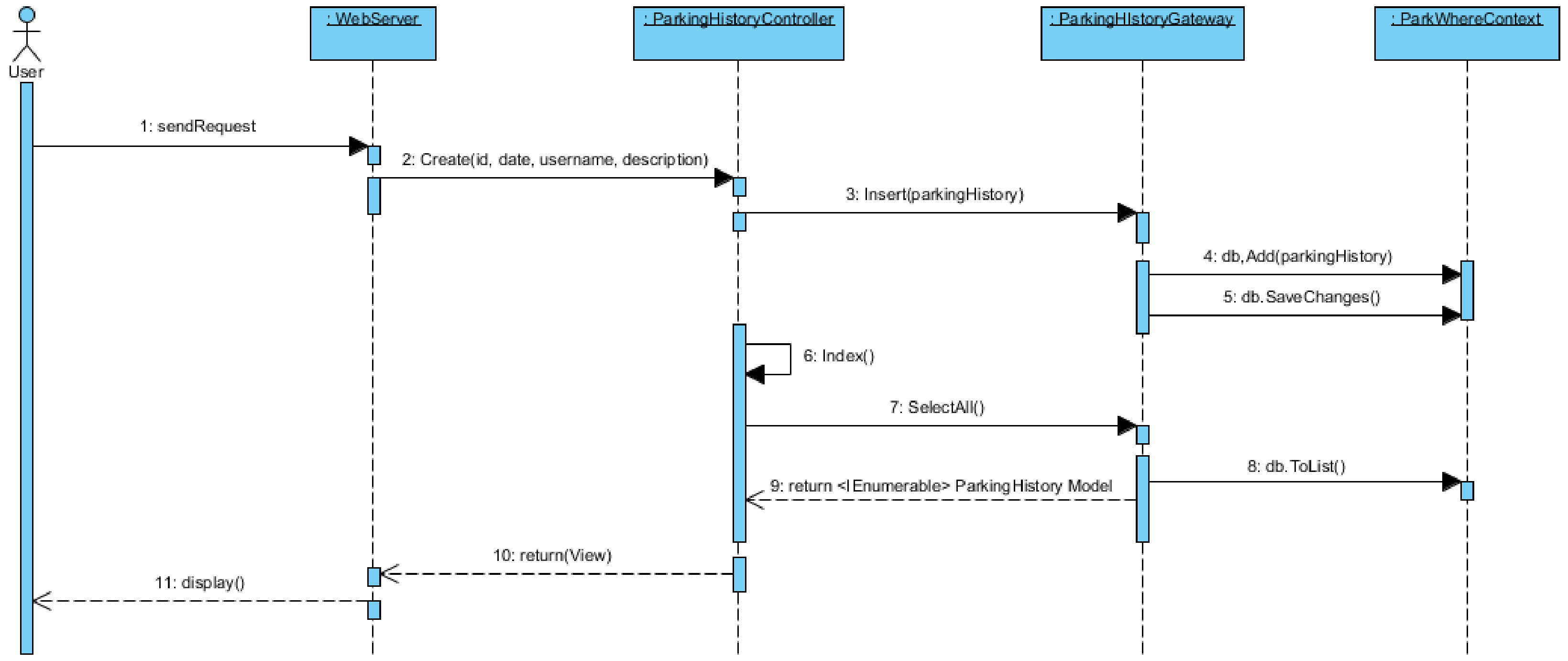
: WebServer
: ParkingHistoryController
: ParkingHIstoryGateway
: ParkWhereContext
User
1: sendRequest
2: Create(id, date, username, description)
3: Insert(parkingHistory)
4: db,Add(parkingHistory)
5: db.SaveChanges()
6: Index()
7: SelectAll()
8: db.ToList()
9: return <IEnumerable> Parking History Model
10: return(View)
11: display()

Figure 7 – Save Carpark Location in Parking History

6. Design Quality

6.1 Quality Attributes

For this system, the team will be using Microsoft's standard of quality attributes as a guideline to design and develop ParkWhere. A reference of Microsoft's standards can be found in the references section at the end of this document.

Usability

Usability attribute in Microsoft's standard refers to the how well the system or application meets the requirements of the user which will result in overall good user experience after using the system. To achieve intuitiveness in ParkWhere, we implement a couple features to aid the user to have good and satisfactory user experience in ParkWhere.

For example, when the user wants to search for an address to show the carpark within that area, they need not fill in the complete address to search, the system have implement an autocomplete using Google API to suggest predictive addresses to the user. This eliminates the erroneous input by the user.

The number of interaction the user needs to take in order to fulfil a task using ParkWhere is also keep to a minimum. For example, in order to search for a carpark, the user only needs to click on the Carpark tab on the navigation bar, type in the address and click the submit button and the search results will be displayed on a map. This short and user friendly steps will help ease the use of the user and give an overall good experience. The results that are displayed in the map give users interaction to click and zoom to know their surroundings.

Availability

According to Microsoft's standards, Availability is defined as the proportion of time the system is up and functioning which can be used as a measurement as a percentage of the total system downtime over a period of time.

With reference to Rob Boucher Jr (2015), ParkWhere is currently hosted using Microsoft Azure Web Application and the data that is stored in the Microsoft Azure database. As such, the data will have automatic backups, point-in-time restore capabilities as well as replicated copies that are available across geographical areas.

Scalability

Scalability in Microsoft’s standard refers to the ability of the system to handle increases in load without affecting the performance of the system as well as having the ability to be expanded.

ParkWhere is using Microsoft Azure and it allows the storing of data of multiple users. A user can create more than one bookmark or saved history and it will all be stored in the database which can be retrieved back.

In the event that the system needs more capacity to hold more users or data, the purchasing of additional storage capacity to scale up the system is possible.

According to Microsoft Azure, “Microsoft Azure supports load balancers for virtual machines and cloud services to allow your applications to scale up and also to prevent a single system failure that could result in loss of service availability. There are three load balancers in Azure: Azure Load Balancer, Internal Load Balancer (ILB), and Traffic Manager." By having our system on Azure, ParkWhere gain the benefits from Azure load balancers, as it helps the application to increase the number concurrent users and the reliability of the application. One of the algorithm used by Azure to help manage large traffic is hash-based distribution, it helps to hash the map traffic to available servers to distribute and ease the congestion.

Reusability

Reusability in Microsoft's standard is the capability for components and subsystems to be suitable for use in other applications. It minimizes duplication of components as well as the time to implement these components again.

With reference to the package diagram in Figure 3, the GeneralController holds the common methods that the other four controllers frequently uses like the Index() and Details() methods. The implementation of the GeneralController saves the implementation time, instead of duplicating the same codes in the other controllers. It also encourages the reusing of components which will save memory space.

The DataGateway class also holds common methods that the other gateway classes uses, which is the SelectAll() and SelectById() methods. Implementing the DataGateway class will also saves implementation time by reusing methods.

Interoperability

Interoperability in Microsoft's standard is defined as the ability to communicate and exchange data with external parties or systems and incorporating it together to operate successfully.

In ParkWhere system it will interact with external services such as Google API to display Google maps on the application and weather information will also be retrieved from National Environment Agency (NEA) API to be displayed to the users. In addition, any new changes in the external service will not affect the system implementation of accessing the resources as their implementation is external to the system.

Supportability

Supportability in Microsoft's standard is defined as the ability of the system to provide information to the user in the event that an error or exception occur which will help the users identify and resolve issues to ensure smooth usage of the application.

In ParkWhere, error messages will be displayed to the users in the event that an error occurs. For example, when a user enters the wrong combination of username and password when accessing their accounts, the system will prompt an error message notifying them that they have entered an invalid username or password and they need to re-enter their login credentials.

When a user is registering for an account and did not fill in the mandatory fields or filled in information in the wrong format, error messages will also be displayed to notify them that they need to enter information in the correct format in order to proceed. The application also implements intuitive design like date picker and auto complete into the application to prevent erroneous input by the user.

Reliability

Reliability in Microsoft's standard is defined as the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.

ParkWhere provides high level of reliability as it makes use of Microsoft Azure as the main source for database. According to Microsoft Azure (2015), "it is guarantee at least 99.9% of the time customers will have connectivity between their Web or Business Microsoft Azure SQL Database and our Internet gateway", hence promoting high level of reliability.

6.2 Design Principles

This section describes about the design principles applied in ParkWhere System.

High Cohesion and Low Coupling

In ParkWhere solution, modules groups related functionality together for high cohesion and low coupling.

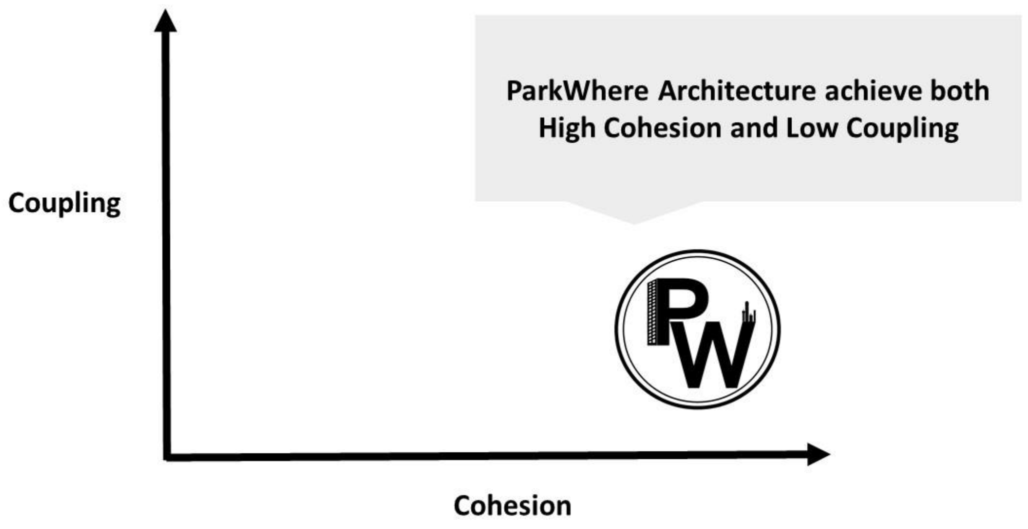
ParkWhere Architecture achieve both High Cohesion and Low Coupling
Coupling
Cohesion

Figure 8 – ParkWhere Cohesion & Coupling

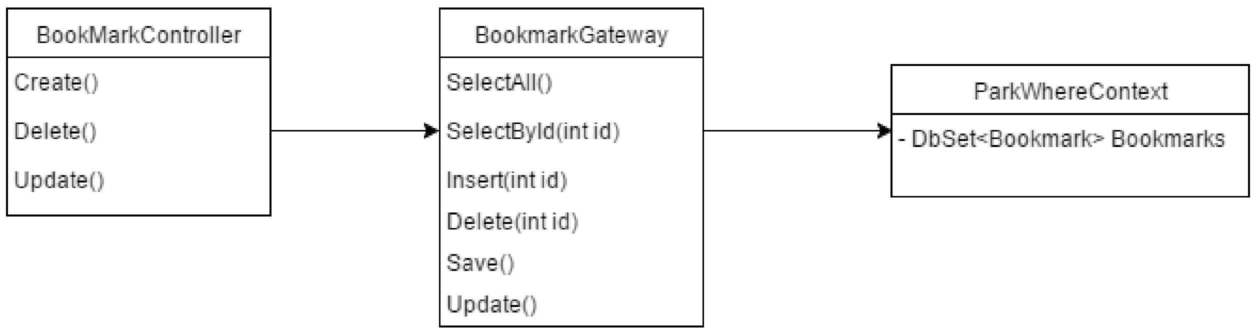

Figure 9 – Detailed Example of High Cohesion & Low Coupling

High Cohesion

In ParkWhere system the controllers pattern used are Page Controller , and it represent that each Controller will handle a specific page action on ParkWhere website. This shows that each controller is focus on handle each functionalities in each view package as show in Fig 1.

Functional Cohesion

Each classes in ParkWhere have a single focus of functions that promotes high cohesion. This means that the application is divided into distinct classes, each holding onto distinct functionalities. For example, BookmarkController in ParkWhere contains multiple methods, such as Create(), Update() and Delete(). These methods only focuses on bookmark related functionality and hence it promotes readability as all close related functions are contained in a single module and not polluted with irrelevant functions.

Low Coupling

In ParkWhere system architecture each package like controller and gateway are segregated and independent and they do not need to know each implement and they can interact with each other.

Data Coupling

Each module in ParkWhere is designed to be independent of any other module. In ParkWhere, it uses service interface that gets only necessary information required. For example, WeatherGateway service provides only current weather forecast and hides the implementation from the GeneralController that calls the service, hence promoting low coupling.

In the event that there are any new changes in the service, new implementation will not affect the modules that interacting with it. Therefore, data coupling promotes the ease of adding in new implementation easily without requiring much modification to current modules.

Single Responsibility

In ParkWhere, with reference to the package diagram in Figure 3, all the controllers that are separated into different controllers and linked together with a general controller.

For example, in the controller package, CarparkController, BookmarksController, ParkingHistoriesController and PetrolStationsController are linked to one GeneralController. Each of these controllers are divided with distinctive responsibilities, this shows a single focus of responsibility, which fulfills the single responsibility principle.

In the Data Access Layer, there is also one DataGateway that all the other gateways in the system like the CarparkGateway, BookmarkGateway, ParkingHistoriesGateway and PetrolStationGateway are linked to, which also fulfills the single responsibility principle.

7. References

- Chapter 16: Quality Attributes. (n.d.). Retrieved March 7, 2016, from https://msdn.microsoft.com/en-us/library/ee658094.aspx
- SLA for SQL Database. (n.d.). Retrieved March 7, 2016, from https://azure.microsoft.com/en-us/support/legal/sla/sql-database/v1_0/
- Software Design principles and design pattern reading. (n.d.). Retrieved March 7, 2016, from https://blogs.msdn.microsoft.com/ampuri/2013/07/24/software-design-principles-and-design-pattern-reading/
- Load Balancers in Microsoft Azure. (2015, August). Retrieved March 16, 2016, from https://azure.microsoft.com/en-us/documentation/articles/load-balancer-overview/
- Krisragh. (2016, March). Scheduler High-Availability and Reliability. Retrieved March, 2016, from https://azure.microsoft.com/en-in/documentation/articles/scheduler-high-availability-reliability/
- Boucher Jr, Rob. (2015, June). Introducing Microsoft Azure. Retrieved March 16, 2016, from https://azure.microsoft.com/en-us/documentation/articles/fundamentals-introduction-to-azure/