CS160 Assignment #3 - Implementation

Table of Contents

Project and Title Authors

Title: ParkHere

Team Name: BlueJ Boys

Team Members:

- Kevin Vu (008861554)
- Stanley Plagata (009072700)
- Ricky Reyes (010674794)
- Nelson Nguyen (009233250)

Preface

Expected readership is any customers and stakeholders.

Version 1.0 on September 26, 2017: Initial Document Creation

We are creating the requirement document for the *ParkHere* application to describe the needed actions for the project.

Version 1.1 on October 7, 2017: Design Document

We are creating the design document for the *ParkHere* application to describe the mobile application. For developers and architecture design.

Version 1.2 on November 5, 2017: Implementation Document

We are creating the implementation document for the *ParkHere* application to describe the implementation changes for the mobile application.

Introduction

The problem with going to highly urbanized areas is the lack of cheap and/or nearby parking. *ParkHere* is the solution to this problem through letting individuals reserve a parking spot for a specified amount of time. *ParkHere* is an mobile application which lets owners lease the parking spots they own to individuals seeking a guaranteed parking spot. Individuals renting a parking spot from owners will be informed about all of the rules and regulations of the parking spot being leased. In addition to user-generated revenue, benefits of *ParkHere* for the parent company include data about parking habits in specific areas of cities.

Architectural Change

Architectural Changes

The project architecture for this application was not changed. The application was created using the Layered Information System, where the application utilizes the Interaction, I/O Management Layer, Resource Management Layer, and the Database Management Layer. The Layered Information System architecture was kept as the choice for *ParkHere* because the application is primarily a transaction-based system. Although Layered Information System sacrifices speed, the tradeoff is necessary in order to preserve the integrity of the private information of the users.

Interaction Layer

Android Studio is used for the Interaction Layer. Displays the interface to the user and all the modifications made to the UI from other users. The interaction layer uses a Google Maps activity which displays the map with all of the listings. Android Studio allows for the flexible creation and management of an Android mobile application

I/O Management Layer

FirebaseUI is used for the I/O Management Layer. This is the layer which handles user authentication and authorization, which will ensure that the private information of the users are secured. The reason FirebaseUI was kept for the I/O Management Layer was because *ParkHere* needs a reliable system for authentication and authorization. Since *ParkHere* handles a large number of transactions, its system architecture must have a higher level of security.

Resource Management Layer

Cloud Functions for Firebase is used for the Resource Management Layer. Using this will allow for the implementation of event handlers in Javascript in the backend. This layer was kept the same because using Google’s Firebase Cloud Functions allows for focusing on implementation rather than worrying about potential scalability issues or the durability of data.

Database Management Layer

Firebase Realtime Database is used for the Database Management Layer. This layer handles transaction management as well as the storage of all user and listing information. There weren’t significant changes to this layer because the application requires good data integrity. Using Firebase Realtime Database is preferred because it allows for the synchronization of data for all users.

Detailed Design Change

Detailed Design Changes

The detailed design of the project has remained mostly unchanged. The application's package design and class design have stayed mostly the same as how it was specified in the design document. The class interaction design has been receiving some small changes in order to adapt it to future implementations.

Package Design

The package uses a layered architecture of Interface, IO Management, Resource Management, and Database Management. The application uses Android Studio for interface management which connects to Firebase. The IO management is controlled by the Firebase UI. Resource Management is handled by the Google API and Firebase Cloud Functions. We will implement the Stripes API for payment functions after the core functionality. Database Management is controlled by Firebase Realtime Database. The reason the current package design is preserved is because when users are submitting private information, the structured layers allow for added security.

Class Design

The design of the classes still closely follows the class diagram design. The only changes to the class diagram includes the removal of the spot class. The reason for removing the spot class was because the listing class was already able to accurately represent what was necessary for listings. The database was structured in a way where all of the listing details were stored within the same child node, because of this the data model class for firebase data retrieval had to be modeled with just one class with the attributes inside the spot class merged with the Listing class. The spot subclasses were also unnecessary because the different types of spots can simply be specified as an attribute in the tuple in the database.

Class Interaction Design

The design of the class interaction has been undergoing small changes. The sequence of calls between classes must change throughout the application creation process. As one implementation method is refactored, more aspects of the design of the functionalities must also be refactored. Some classes have been broken up into multiple activities. For example, the RequestSpot class was turned into BrowseListing activity sequence.

Requirement Change

Functionality of Split-Booking

If *ParkHere* were to support the implementation of split-booking, then the current implementation of booking would require some changes. Currently, the method of booking a listing is selecting the listing from a list and reserving it. In order to integrate this feature into the implementation of booking, the Browse Listing activity sequence would require additional activities where the user chooses the start date, start time, end date, and end time of their desired booking. After the user specifies this information and reserves the booking, the previous listing is deleted and is created with new listings. For example, there is an advertised listing starting on 11/12 at 2:00PM and ending on 11/13 at 10:00AM. If a user books the listing starting on 11/12 at 5:00PM and ending on 11/13 at 1:00AM, then the GUI makes a call to Firebase to delete the original listing and to create two new listings:

1. a listing starting on 11/12 at 2:00PM and ending on 11/12 at 5:00PM
2. a listing starting on 11/13 at 1:00AM and ending on 11/13 at 10:00AM

With this implementation, the user booking the reservation gets the reservation time they want while the owner of the listing is still able to advertise and sell the remaining times for the listing.

Cancellations of Split-Booking

If a user does a split-booking, but later cancels it, then the booking will be remade as the original owner’s listing and merged with the other listings that came from the original listing. For example, if the same user from the previous example cancels their listing, then the booked listing will be remade. Then, the original listing owner will have three listings:

1. a listing starting on 11/12 at 2:00PM and ending on 11/12 at 5:00PM
2. a listing starting on 11/13 at 1:00AM and ending on 11/13 at 10:00AM
3. a listing starting on 11/12 at 5:00PM and ending on 11/13 at 1:00AM

The GUI will call to Firebase and check if the owner user has any listings with matching start and end times. If there are listings with matching start and end times, then the multiple listings are deleted and created as a single listing.