|  |  |  |
| --- | --- | --- |
| **The Edward S. Rogers Sr. Department of**  **Electrical and Computer Engineering**  **University of Toronto** | | |
| **ECE496Y Design Project Course**  **Group Final Report** | | |
| Title:  Food Place Portal | | |
| Project I.D.#: | 256 | |
| Team members: | Name: | Email: |
| Andy Yi-Chun Huang | andy.huang@mail.utoronto.ca |
| Muhammad Azhar | omar.azhar@mail.utoronto.ca |
|  | Guiqi Wang | elvinqi.wang@mail.utoronto.ca |
|  |  |  |
| Supervisor: | Professor Michael Stumm | |
| Section #: | 1 | |
| Administrator: | Professor Khoman Phang | |
| Submission Date: | March 19th 2015 | |

**Executive Summary (author: Muhammad Azhar)**

Since most students and staff members are busy in the University of Toronto St. George (UTSG) campus, they buy their lunch from food places in the campus. Even with many options available, most people miss out on many of these options due to their busy lifestyle. Whilst services such as Yelp are available that provide information and feedback from other people about some food places, they fail to cover all of the food places in UTSG, and they fail to provide the platform for the food places to send information such as promotions and events to their customers. To solve these issues, our goal is to develop a sustainable food portal for a mobile platform that provides information on all food places in UTSG and allows the dining places to send real-time messages to interested users.

The system must store and display information about each food place in UTSG. It must also add new information and modify/delete existing information upon request. Administrators must be able to create their food place’s profile and send real-time messages to users subscribed to their food place. The system must log in users through an external social media account and only permit them to subscribe to a food place. The system shall be developed for Android mobile devices, which shall be connected to the Internet to use the application. It shall be hosted in an external server for public Internet access. It should complete user requests in little time, handle many requests at the same time, and store some data from the server into the local device to be accessed offline.

The final design consists of a user interface, an admin interface (both on top of an Android Service Development Kit (SDK) framework), a back end Python Bottle server, and a database managed by SQLite. The user/admin interface takes in user inputs through its interactive interface and generates an application programming interface (API) request to send to the server along with a package containing input arguments (if necessary). The server will receive the request and package and parse them. If the request matches one of the known API requests, it will execute the appropriate API and send the result back to the user/admin interface in the form of a JSON object. For Google logins, the Google OAuth service is applied onto the system. For PUSH message management, the Google Cloud Messaging (GCM) service is utilised by the system. Using the Android SDK, SQLite and Bottle, there is little performance, memory, and coding overhead in the system implementation. However, we are limited in how much we can optimise the system. Also, whilst the third-party services are reliable and easily integrated into the system, they are out of our control.

Various system and module-level tests were conducted to test and verify our system. The tests indicate that each module works correctly, and all modules integrate together well in forming the system. After completing this report, we will add in a few more features, conduct benchmark tests to determine the system’s performance, and prepare for the Design Fair on March 28, 29, and 30.

**Group Highlights and Individual Contributions**

**Group Highlights (author: Andy Huang)**

Our design team has made significant progress since the Design Review meeting and the Progress Report. Firstly, we have developed the interface and functionalities of the three individual modules: the client interface, the administrator interface, and the back end server. Secondly, we have interconnected these modules with application programming interface (API) calls implemented using HTTP Requests and JSON responses. Thirdly, we have incorporated various third-party libraries and frameworks into our project in order to enhance its functionalities.

For the client application, we have designed and developed the user interface and the underlying program services. In addition, we have successfully integrated Google Maps, Google OAuth login and Google Cloud Messaging (PUSH notifications) into our front end client software. As a result, the users are able to login with their Google accounts to check for dining information such as location, menu item, price, and hours of operations, to submit restaurant reviews or feedbacks, and to receive messages about promotional events through the use of our Android mobile application.

For the administrator application, we have designed and developed the user interface and the underlying program services. As a result, the restaurant vendors can create their own profile page, upload images of the restaurant or food items, and communicate with the customers using a real-time multicast messaging system.

For the server program, we have developed a Python back end program that utilizes the Bottle framework as well as several third-party libraries such as "google-api-python-client" and "python-gcm" to assist us with the integration with Google services. Furthermore, we have developed a database in SQLite and the tables that enable efficient data transactions. Finally, we have deployed all of the server components onto Amazon Web Services (AWS). As a result, the client or administrator applications can send data requests such as read, write, update or delete from anywhere in the public Internet domain and receive a response according to the requests.

**Individual Contributions: Andy Huang (author: Andy Huang)**

My main contribution to this project is the design and development of the client user interface on the Android mobile platform. I started off the project by designing the user interface (UI) elements, including their images, sizes and positions on each screen, and I programmed these screen mock-ups in the Android XML layout files. In the following step, I made the application interactive by programming the underlying logic of these UI elements and connecting the various screens with button clicks. Furthermore, I integrated the Android Google Maps package into the application and plotted restaurant locations as markers on the map. I also implemented the OAuth authentication protocol to enable users to login with their Google accounts and the receiver for Google Cloud Messaging to enable users to receive PUSH notifications from our back end server. In the final step, I connected the client module to the server module by setting up the sending of HTTP Requests and the parsing of JSON responses on the mobile application. All of these efforts resulted in the completion of the "end-user mobile application" module of our system.

In addition to my contributions to the end-user mobile application, I also contributed to the SQLite database and the back end Python server modules of the system. Due to the absence of Hailin (who dropped out shortly after project kickoff), Muhammad and I had to work together to take on Hailin's original responsibility of developing and managing the SQLite database module. As a result, I developed the initial database schema with 3 tables that holds restaurant data such as name, location, menu items, item price and reviews. From there, Muhammad took over the database module to expand and integrate with the server program. For the back end Python server module, I incorporated two third-party libraries: “google-api-python-client” and “python-gcm”. I used the first library to implement the server logic of the OAuth authentication protocol in order to complete the end-to-end transaction for the users to login to our mobile application with their Google accounts. I used the second library to implement the server logic to send PUSH notifications to the mobile application.

Overall, my main focus in this project is the development of the end-user mobile application module and I was able to successfully complete the associated tasks. However, aside from my main responsibility, I also made significant contributions in the early stages of database development as well as in the programming of two back end server features.

**Individual Contributions: Muhammad Azhar (author: Muhammad Azhar)**

My main contributions to this project involve the design and development of the server and the continued design and development of the database. Initially, I designed and developed API’s associated with simple, generic requests. They returned a simple string message to the client, a JSON object with a message, or a JSON object with an array of predetermined data. These API’s set the template for JSON object creation and formatting for later API’s.

After discussing with Andy about the API’s he wanted for the user client interface, I built some API’s that fetched all of the map points, menu items of a food place, and reviews of a food place. To accomplish those requests, I integrated the database with the server through SQLite in each of the new API’s. Later on, I designed and developed the rest of the API’s needed for fetching the details of a food place, review submissions, adding subscriptions, and removing subscriptions for a given food place.

I did the same with Guiqi (Elvin) and designed and developed API’s that allowed Elvin to register a new administrator and create a new food place, login the administrator, add/modify the details and images of the food place, add/modify/remove the menu items of the food place, and send a PUSH message to a food place’s subscribers. However, for the admin interface, I also designed the requests that the admin interface needed to send for each API, specifying the name (HTTP Request to send) and the specific parameters that were required for it.

After Andy developed the initial database schema along with sample tables, I expanded upon that database by adding in more tables as well as integrating it with the server through SQLite. Through discussions with both teammates over the information the server should interact with, I created new tables that stored the details of each food place, the subscribed users to each food place, and the login credentials of each administrator account. I also added sample data for the new tables through both a database manager program and the server itself, which was used in the testing and verification of the food place hub.

In summary, my main responsibility was the design and development of the server, and I successfully completed the tasks relating to this. Also, I expanded, refined, and integrated the database with the server. In the end, I deployed the complete server and database onto AWS by storing the appropriate files into the Amazon server, starting a virtual instance of the server, and running the server on that instance. This allows for the application to connect to the server from anywhere through the public Internet.

**Individual Contributions: Guiqi Wang (author: Guiqi Wang)**

My responsibilities in this project were to design and test the "administrator interface" part of our system. Specifically, I was solely responsible for the following tasks:

* Configuring the network socket and parsing system of the admin user interface
* Establishing the starter admin user interface
* Implementing the complete admin user interface

All of my tasks were finished on time. I started from knowing little about Android development. In order to design and implement an administrator user interface, lots of reading of the Android development tutorial was accomplished. Throughout the process of doing the project, I learned and used Java and XML for managing the Android interface layout. This enabled me to accomplish the logic part of the main Android interface, which allowed the admin to login and parse information for sending. I also learned the HTTP networking protocol for setting up a connection with the server and exchanging data between the Android client and server. Furthermore, tests cases for each module were set up, which showed that the admin user interface was able to function properly.

All my tasks were assigned after the initial team discussion. Due to one of our member’s absence for this project, our schedule was disrupted. Response parsing tasks were decided to be incorporated into each member's module of responsibility. I explained the main structure of the admin user interface module to my group members, thus the team possessed a better understanding of my work. In addition, the server API’s were built according to our team discussions. In the system tests, because two parts were developed individually by Muhammad and me, we have to cooperate in the debugging process frequently. In order to improve efficiency, Skype meetings were regularly held during bug fixing. All of my tasks were finished according to the requirements and decisions from our team, so that our team could be able to make progress on time.

**Acknowledgements**

Our design team would like to acknowledge the support we have received from both our administrator and supervisor throughout the project development process. We would like to thank our supervisor, Professor Michael Stumm, for his advice in project scoping, designing attractive software features, and the course deliverables. Furthermore, we would like to thank our administrator, Professor Khoman Phang, for his helpful feedback with regards to the direction of our project in the initial stages, the project documentations as well as the oral presentation.

**Table of Contents**

**Section 1: The U of T Food Place Portal** ……………………………………………………………….. 1

Section 1.1: Introduction ………………………………………………………………………….. 3

Section 1.2: Background and Motivation …………………………………………………………. 4

Section 1.3: Project Goal ………………………………………………………………………….. 4

Section 1.4: Project Requirements ………………………………………………………………… 4

Section 1.4.1: Functions …………………………………………………………………... 4

Section 1.4.2: Constraints …………………………………………………………………. 4

Section 1.4.3: Objectives ………………………………………………………………….. 4

**Section 2: Final Design** …………………………………………………………………………………... 3

Section 2.1: System-Level Overview ............................................................................................... 3

Section 2.2: Module Level Descriptions and Design ....................................................................... 4

Section 2.2.1: End User Mobile Application ....................................................................... 4

Section 2.2.2: Administrator Interface ................................................................................. 4

Section 2.2.3: Server Program ............................................................................................. 4

Section 2.2.4: SQLite Database ........................................................................................... 4

Section 2.3: Assessment of Final Design ......................................................................................... 4

Section 2.3.1: Strengths ………………………………………………………………….... 4

Section 2.3.2: Weaknesses ………………………………………………………………… 4

Section 2.3.3: Trade-Offs ………………………………………………………………….. 4

**Section 3: Testing and Verification** ............................................................................................................ 3

**Section 4: Summary and Conclusions** ........................................................................................................ 3

**List of References** .........................................................................................................................................5

**Appendix A: Gantt Chart History** .............................................................................................................6

**Appendix B: Original Validation and Acceptance Tests** .........................................................................8

**Appendix C: Screen Capture of System Level Tests** ................................................................................8

**Appendix D: Database Query Test Cases** ..................................................................................................8

**1. The U of T Food Place Portal**

**1.1 Introduction (author: Andy Huang)**

This report summarizes the motivation, design, implementation, and testing of a food place portal developed on the Android mobile platform as part of our final year design project course ECE496. The report concludes with suggestions of improvements and future work.

**1.2 Background and Motivation (author: Andy Huang and Guiqi Wang)**

According to statistics from the University of Toronto, there are around 58,000 students and 12,000 faculty staff members in the St. George campus [1]. Due to their busy schedules, most of them dine on-campus. Although several bring homemade food to eat, most still buy their lunch from various dining locations in and around the University of Toronto St. George (UTSG) campus. According to the UTSG Food and Meal Services directory, there are more than 100 dining places on campus that offer a wide range of meals [2]. However, due to their busy schedules, most people miss out on many of these options.

Nowadays, there are lots of useful portals where people can search for adequate dining locations. For example, Yelp provides information on various dining locations. Yelp users can search for their favourite food and restaurants by simply entering keywords or using the filter options. However, it only provides information on 15 dining places on campus [3] compared to the U of T Food and Services website, which covers 35 dining places [4]. Also, Yelp’s search engine is not comprehensive as inputting one keyword into the search engine cannot provide users with a complete list of relevant dining places [5]. On the other hand, the university provides menu and hours of operations of many dining locations through websites like UeaT and CampusDish [6]. However, it disregards places such as food trucks and hotdog stands. In addition, all these food portals are missing a two-way communication channel for the customers and the businesses to connect with each other. While customers can often leave reviews or feedbacks for the businesses, the businesses often cannot deliver real-time information such as promotional events and address multiple users effectively using a one-to-many messaging model.

In order to provide better assistance to students and staff members in searching for suitable dining places as well as to enable efficient communication, it is necessary for our U of T Dining Portal to facilitate a service whereby students and staff members can check for dining information through the portal and to provide a multicast messaging system. These services will not only enable users to determine where to find their favourite food and meals on campus, but also promote the less popular dining places to a wider range of students and staff on the UTSG campus.

**1.3 Project Goal (author: Andy Huang)**

The goal of our project is to provide a mobile platform for a sustainable food portal that exposes the dining options on campus and helps the users find their favourite food and meals as well as to receive real-time communication of certain events from their subscribed food locations.

**1.4 Project Requirements (author: Guiqi Wang)**

**1.4.1 Functions**

1. The system must store the information such as the name, description, address, working hours, and menu of the food places inside and around U of T, St. George campus.
2. The system must be able to add new or update/delete existing information in the food portal.
3. The system must display information about the dining locations upon users' requests.
4. The system must allow for an administrative entity to create a profile page, upload initial food place information, and subsequently update these/add more details.
5. The system must allow the administrators to send real-time information to the users who have subscribed to that food place.
6. The system must integrate with social media and allow the users to login to the mobile application with an external account.
   * The system must allow logged-in users to subscribe to a food place.

**1.4.2 Constraints**

1. The design shall be developed for mobile devices running the Android operating system.
2. The design shall require Internet connection on either mobile data or Wi-Fi.
3. The design shall be hosted on the public Internet domain to be accessible by the users.
4. The design shall only allow users to log in with a Google account.

**1.4.3 Objectives**

Performance

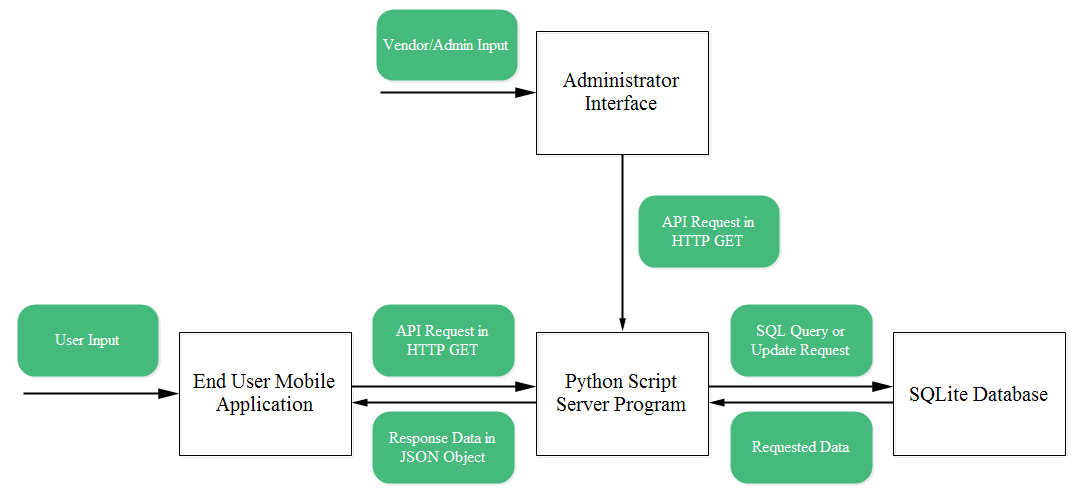
1. The response time of the software should be within 1 second [7].
2. The design should be backwards compatible up to Android 3.1 [8].
3. The design should allow for at least 100 requests to be handled at the same time.
4. The design should cache some data locally to allow user access without the Internet.

User Experience

1. The number of pages to navigate through to obtain the desired information should not be more than 4 pages.
2. The design should have a uniform font type that is readable for everyone.
3. The design should provide security data encryption for administrator login.

**2. Final Design**

**2.1 System-level Overview (author: Andy Huang)**Figure 1: System Block Diagram

****

End users (students and staff members) will first provide input using our mobile application‘s interactive user interface. Upon receiving an input, the application program will generate an application programming interface (API) request. These requests will be constructed in a HTTP URL format and sent to the server. Our server program will parse the client requests and construct query commands using the structured query language (SQL). The connection between a server and database is established via a connection object. Once a connection is established, our server program will query the database, which will return with the requested data that can be navigated via a cursor object. The server program will bundle the raw data into a structured object using JavaScript Object Notation (JSON). The JSON object will be passed back to the mobile application where it will be parsed and displayed on a user interface to the end users. In addition, we also have an administrator interface for the dining place vendors. This system will take data update or creation requests from an administrator as input and generate an API request to the server. The server program will parse the request and issue a SQL command to update the data in the database accordingly.

**2.2 Module-level Descriptions and Design**

**2.2.1 End User Mobile Application (author: Andy Huang)**

Components:

* User interface
* Google Map
* Google Cloud Messaging Receiver
* Logic Process Center
* Network Socket

Input:

* User requests to retrieve data through interactive interface
* Server response of requested data in JSON format

Output:

* API request to server to get the corresponding data
* Requested data shown on the graphical user interface

Functionality:

The client interface application handles the request from the user and generates the corresponding API to the server to request for the desired data. The client then accepts the corresponding data from the server and transforms the data into a readable format on the application screen for users. In addition, the client also receives PUSH notifications sent by the restaurant vendors and initiates the OAuth authentication protocol to enable users to login with their Google accounts.

**2.2.2 Administrator Interface (author: Guiqi Wang)**

Components:

* User interface
* Logic Process Center
* Network Socket

Input:

* User request to update data through interactive interface
* Server acknowledgment message

Output:

* API request to server to update the corresponding data
* Transaction status (success/fail)

Functionality:

The administrator interface handles updating and adding requests and generates the corresponding API to the server to update existing or add new data into the database. The client will accept the corresponding transaction status message from the server and display it on the interface. In addition, the admin interface will also forward the PUSH notifications to the back end server, which will be sent to the subscribed users.

**2.2.3 Server Program (author: Muhammad Azhar)**

Components:

* Bottle: server framework for Python servers that will handle connections and sessions between users and the server
* SQLite: access the database and transfer SQL messages and resulting tables between the server and database
* JSON: Organizes and encapsulates the requested data to be sent to the client
* Amazon Web Services (AWS): cloud server that will host the server program
* Google Cloud Messaging (GCM): service that will allow devices to send and receive PUSH messages
* OAuth 2.0: service that will allow users to log in to the system with their Google account

Input:

* Client request to
  + fetch data
  + add, remove, or modify data
  + register and log in a client to the system
  + send a PUSH message to specific devices

Output:

* Acknowledgement message accompanied by the requested data (if required)
* Error message if the server fails to execute the client request

Functionality:

The server handles HTTP POST and GET requests sent by the application. The POST requests consist of a message describing the request along with a form containing the input arguments for the request. The GET requests consist solely of a message with some input arguments if needed. If the server cannot realise the request, it sends a HTTP 404 (Not Found) message back to the application. Otherwise, the API pertaining to that request is executed.

Most API’s require some interaction with the database, which are performed through SQLite (see Section 2.2.4 for more details). Upon a successful execution of a SQL request, the server will create a JSON object with relevant data and/or a return value (to indicate the result of the operation) to send back to the application. This object and a HTTP 200 (OK) message is sent to the client upon a success.

For logging in users to the system through OAuth, the server acts as a middle man between the application and the Google server. It will pass on information between the application and Google servers for a user authorization, which is completed at the application end.

For PUSH message transmission, the server will store the device ID’s pertaining to corresponding users into memory in the form of a dictionary. By storing them into memory, the PUSH messaging API will not need to fetch the ID’s from the disk, therefore decreasing the execution latency. When the food place admin sends a request to transmit a message to its subscribers, the PUSH messaging API will retrieve the device ID’s for the subscribers and pass them onto the GCM service, which will send the message to the particular users’ devices.

**2.2.4 SQLite Database (author: Andy Huang)**

Components:

* A SQLite database file with the table schemas and the data type designed and developed prior to the application launch

Input:

* SQL queries to get the food information that is of interest
* SQL queries to manipulate or add food information data into the database

Output:

* The food information data that satisfy the query conditions
* The updated database due to the SQL update/insert/delete requests

Functionality:

The database stores and provides information such as a dining location's name, description, images, hours of operation, address, menu items and price. The database handles both the query and the data manipulation requests from the server. In the case of querying, the database will get the data that satisfy the query and return this information to the server. In the case of data manipulation, the database will add or update information depending on the type of the SQL manipulation request.

**2.3 Assessment of Final Design (author: Muhammad Azhar)**

2.3.1 Strengths

* Bottle has little overhead for the server setup and features development, which makes the development of new features and debugging easier.
* Only one file is needed to run the Bottle framework, resulting in a low memory and CPU overhead for the framework.
* SQLite is integrated into Python, so there is little coding, memory, and CPU overhead for using SQLite to manage the database [9].
  + No separate server process needed to manage the database
  + Can access the database through standard SQL requests
* The third-party services (GCM, OAuth, AWS) are reliable in their operation and easy to integrate into the system.
* The Android SDK allows for the third-party Google services and API’s to easily integrate with the application.
* AWS’s virtual machine instance allows for the server program to perform optimally without interference from programs and processes outside of this instance.
  + Server program’s latency and the number of requests it can process at the same time will not be impacted
* PUSH messages and food place locations are cached into the app.
  + Requests do not need to be sent to reload the same information.
  + App latency is reduced as a result.

2.3.2 Weaknesses

* We have no control over the operation of third-party services
  + If these services are down, then the system will not run properly
* The performance of the system is limited by the frameworks over which the system is implemented on.
* Since some information is cached into the application, more of the device’s memory will be used up to store that data
* Without Internet access, food place details and PUSH messages cannot be received.
* Using Google’s OAuth service limits the potential number of users who can log into the system compared to users using their Facebook account to log in.

2.3.3 Trade-Offs

Table 1. Trade-offs of the components, functionalities and features of this system

|  |  |  |
| --- | --- | --- |
| **Component/Functionality/Feature** | **Gains** | **Losses** |
| Third-party services | * Reliable * Easily integrated into the system * Optimal performance of the server program | * No control over the services |
| Google OAuth 2.0 | * Reliable, secure handling of user logins | * May miss out on those users who do not use/have no Google account |
| Android SDK | * Google services and API’s are easily integrated into the application * Little to no overhead is present in implementing and running the app. | * The app’s performance is limited by the Android SDK framework. |
| Bottle | * Little to no overhead is present in implementing and running the server program. | * The server program’s performance is limited by the Bottle framework. |
| Caching of map points and PUSH messages only | * Map points and PUSH messages already in the app do not need to be fetched from the server again. * Food place details will never be outdated as their details change often. | * Fetching food place details already obtained before will result in unnecessary latency * Map points may be outdated |

**3. Testing and Verification (author: Andy Huang, Guiqi Wang)**

For the testing and verification of our project, we will be building a checklist based on the functions and objectives listed in the "Project Requirements" section above. For each item on the list, we will create a separate test case with a script of manual actions (i.e. button clicks). Finally, we will be following the script to interact with our mobile application and present the screenshots of the results (in Appendix C). Please note that most test cases involve multiple modules of our system, and these tests can be considered as system-level tests.

Table 2. Test cases, their results, and criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Description of the Test Case | Modules Involved | Compliance?  (Pass/Fail) | Manual Script Actions |
| 1 | Information storage in database (function #1) | Database | Pass | **1)** Use SQLite Manager via Firefox plugin to inspect the contents of the database file and run the query test cases listed in Appendix D |
| 2 | Add new information by administrators (function #2) | Admin  +  Server | Partial Pass  See note 1 | **1)** Launch the admin application  **2)** Fill in the information on the screen  **3)** Click the "Signup" button  **4)** See data sending on the application screen as well as data received by the test server |
| 3 | Update existing information by administrators (function #2) | Admin + Server | Partial Pass  See note 1 | **1)** Launch the admin application  **2)** Input a credential and sign in  **3)** Press "Business" button  **4)** Fill in the data to update  **5)** Press update button |
| 4 | Submit restaurant review by users (function #2) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is resolved) **2)** Click on the map button from the home screen **3)** Click on one of the map markers  **4)** Click on the information window to see restaurant details  **5)** Click on "See Restaurant Reviews" button  **6)** Click on "Submit Your Review"  **7)** Input some review information and click "Submit" button |
| 5 | Add a new menu item by users (function #2) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is resolved) **2)** Click on the map button from the home screen **3)** Click on one of the map markers  **4)** Click on the information window to see restaurant details  **5)** Click on "See Menu" button  **6)** Click on "Add an Item"  **7)** Input an item name/price and click "Submit" button |
| 6 | Update existing restaurant detail information by users (function #2) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is resolved) **2)** Click on the map button from the home screen **3)** Click on one of the map markers  **4)** Click on the information window to see restaurant details  **5)** Click on the edit icon beside the information to update, input the new information and click "Submit" |
| 7 | Update existing menu item information by users (function #2) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is resolved) **2)** Click on the map button from the home screen **3)** Click on one of the map markers  **4)** Click on the information window to see restaurant details  **5)** Click on "See Menu" button  **6)** Click on edit icon beside the item, input the new information and click "Submit" |
| 8 | Display information of dining locations (function #3) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is resolved) **2)** Click on the map button from the home screen **3)** Click on one of the markers to expand it  **4)** Click on the information window to see restaurant details  **5)** Click on "see menu" or "see restaurant reviews" to explore further information of the place |
| 9 | Create a profile page and upload initial restaurant information by administrators (function #4) | Admin + Server | Partial Pass  See note 1 | **1)** Launch the admin application  **2)** Input a credential and sign in  **3)** Press "Personal" button  **4)** Fill in the data to update  **5)** Press update button |
| 10 | PUSH notification from administrator to subscribed users (function #5) | Admin + Client +  Server | Partial Pass  See note 2 | **1)** Launch the user application  (sign-in status is resolved) **2)** Click on the social button from the home screen  **3)** The button click will invoke the server to send a PUSH notification  **4)** Receive the notification and check the notification inbox |
| 11 | Integration with Google to allow login via Google Account (function #6) | Client +  Server | Pass | **1)** Launch the user application (sign-in status is not resolved)  **2)** Click "Sign-in with Google"  **3)** Enter Google Account email/password **4)** Click accept to authorize "U of T Dining" to obtain certain profile information from Google  **5)** Observe a popup dialog welcoming the signed-in user |

**Note 1:** Elvin (Guiqi) was not able to complete the network connection part of the administrator interface with the Python server module. As a result, he tested the request/response via a third-party test server to verify the data transmission.

**Note 2:** As a result of the connection failure between the administrator interface and the Python server, Andy has created a test button for sending PUSH notifications on the client application. Upon the button click, the python server will be invoked and send out a PUSH notification to the client users.

**4. Summary and Conclusions**

**List of References**

[1] University of Toronto, “Quick Facts” [Online] Accessed September 29, 2014. Available:

<http://www.utoronto.ca/about-uoft/quickfacts>

[2] University of Toronto, “University of Toronto map” [Online] Accessed September 15, 2014. Available:<http://map.utoronto.ca/>

[3]Yelp Canada, “Food Search result 1” [Online] Accessed September 28, 2014. Available: http://www.yelp.ca/search?find\_desc=take+out&find\_loc=univerisity+of+toronto&ns=1#find\_desc

[4]Yelp Canada, “Food Search result 2” [Online] Accessed September 28, 2014. Available: http://www.yelp.ca/search?find\_desc=take+out&find\_loc=univerisity+of+toronto&ns=1#find\_desc=food

[5] Yelp Canada, “Snack Search result 2” [Online] Accessed September 28, 2014. Available: http://www.yelp.ca/biz/ken-hos-snack-service-toronto

[6]Campusdish, “Dining with University of Toronto” [Online] Accessed September 28, 2014. Available: <http://www.campusdish.com/en-US/CA/TorontoStGeorge/Locations/>

[7] Jakob Nielsen, “Website Response Times”, June 21, 2010. Available:<http://www.nngroup.com/articles/website-response-times/>

[8] Google, “Google Maps Android API v2 - Getting Started” [Online] Accessed October 20, 2014. Available:<https://developers.google.com/maps/documentation/android/start>

[9] F.L. Drake, Jr. (2015, March 16), “sqlite3 — DB-API 2.0 interface for SQLite databases” [Online] Accessed March 18, 2015. Available: <https://docs.python.org/2/library/sqlite3.html>

**Appendix A: Gantt Chart History (author: Muhammad Azhar)**

Figure 1: Finalized Work Breakdown Structure and Gantt Chart

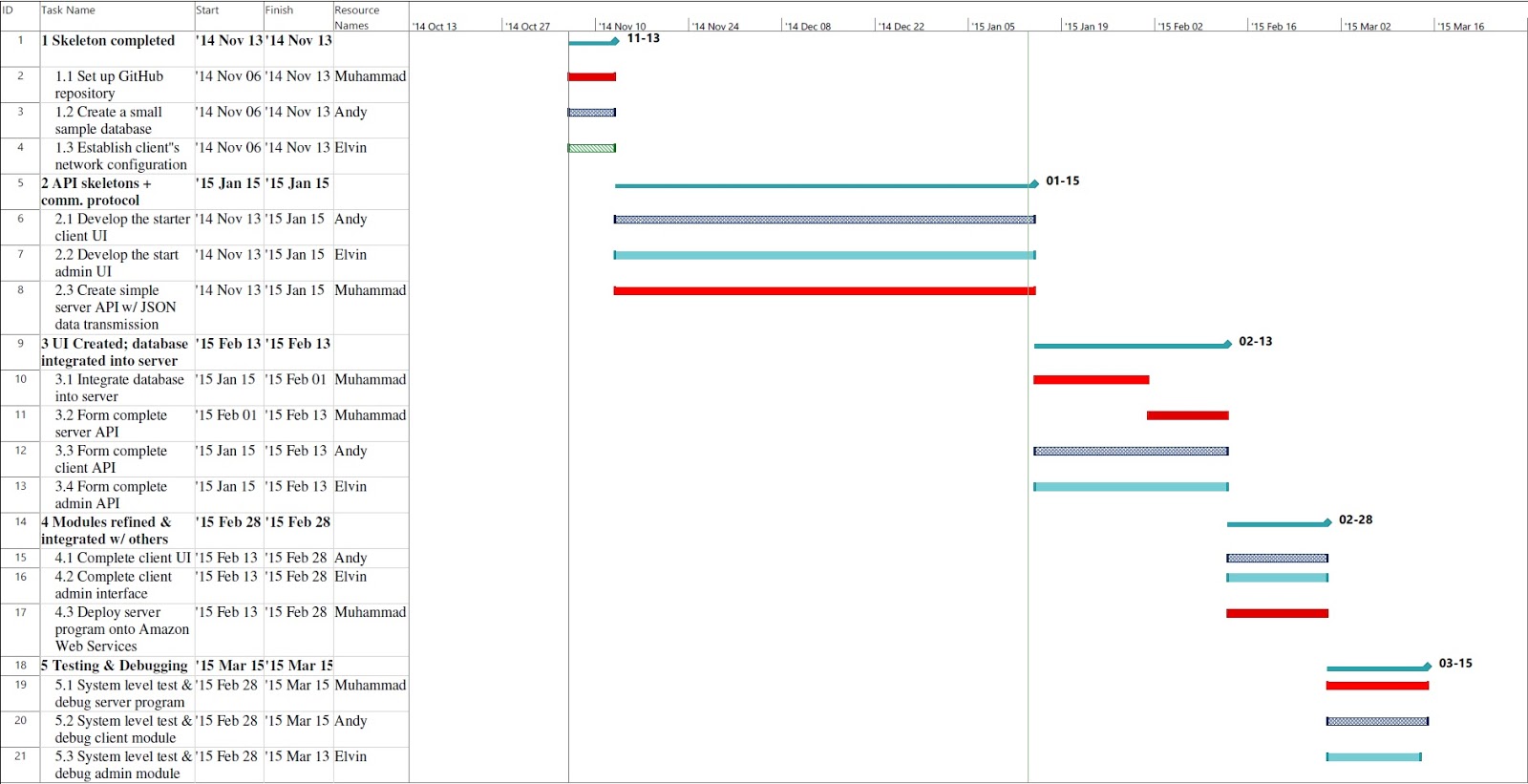
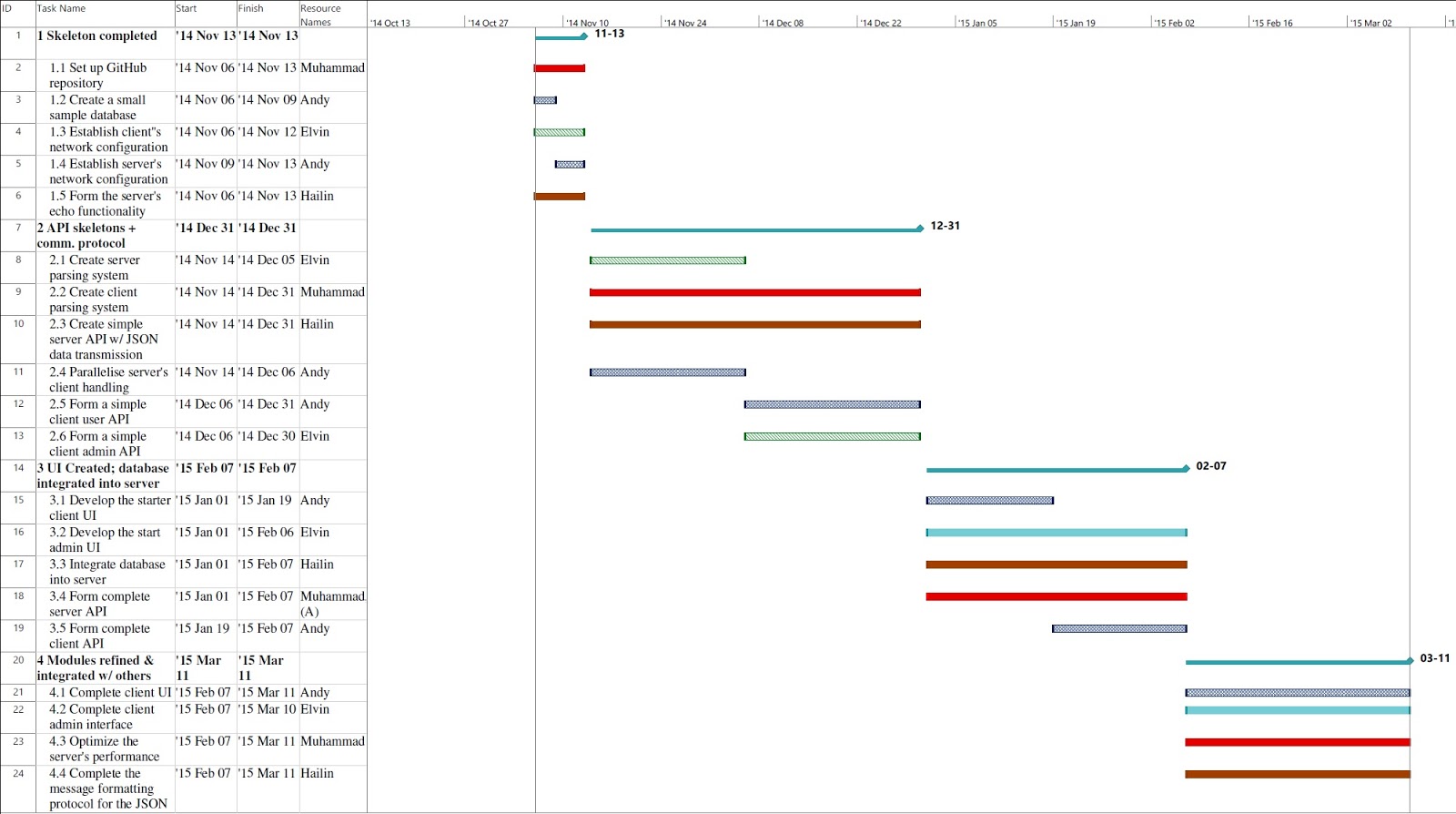
****

Figure 2: Original Work Breakdown Structure and Gantt Chart

****

**Appendix B: Original Validation and Acceptance Test (author: Entire Group)**

Table 3. Description and success criteria of module-level tests for each module of the system

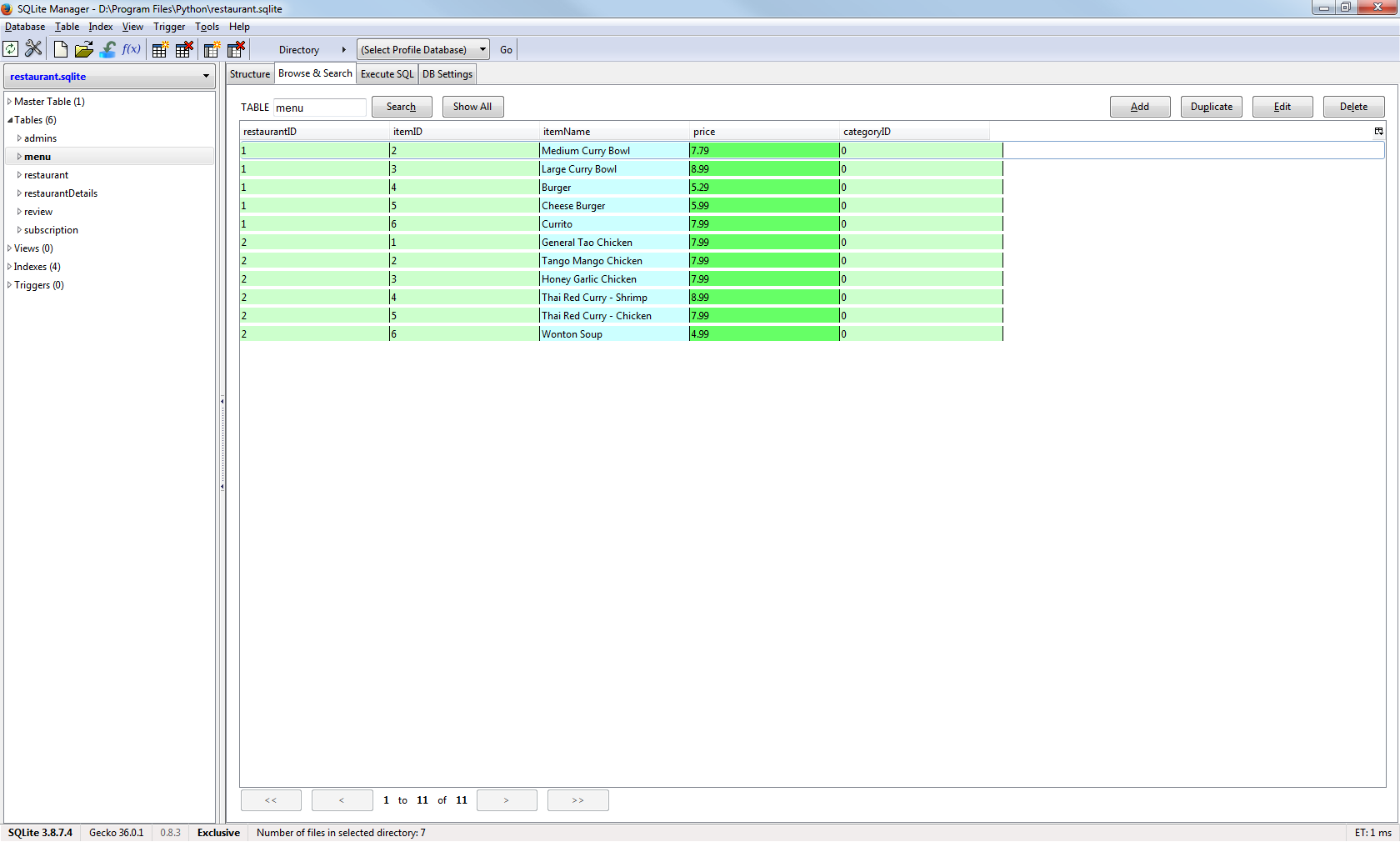
|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **Validation Tests** | **Test Procedure** | **Success Criteria** |
| Database | Data request | 1. Send queries to the database that request for certain data. | The output should match our manual extraction of that data from the database. |
| Data insertion | 1. Send queries that insert data into the database. 2. Request the database for the inserted data. | The database should output the newly inserted data. |
| Data update | 1. Send queries that edit certain existing data in the database. 2. Request the database for the edited data via a query. | The database should output the changed data. |
| Data deletion | 1. Send queries that delete certain existing data in the database. 2. Request the database for the deleted data via a query. | The database should not output anything in response to the request. |
| End user mobile application | Navigation through the application pages | 1. Press the buttons on the app. | The new page should be the page that the button described as the destination. |
| Obtain food place details from the map | 1. Press the markers on the map. | The map should pop out sample information on the marker on the map. |
| Comment and rate on food places | 1. Type in a comment and rate the food place at its page. | The application should display the written comment and rating. |
| Administrator interface | Display typed characters on the login dialog boxes | 1. Type characters into the username and password dialog boxes. | The username box should show the exact characters typed into it, and the password box should show asterisks for each character typed into the box. |
| Login button response | 1. Press the login button without any input into the dialog boxes. | The page should indicate that the incorrect username and password were inputted. |
| Server | Fixed output test | 1. Send a URL to the localhost through a web browser for the fixed output. | The browser should display the correct fixed output. |
| Echo test | 1. Send a URL to the localhost through a web browser with some input to be echoed back. | The browser should display the echoed input along with some fixed characters. |
| JSON output test | 1. Send a URL to the local through a web browser for obtaining a sample JSON array. | The browser should display the sample JSON array. |

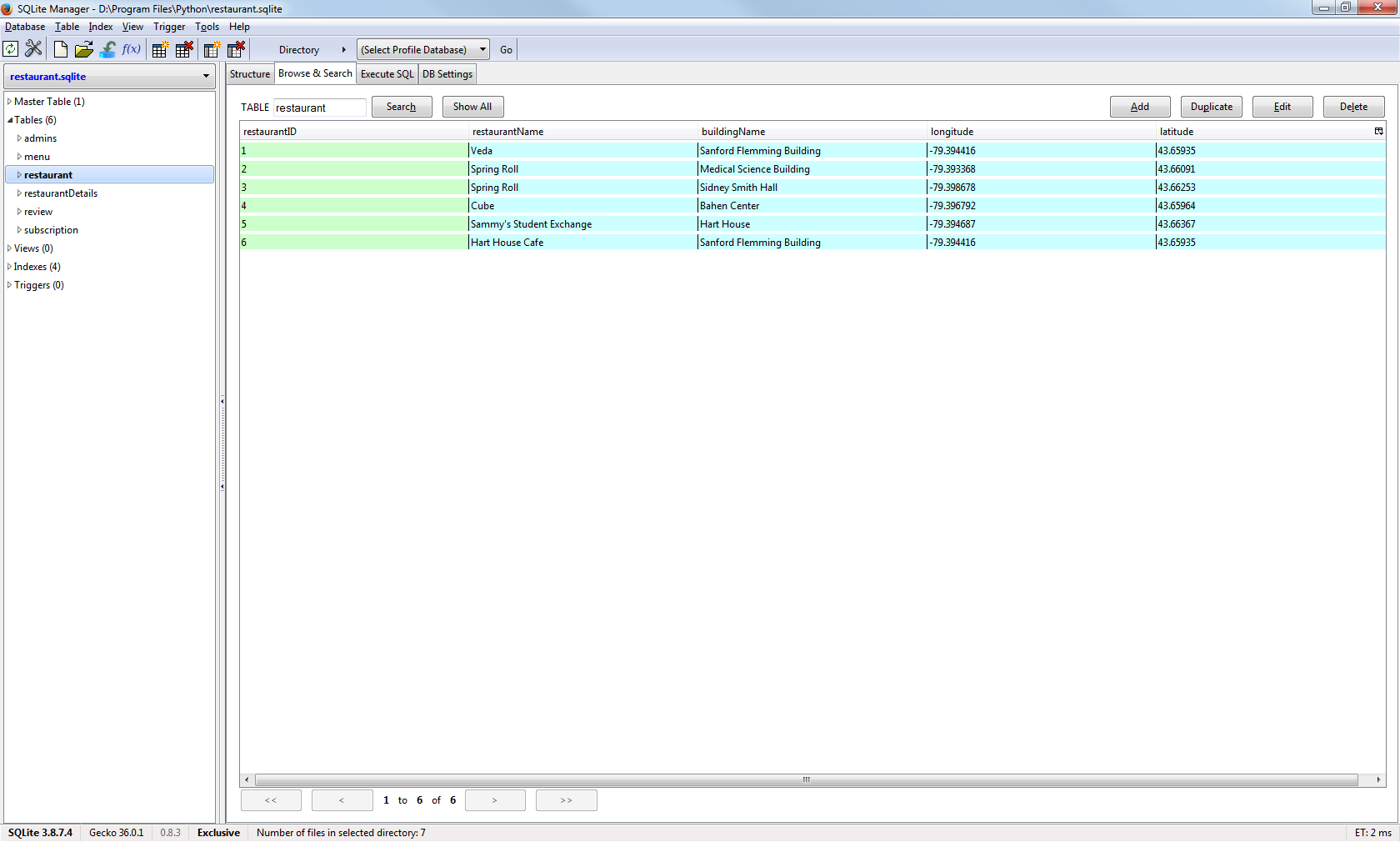
Table 4. Description and criteria of system-level tests for the food place hub.

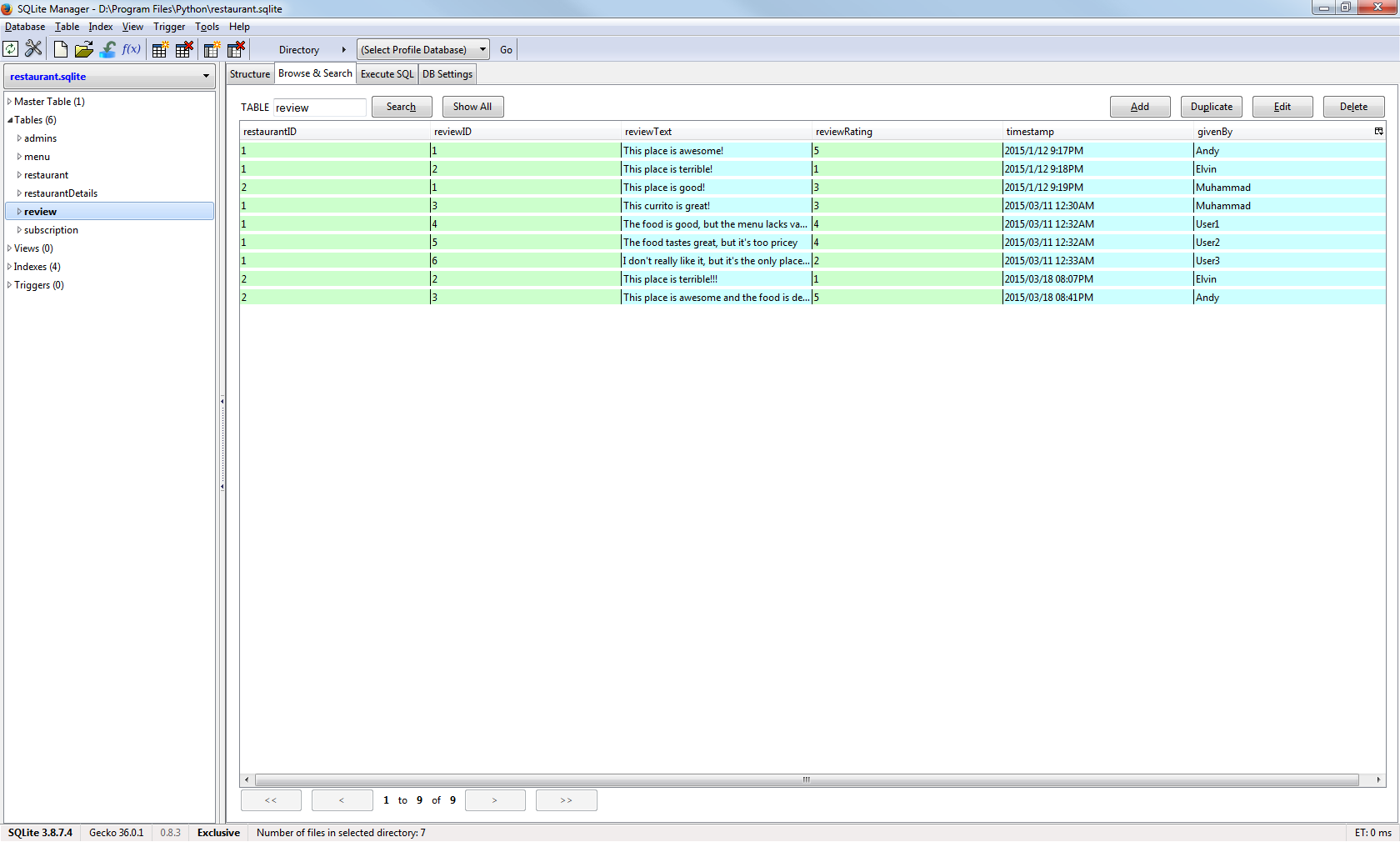
|  |  |  |
| --- | --- | --- |
| **Validation Tests** | **Test Procedure** | **Success Criteria** |
| Request for data through the server. | 1. Send a query from the server to the database requesting for certain sample data. | The server should output the data in the form of a table, and the output should match the data present in the database. |
| Insert data through the server. | 1. Send a query from the server to the database with new data to insert. 2. Send a query directly through the database for the new data. | The database should directly output the new data. |
| Edit data through the server. | 1. Send a query from the server to the database with new data to replace some existing data. 2. Send a query directly through the database for the edited data. | The database should directly output the new data that replaced the previous data. |
| Delete data through the server. | 1. Send a query to the database with the data to delete. 2. Send a query directly through the database for the deleted data. 3. Send a query from the server to the database for the deleted data. | The database should not output anything itself, and the server’s cursor object should have its value set to NULL. |
| Server processing client interface requests | 1. Press the buttons on the app that provide more information on sample food places. | The server should return sample output in the form of JSON objects. The application should display that data in a table. |
| Server processing admin interface requests | 1. Type up a sample username and password (correct ones) into the dialog boxes, and press the login button to send them to the server. 2. Type up a sample username and password (incorrect ones), and press the login button to send them to the server. | The server should output a message on the interface indicating that the login was successful for the correct inputs. For the incorrect inputs, the server should output a message indicating that the username and/or password were incorrect. |

**Appendix C: Screen Capture of System Level Tests (author: Andy Huang)**

**Test Case 1 (SQLite database):**

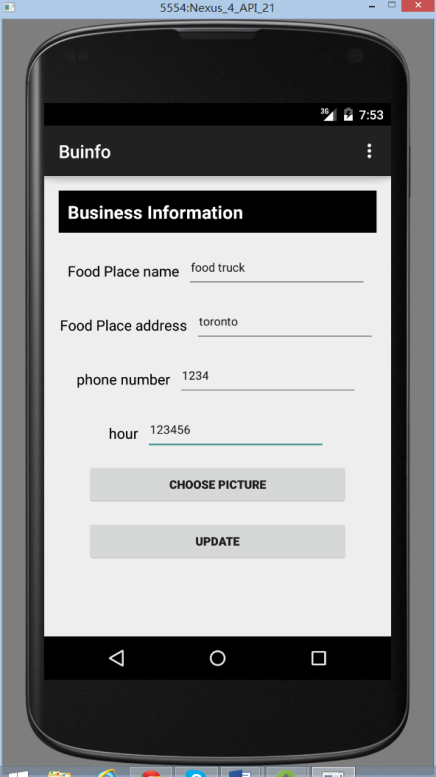
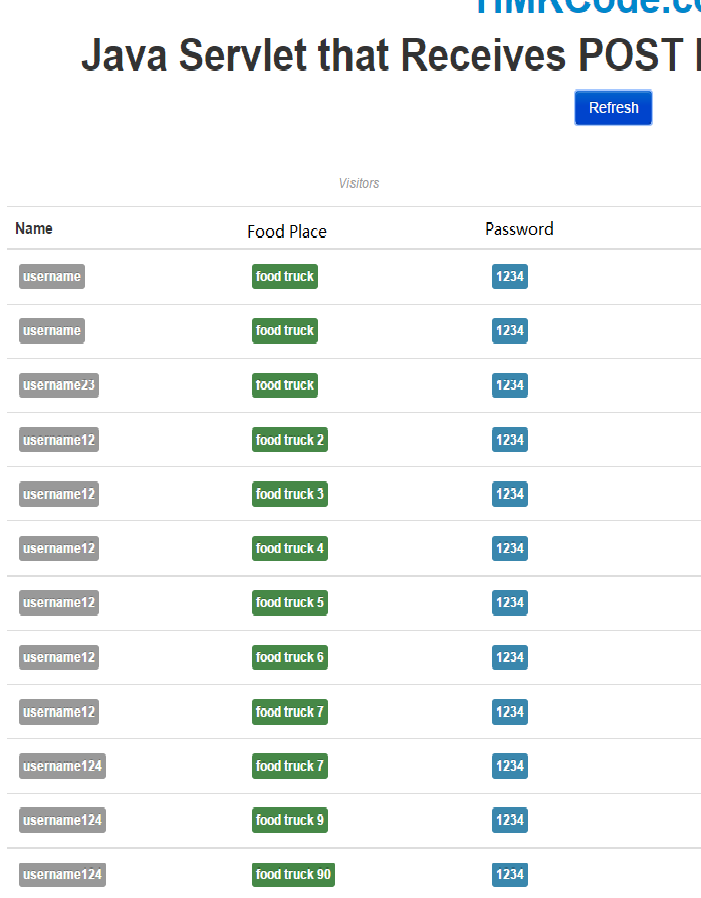
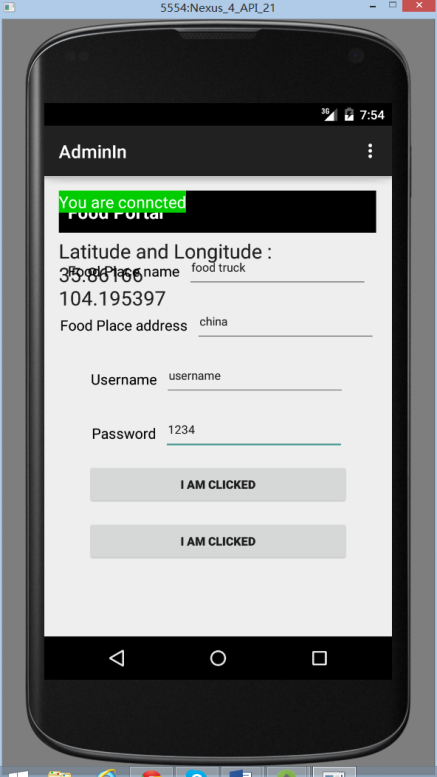
****Figure 2**.** Database table for restaurant menu items

Figure 3. Database table for restaurant locations

Figure 4. Database table for restaurant reviews

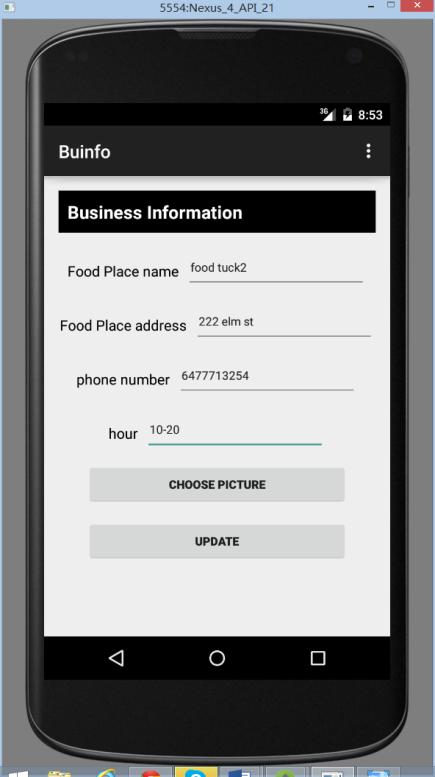
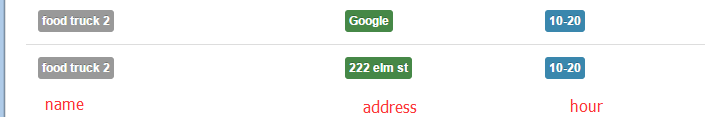
**Test Case 2 (Admin + Third-Party Test Server)**

Figure 5. Screen flow showing the procedure for adding new information

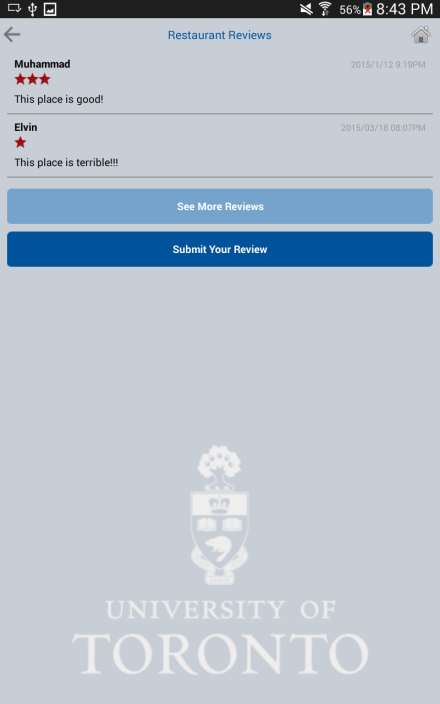
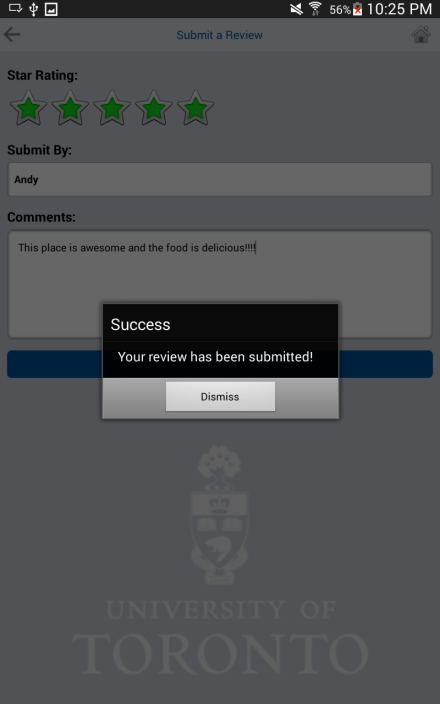
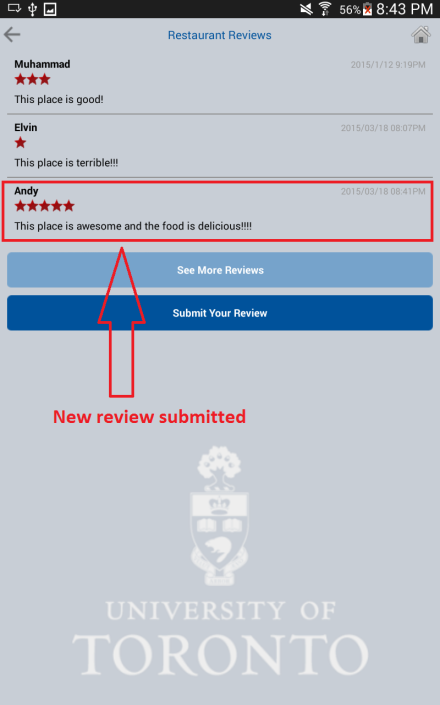
**** 

**Test Case 3 (Admin + Third-Party Test Server)**

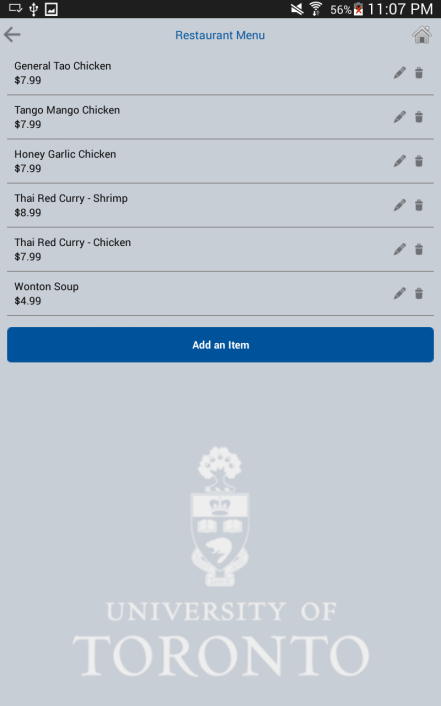
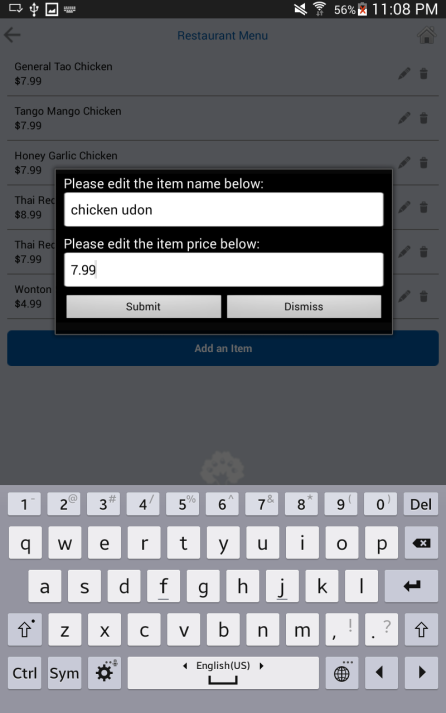
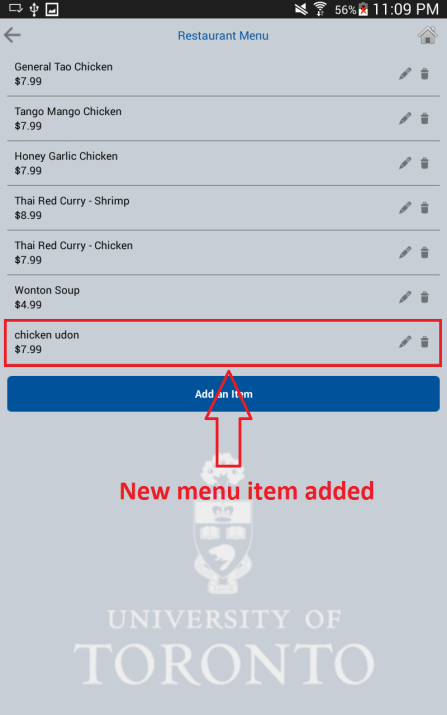
Figure 6. Screen flow showing the procedure of updating existing restaurant information

 ****

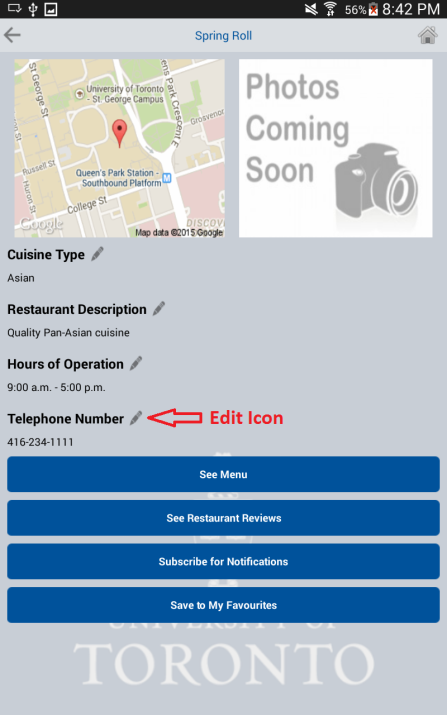
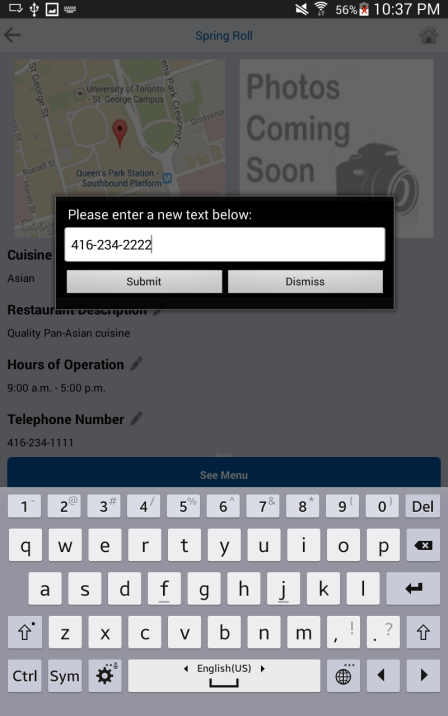
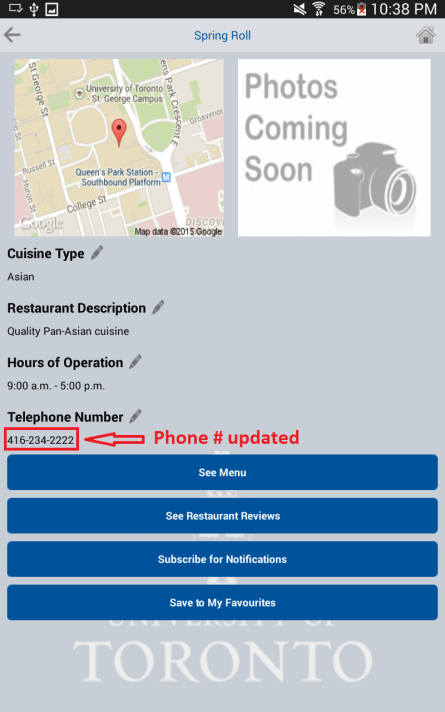
**Test Case 4 (Client + Python Server)**

Figure 7. Screen flow showing the procedure of restaurant review submission  
  

**Test Case 5 (Client + Python Server)**

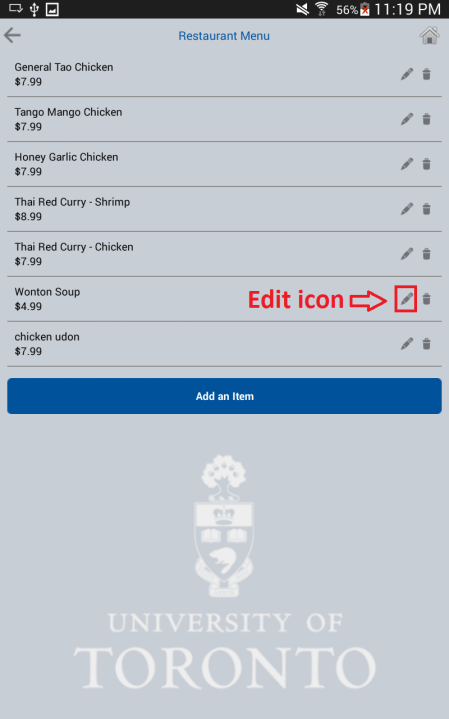
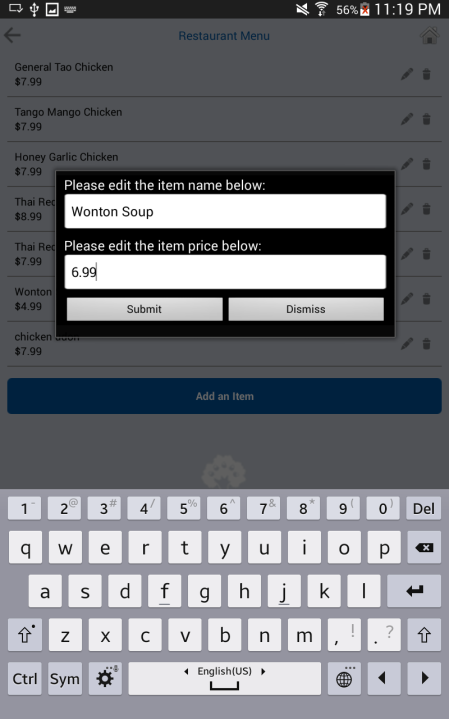
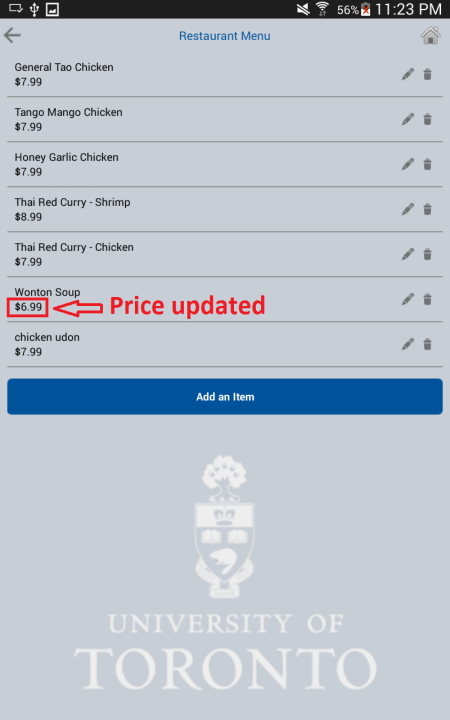
Figure 8. Screen flow showing the procedure of adding a new menu item  
****  

**Test Case 6 (Client + Python Server)**

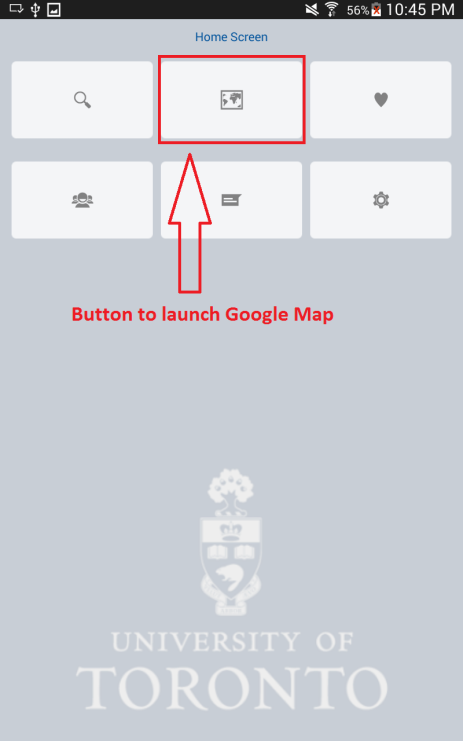
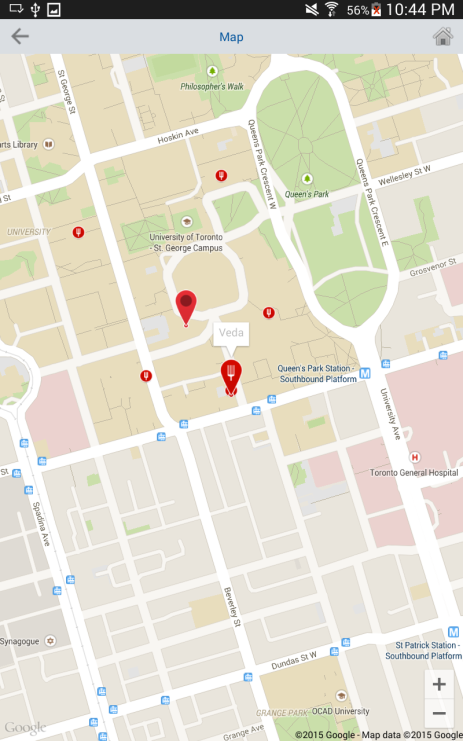
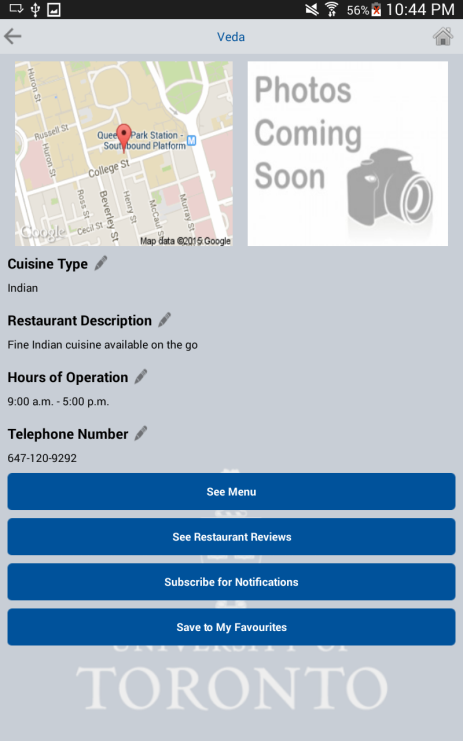
Figure 9. Screen flow showing the procedure of editing restaurant details  
  

**Test Case 7 (Client + Python Server)**

Figure 10. Screen flow showing the procedure of editing menu information

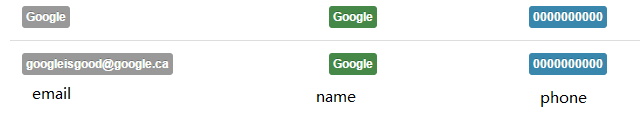
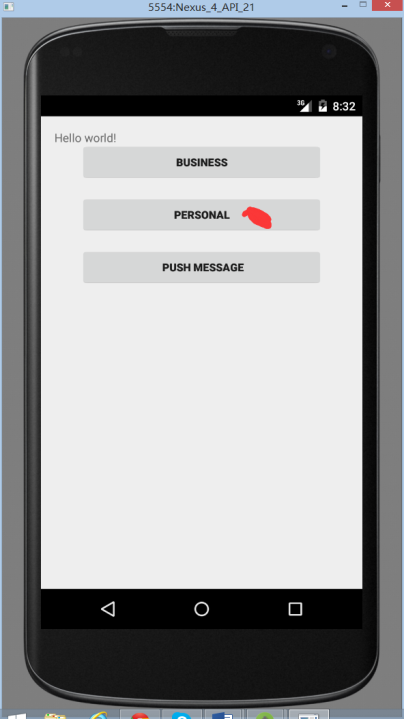
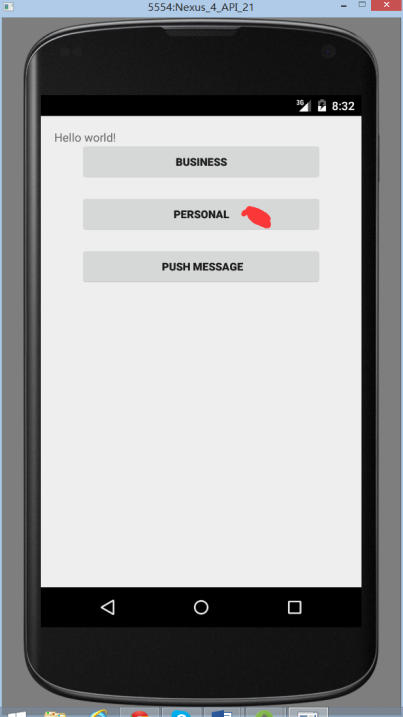
**  **

**Test Case 8 (Client + Python Server)**

Figure 11. Screen flow showing the procedure of showing restaurant information  
  

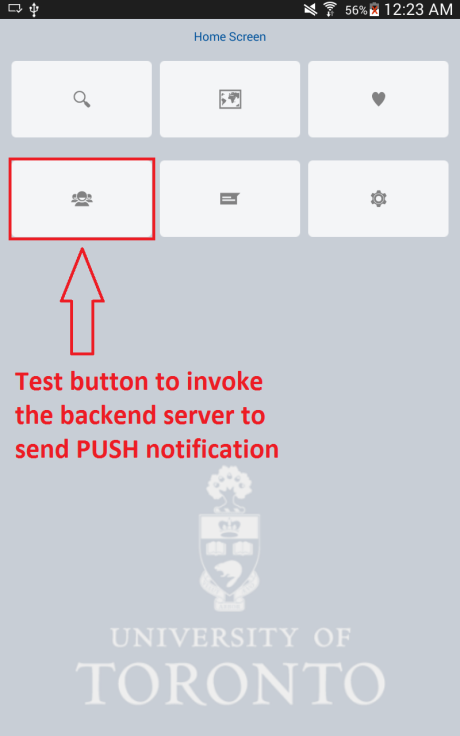
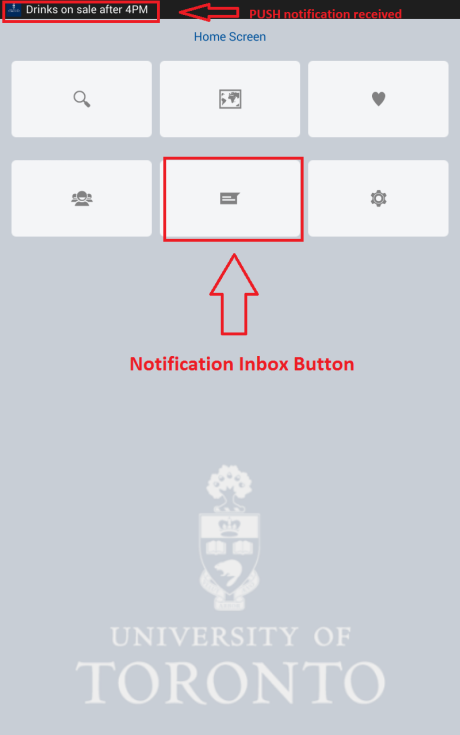
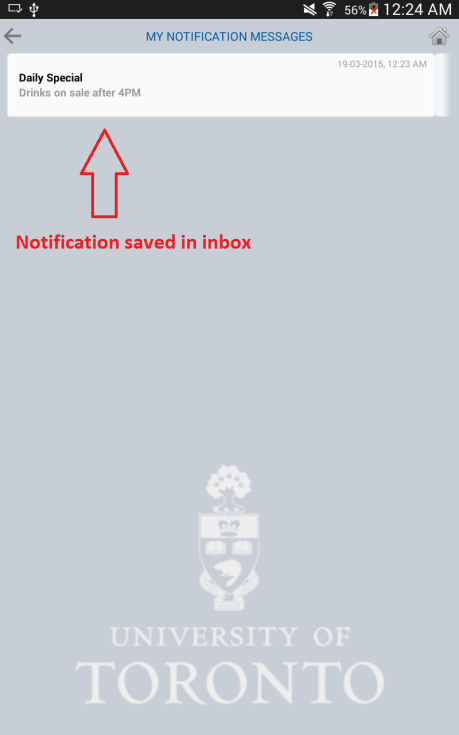
**Test Case 9 (Admin + Third-Party Test Server)**

Figure 12. Screen flow showing the procedure of creating restaurant profile page



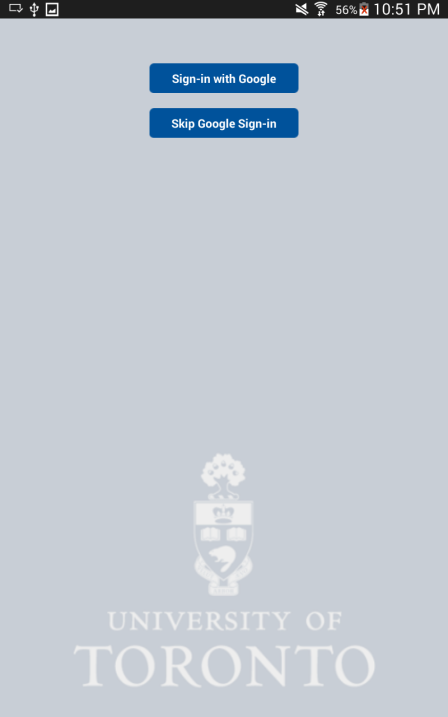
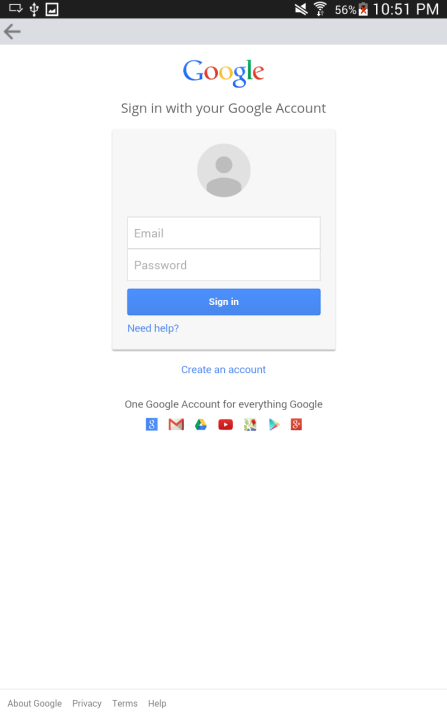
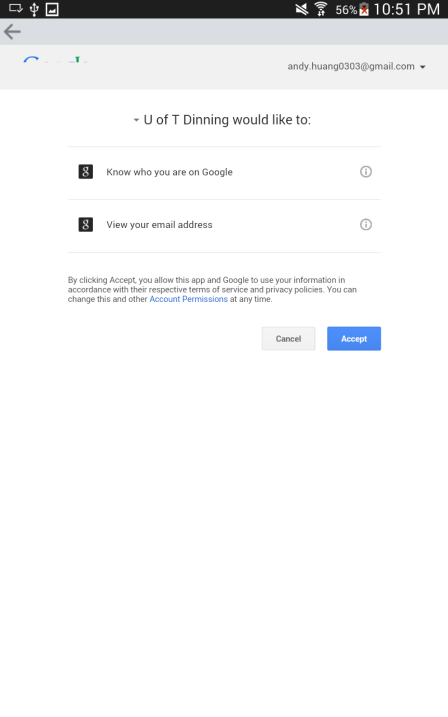
**Test Case 10 (Client + Python Server)**

Figure 13. Screen flow showing the reception of PUSH notification

**  **

**Test Case 11 (Client + Python Server)**

Figure 14. Screen flow showing Google Login with OAuth protocol

**Appendix D: Database Query Test Cases (author: Andy Huang)**

**Unit Test Case 1: Get all restaurants in the database to plot on the client Google Map**  
Query: SELECT \* FROM restaurant;

**Unit Test Case 2: Get all menu items in Veda**  
Query: SELECT \* FROM menu WHERE menu.restaurantID = (SELECT restaurantID FROM restaurant WHERE restaurantName = "Veda");

**Unit Test Case 3: Get all menu items in SpringRoll @ Medical Science Building**  
Query: SELECT \* FROM menu WHERE menu.restaurantID = (SELECT restaurantID FROM restaurant WHERE restaurantName = "Spring Roll" and buildingName = "Medical Science Building");

**Unit Test Case 4: Get the average review rating of Veda**  
Query: SELECT AVG(reviewRating) FROM review WHERE review.restaurantID = (SELECT restaurantID FROM restaurant WHERE restaurantName = "Veda");

**Unit Test Case 5: Get all reviews of Veda**  
Query: SELECT \* FROM review WHERE review.restaurantID = (SELECT restaurantID FROM restaurant WHERE restaurantName = "Veda");

**Unit Test Case 6: Add a new restaurant point of interest**Query: INSERT INTO restaurant VALUES (6, "Hart House Cafe", "Sanford Flemming Building", "-79.394416", "43.65935");

**Unit Test Case 7: Add a new menu item**  
Query: INSERT INTO menu VALUES (1, 7, "3-Pack Samosa", 4.49, 0);

**Unit Test Case 8: Add a review for a restaurant**Query: INSERT INTO review VALUES (1, 3, “Nice place to dine!”, 4, “2015/1/12 10:32PM”, “Reviewer Bob”);

**Unit Test Case 9: Edit a menu item price**Query: UPDATE menu SET price=5.99 WHERE restaurantID=1 and itemID=1

**Unit Test Case 10: Delete a menu item**  
Query: DELETE FROM menu WHERE restaurantID=1 and itemID=1

Note: All the above test cases will be frequently used in the data transactions between the database and the server. In many cases, the ID constant values would be passed by the client interface as a command parameter.