System Design
Team 2

Find My Bathroom

Joshua Liang 010380383

Marietta Asemwota 010587421

Parth Patel 010455315

Gerardo Garcia 012030629

Preface

The Model-View-Controller (MVC) is an architectural pattern that separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.); these interactions will be sent to the View and the Model.

This architecture is very popular for designing web applications, which fits our user requirements. MVC can also be used to design mobile, desktop, and other clients. This is efficient for us as we decide to expand our project to become responsive, as it can be accessed from the web, mobile, and other clients. Programming languages such as Java and PHP, which we will be using, have MVC frameworks that are used in web application development.

Introduction

With the separation of the use of code between the three file types, you obtain a separation in web application logic. You will obtain an almost complete separation in the programming logic and interface code (markup and styles), and further separation in the type of logical code for the application.

Basically, a web application or piece of software that follows the MVC structure separates the three main types of functionality into three types of files: models, views, and controllers. This allows for each portion to be designed, implemented, and tested independently from any other one, keeping code organized. Keeping the code organized means being able to find what is needed quickly, test features, correct or alter them quicker, and add new functionality with ease. It also means more efficient code, and a better way to reuse code for faster applications.

Probably one of the greatest benefits is that many developers understand and use the MVC structure for creating web applications. If developers use any of the popular web development frameworks, then they understand and use the MVC structure as well. Because of this consistency, managing a project between several developers can be easier as well.

Controller

The controller can be considered the “middle man” of the application. It works with the user, taking in data, and then working with the model to get the appropriate data or calculation, and then working with the view to show the response to the user.

When our user carries out any action, it first goes to the controller. It will take in any data, for example, $_GET and $_POST variables in PHP, and determine what should be done with the data. In short, models deal with handling data and extended functionality. Therefore, the controller’s job at this point is to determine which model should be called, and then call the appropriate function within that model. For example, our controller will determine which bathrooms to be called based off of user’s choice of criteria, and then call the navigate, rate, add, etc. appropriate function. After calling the function, it will catch the result.

After it gets whatever information it needs from the model, it will determine what view to send that feedback to and send the user there. For example, in our system, the controller would get the bathroom information from the model, and send that feedback to our map view and list view, and they will send the user there.

Model

A model is simply a representation of something we need to deal with our application. It is a “model” for something we must represent in code, such as a book, user, bank account, or whatever. The model is responsible for holding the functions and variables that are involved with whatever it’s representing. You can think of a model’s logic as the core concept to object-oriented programming — models are just our “classes”. However, don’t let this confuse you as controllers are technically structured as classes as well.

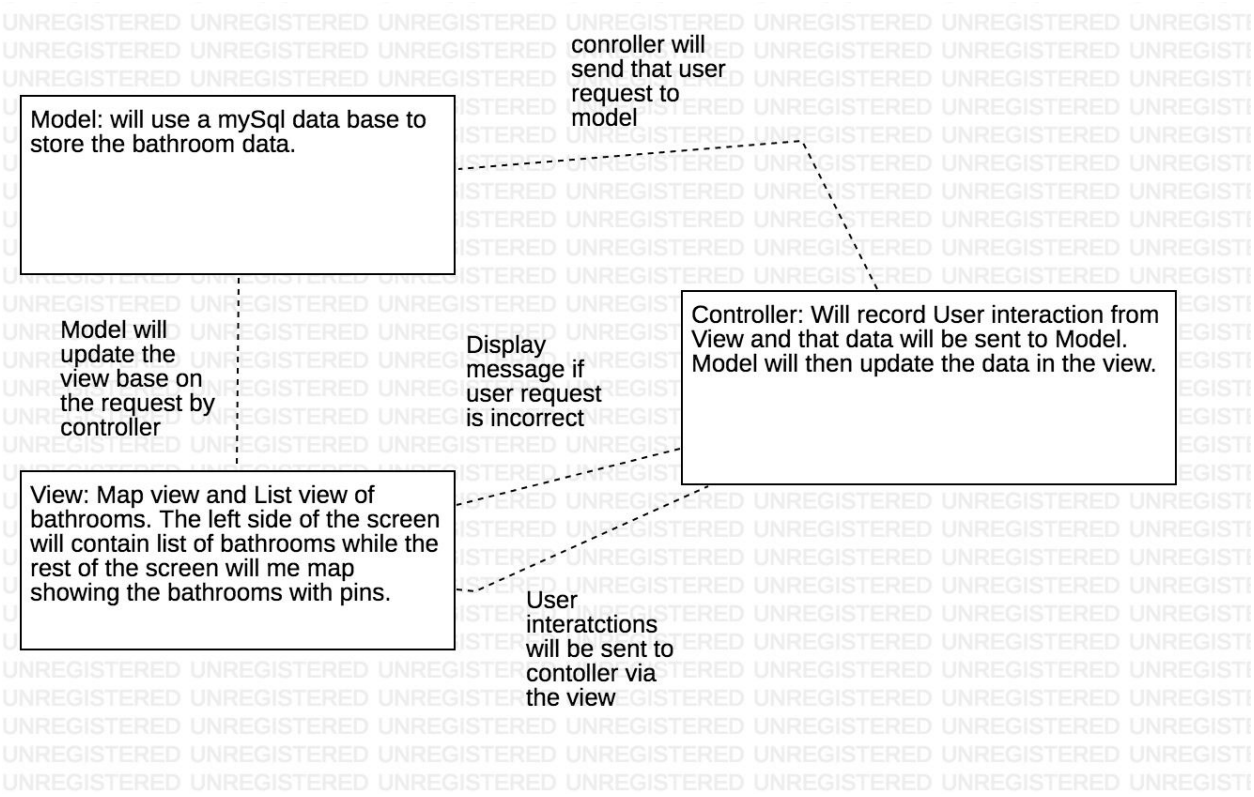
For example, a “Bathroom” model would be a class that represents our bathroom. (It models our bathroom.) It holds variables that pertain to the bathroom’s information, such as the bathroom’s location. It also holds functions that get the bathroom’s criteria, set it, and update it. Other functions could also be to get the user’s input when the user decides to add a bathroom or do anything else that would require interaction with the database, or otherwise an extensive function.

View

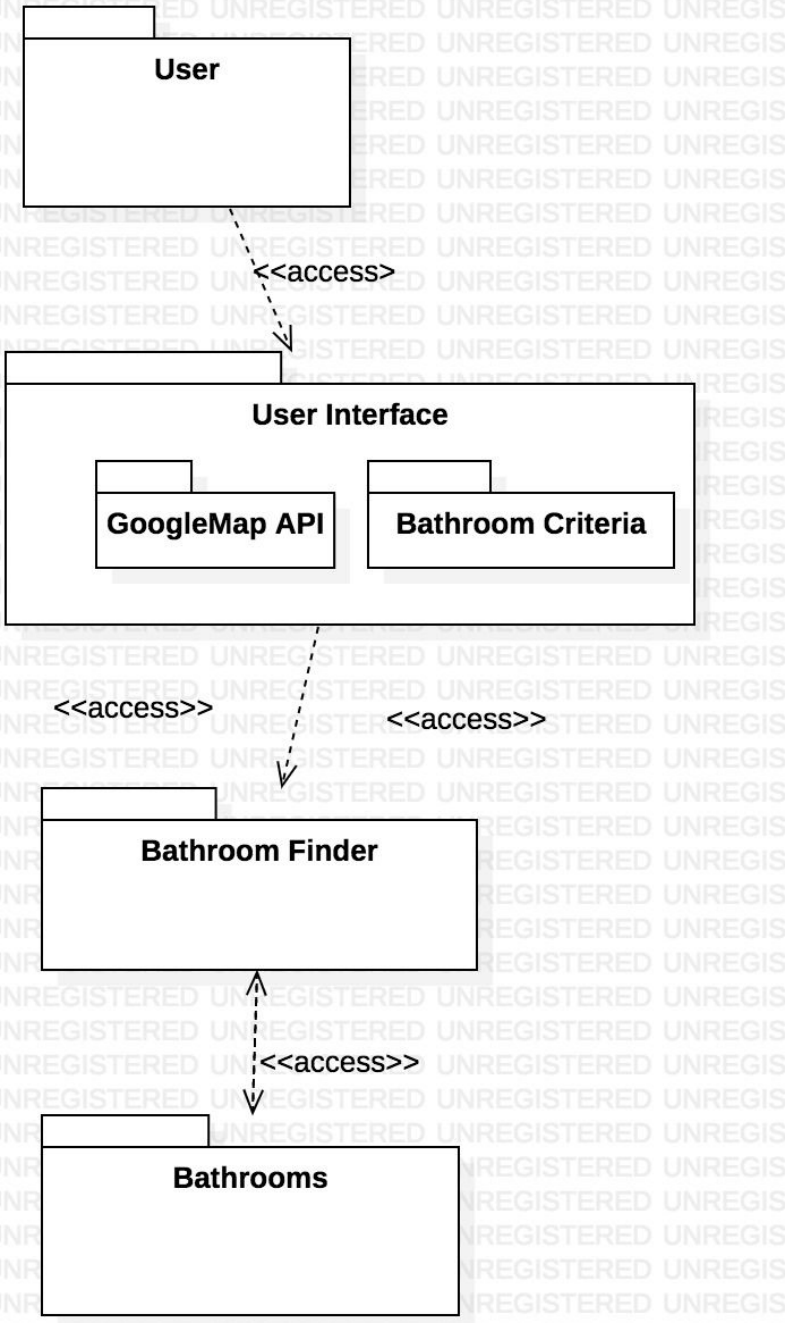
Finally, after the controller requests information from the model, it sends it to a view. A view is just like the application’s templating system — there might be a view for a certain type of page layout like our map and list GUI, or a mobile view, or a view for our criteria choosing. A view will contain all of the markup, CSS, javascript, etc. that you use to design a web page.

It’s what the user sees, and what the controller turns to the user. The controller simply forwards the user to the correct view, based on what they’re trying to do, but after they’ve received data from the model and forwarded this information to the view. The view then displays the information it’s given, in the format it’s structured with.

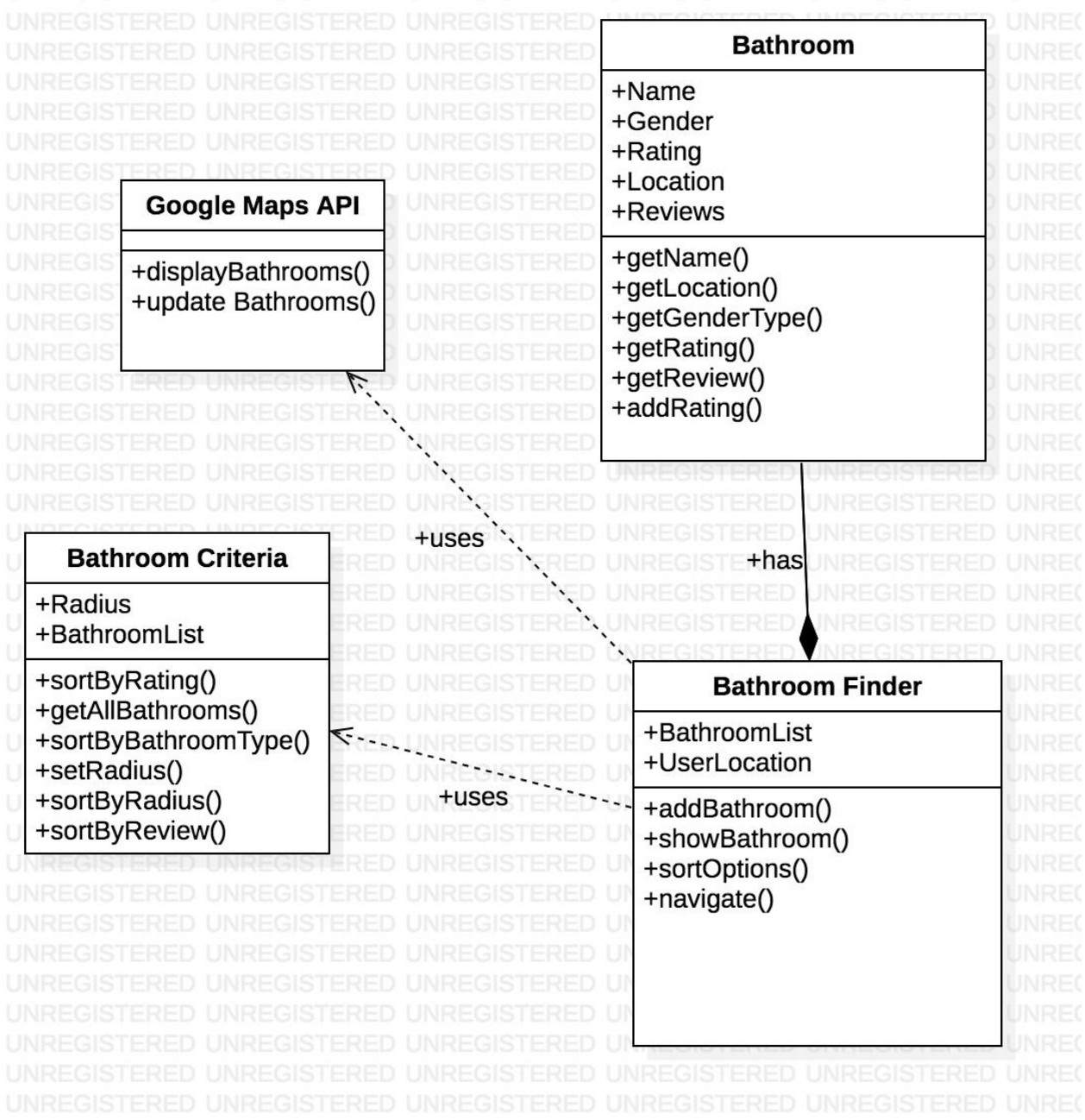
Architecture Pattern:



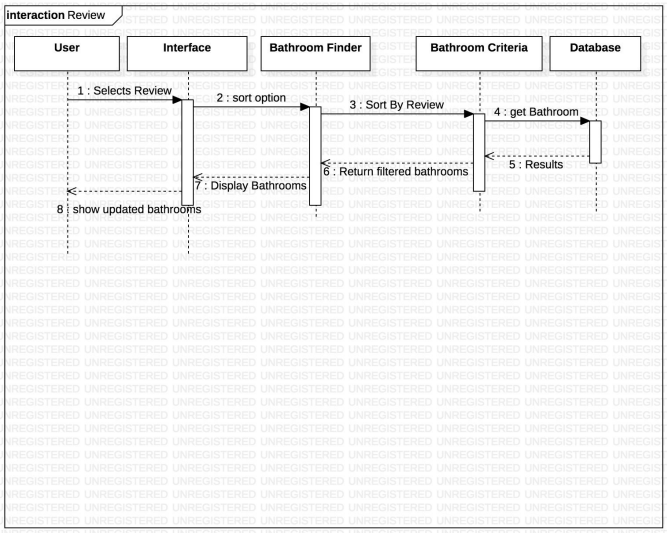
Package Diagram:

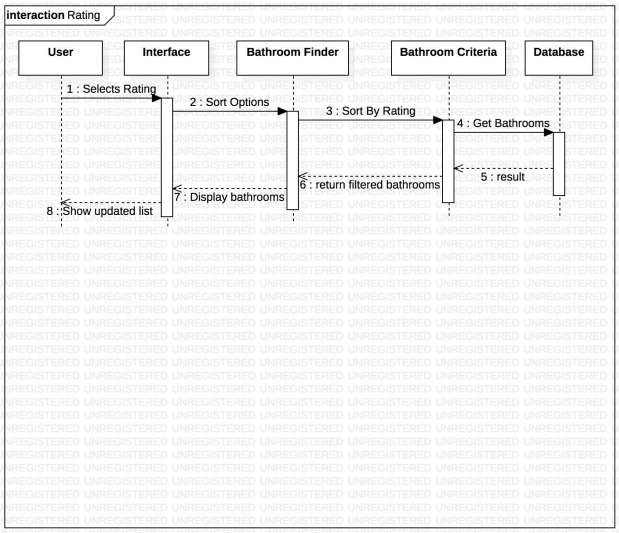

User
<<access>
User Interface
GoogleMap API
Bathroom Criteria
<<access>>
<<access>>
Bathroom Finder
<<access>>
Bathrooms

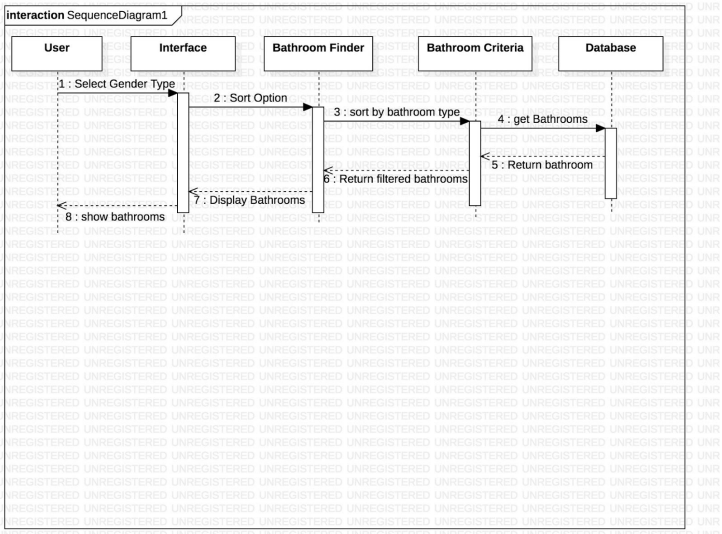
Class Diagram:

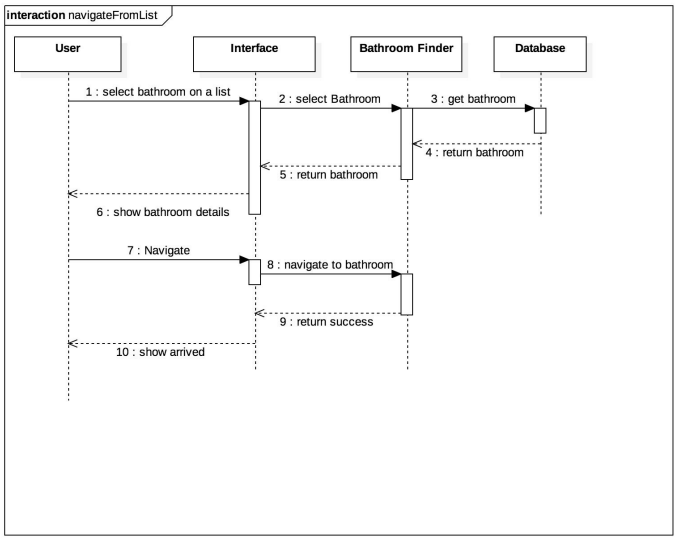


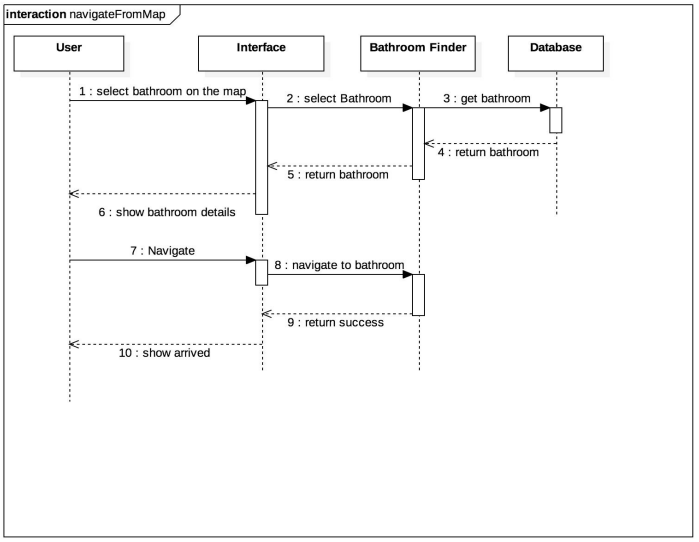
Sequence Diagram:

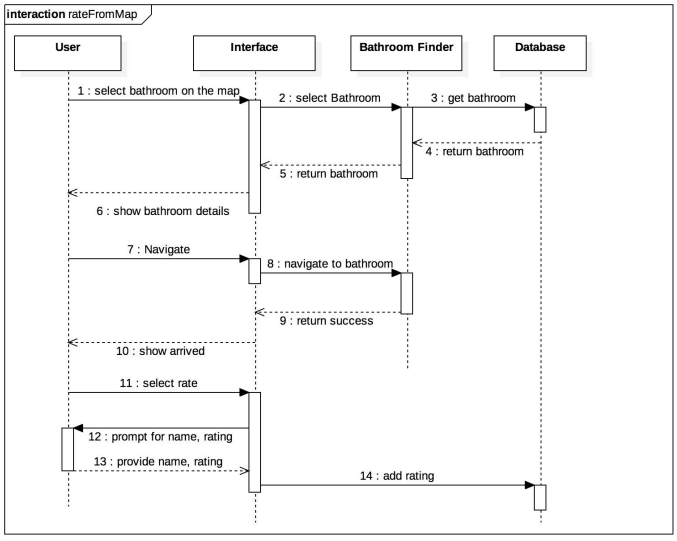

interaction Review
User
Interface
Bathroom Finder
Bathroom Criteria
Database
1 : Selects Review
2 : sort option
3 : Sort By Review
4 : get Bathroom
5 : Results
6 : Return filtered bathrooms
7 : Display Bathrooms
8 : show updated bathrooms

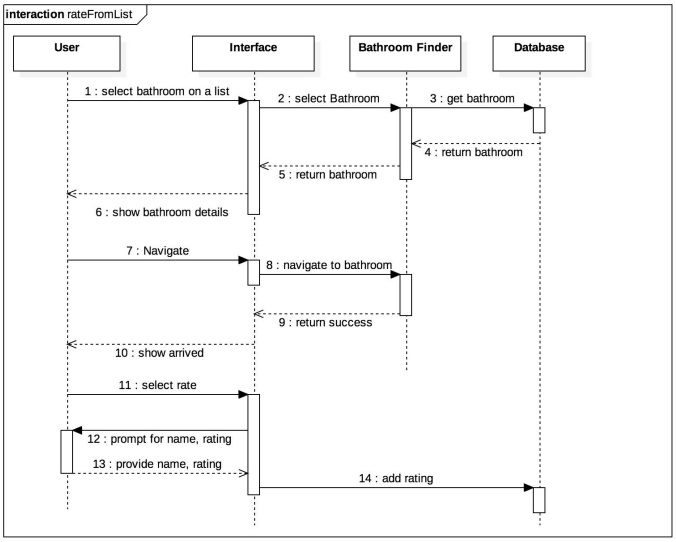

interaction Rating
User
Interface
Bathroom Finder
Bathroom Criteria
Database
1 : Selects Rating
2 : Sort Options
3 : Sort By Rating
4 : Get Bathrooms
5 : result
6 : return filtered bathrooms
7 : Display bathrooms
8 : Show updated list

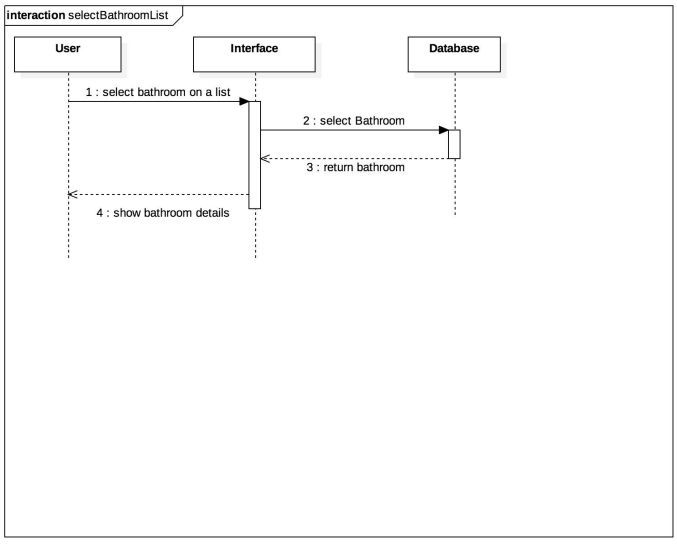

interaction SequenceDiagram1
User
Interface
Bathroom Finder
Bathroom Criteria
Database
1 : Select Gender Type
2 : Sort Option
3 : sort by bathroom type
4 : get Bathrooms
5 : Return bathroom
6 : Return filtered bathrooms
7 : Display Bathrooms
8 : show bathrooms

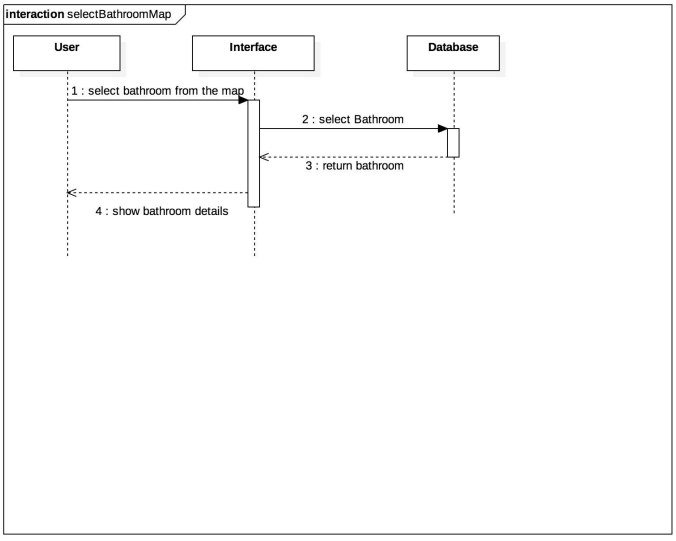

interaction navigateFromList
User
Interface
Bathroom Finder
Database
1 : select bathroom on a list
2 : select Bathroom
3 : get bathroom
4 : return bathroom
5 : return bathroom
6 : show bathroom details
7 : Navigate
8 : navigate to bathroom
9 : return success
10 : show arrived

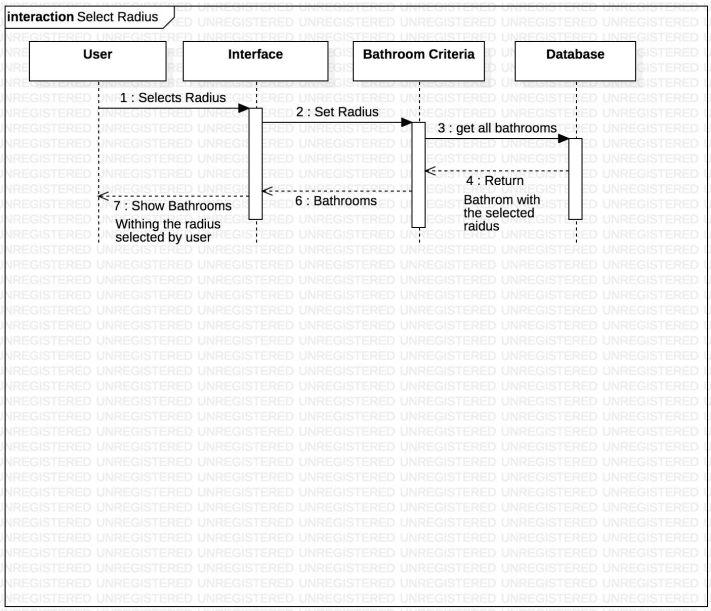

interaction navigateFromMap
User
Interface
Bathroom Finder
Database
1 : select bathroom on the map
2 : select Bathroom
3 : get bathroom
4 : return bathroom
5 : return bathroom
6 : show bathroom details
7 : Navigate
8 : navigate to bathroom
9 : return success
10 : show arrived


interaction rateFromMap
User
Interface
Bathroom Finder
Database
1 : select bathroom on the map
2 : select Bathroom
3 : get bathroom
4 : return bathroom
5 : return bathroom
6 : show bathroom details
7 : Navigate
8 : navigate to bathroom
9 : return success
10 : show arrived
11 : select rate
12 : prompt for name, rating
13 : provide name, rating
14 : add rating


interaction rateFromList
User
Interface
Bathroom Finder
Database
1 : select bathroom on a list
2 : select Bathroom
3 : get bathroom
4 : return bathroom
5 : return bathroom
6 : show bathroom details
7 : Navigate
8 : navigate to bathroom
9 : return success
10 : show arrived
11 : select rate
12 : prompt for name, rating
13 : provide name, rating
14 : add rating


interaction selectBathroomList
User
Interface
Database
1 : select bathroom on a list
2 : select Bathroom
3 : return bathroom
4 : show bathroom details


interaction selectBathroomMap
User
Interface
Database
1 : select bathroom from the map
2 : select Bathroom
3 : return bathroom
4 : show bathroom details


interaction Select Radius
User
Interface
Bathroom Criteria
Database
1 : Selects Radius
2 : Set Radius
3 : get all bathrooms
4 : Return
Bathrom with
the selected
raidus
6 : Bathrooms
7 : Show Bathrooms
Withing the radius
selected by user