Smart Waiter System Architecture

Meraj Patel #1137491
Pavneet Jauhal #1149311
Shan Perera #1150394

January 10, 2016

Contents

1 Introduction **2**
- 1.1 Purpose . . . 2
- 1.2 Description . . . 2
- 1.3 Scope . . . 2

2 Overview **2**
- 2.1 Design Principals . . . 2
  - 2.1.1 Information Hiding . . . 2
  - 2.1.2 Single Responsibility Principal . . . 2
- 2.2 Document Structure . . . 2
- 2.3 Revision History . . . 2

3 Control Flow Diagram **2**

4 Module Decomposition **3**
- 4.1 Ordering Class Structure . . . 3
  - 4.1.1 MenuCategories Class . . . 3
  - 4.1.2 MenuItems Class . . . 4
  - 4.1.3 User Class . . . 4
- 4.2 Camera Structure . . . 4
  - 4.2.1 Design Principles . . . 5
  - 4.2.2 onActivityResult . . . 5
  - 4.2.3 Module Decomposition . . . 5
- 4.3 Accounts Structure . . . 5
  - 4.3.1 Design Principles . . . 6

5 Traceability Matrix **6**

6 Likely Future Changes **6**
- 6.1 Anticipated Changes . . . 6
- 6.2 Extra Design Features . . . 7

List of Figures

List of Tables

1 Revision History Table . . . 2
2 Trace Between Requirements and Modules . . . 6

1 Introduction

1.1 Purpose

1.2 Description

1.3 Scope

2 Overview

2.1 Design Principals

2.1.1 Information Hiding

2.1.2 Single Responsibility Principal

This principal states every class should have responsibility of a single action within the software architecture. This action shall be encompassed in a class. Doing so will keep each process action separate allowing better organization.

2.2 Document Structure

This document structure is based on the general software design document template. This document provides a general overview of purpose, description and scope of Smart Waiter. It then provides a detailed review of system architecture and data design in decomposed components. Lastly, the document provides project schedule.

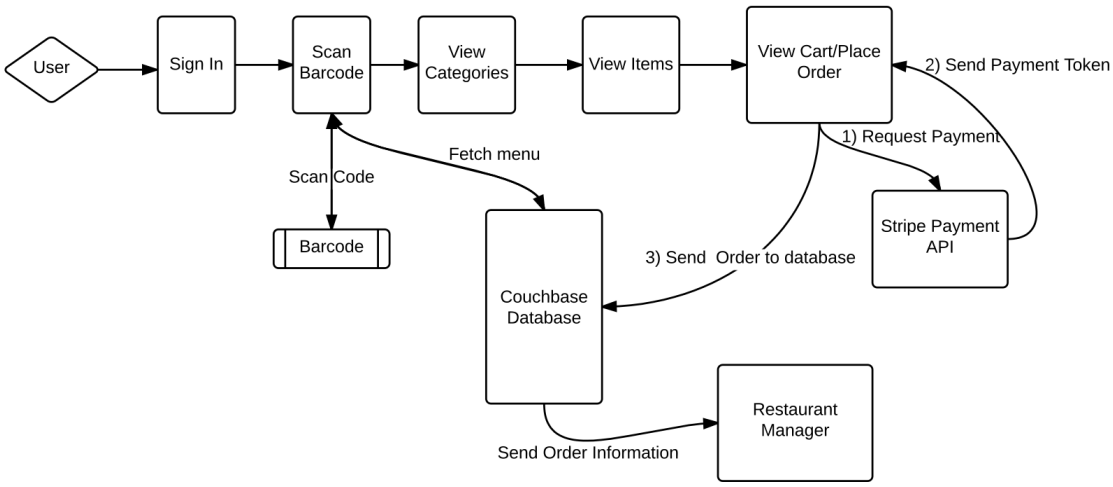
2.3 Revision History

Date	Comments
January 11, 2016	first draft.

Table 1: Revision History Table

3 Control Flow Diagram

Below is a high level overview of Smart Waiter operation. The operation is decomposed into components which is introduced in this section and will be thoroughly explained in the following sections.



4 Module Decomposition

4.1 Ordering Class Structure

There are three primary classes used to hold all vital information regarding menu information. These are: MenuCategories, MenuItems and User. In correspondence with single responsibility principal, each class is encompasses a single purpose. This purpose and how classes correlate with each other is described below.

4.1.1 MenuCategories Class

This class purpose is to store menu category information of a restaurant menu. This entails category name, picture and a reference to category items. To do so, there are three main variables used within this class to hold this information:

categoryName: Stores category name in the form of String

picURL: Stores URL of category picture

categoryItems: Stores an array of objects that reference MenuItems class

This class is instantiated and called upon when a user successfully scans a barcode. The JSON response acquired from the database is parsed, and information related to menu categories is stored appropriately within this class. The application uses this class to display category information when "Menu Categories" page is spawned (please see section 5.1.1 to view picture).

4.1.2 MenuItems Class

This class purpose is to store menu item information of a restaurant menu. This means, item name, price and description. Thee main variables are used within this class to hold this information.

itemName: Stores item name

itemPrice: Stores item price

itemDetail: Stores description of item

Objects of this class are instantiated from the MenuCategories class. By successfully scanning the barcode and parsing the JSON response, MenuItem objects are instantiated and used to save item details. These objects are held within categoryItems list seen in Menu Categories class. This application uses this class to display item information of a category when "Menu Items" page is spawned (please see section 5.1.2 to view picture). These objects are also used in reference to items the user would like to order. This is thoroughly discussed in the proceeding section.

4.1.3 User Class

User class is used to store menu item a user would like to order. For now, there is only one important variable to consider:

userItem: an array that stores objects of MenuItems class. This is used to save menu items a user would like to order.

Only one object is instantiated through the life time of this application. This object represents the user cart of menu items he/she would like to order. This object is called upon when a user decides to add an item to their cart. When this occurs, a copy of the MenuItem object is saved within the userItem variables. This way, there is a list maintained to hold all item a user would like to order. When a user confirms and sends their order, the item information is extracted from userItem list, formatted into a JSON request and send back to the database for further processing. Having this class implemented allows extendibility in the future, as all vital information pertaining to a user can be stored within the class (for example, user settings).

4.2 Camera Structure

There is one main function related to the Camera structure: onActivityResult. The camera is used to read QR codes, which contain the location of where the menu data is stored in relation to the current restaurant. The QR codes are read using an embedded version of the ZXing library. When the application starts, a ZXing specific variable (IntentIntegrator) is initialized, when the user clicks the "Scan QR Code" button on the first screen, the Camera is initialized.

4.2.1 Design Principles

This component follows the Single Responsibility and the Information Hiding principle.

Single Responsibility: The responsibility of this component is to capture and parse QR code data. If this component had to be changed, it would only affect the camera functionality of the application, and not other components like the menu or accounts.

Information Hiding: The camera component is separate from every other component of this application. If any changes were to be made to the camera application, it will not affect the other components of the application. However, it is unlikely that there will be any changes applied to this component of the application.

4.2.2 onActivityResult

This function uses an integrated version of the ZXing library to scan QR codes. After the camera is initialized, the application waits for the user to pass a QR code, this is done by taking a picture of the QR code with the camera. The function onActivityResult is called as soon as a QR code is passed. It takes, as input a requestCode, resultCode, and the current intent:

requestCode: Used to parse the QR code

resultCode: Used to parse the QR code

intent: Gives the current intent, used to parse the QR code

Using the given input, the function parses and gets the contents of the QR code that was passed to the function using the ZXing function parseActivityResult. The information stored in the QR code is then saved to a local variable to be used by the other components of the application.

4.2.3 Module Decomposition

Module: onActivityResult

Secret: How the QR code is parsed and read

Service: Parses and gets the content of the QR code

4.3 Accounts Structure

There are 3 main classes related to the Accounts structure: User, Card, and Stripe. Accounts are used to associate a user identifier for transactions and for storing a user's credit card information securely. Credit card information transfer and storage uses Stripe API. This reduces the burden on the developers, since we can use an established API that follows the restrictions applied by credit card companies, and handles the secure transfer of sensitive information, rather than developing an efficient and secure system ourselves.

4.3.1 Design Principles

This component follows the Single Responsibility and the Information Hiding principle.

Single Responsibility: The responsibility of this component is to securely store a user's credit card information for use in transactions. If this component had to be changed, it would only affect the camera functionality of the application, and not other components like the menu or accounts.

Information Hiding: The camera component is separate from every other component of this application. If any changes were to be made to the camera application, it will not affect the other components of the application. However, it is unlikely that there will be any changes applied to this component of the application.

5 Traceability Matrix

Below describes how our decomposed modules relate to requirements specified in SRS

Req.	**Modules**
R1	Camera Structure
R2	Accounts Structure
R3	Accounts Structure
R6	Ordering Class Structure
R7	Ordering Class Structure
R9	Ordering Class Structure
R13	Accounts Structure
R14	Accounts Structure
R15	Ordering Class Structure
R16	Ordering Class Structure
R18	Ordering Class Structure
R19	Ordering Class Structure

Table 2: Trace Between Requirements and Modules

6 Likely Future Changes

6.1 Anticipated Changes

Database: Consolidation of both databases (for menu items, and orders)

Accounts: Switch to self made account system rather then using Facebook API as it may have compatibility issues

6.2 Extra Design Features

User Preferences: Make more use of User class by providing the user the ability to store preferences. Eg, food allergies, past meals