Table of Contents

1. Project Title and Authors

Team Number: 11
Team Name: Grindhouse
Team members:
Mallika Jain 6559234094
Stacy Tran 5305330579
Sandeep Suresh 7720774451
Ethan Smith 5910317497
Shayanna Ahuja 6061783107

2. Preface

Bean&Leaf is an Android application that provides customers with an easy way to discover local coffee shops and tea houses and merchants a platform to market their establishments as well as track the success of their business. After following our previous design document and implementing core functionality of our Android application based on the requirements, we carefully outline the design changes our team has made along with the rationale for each during our implementation.

3. Introduction

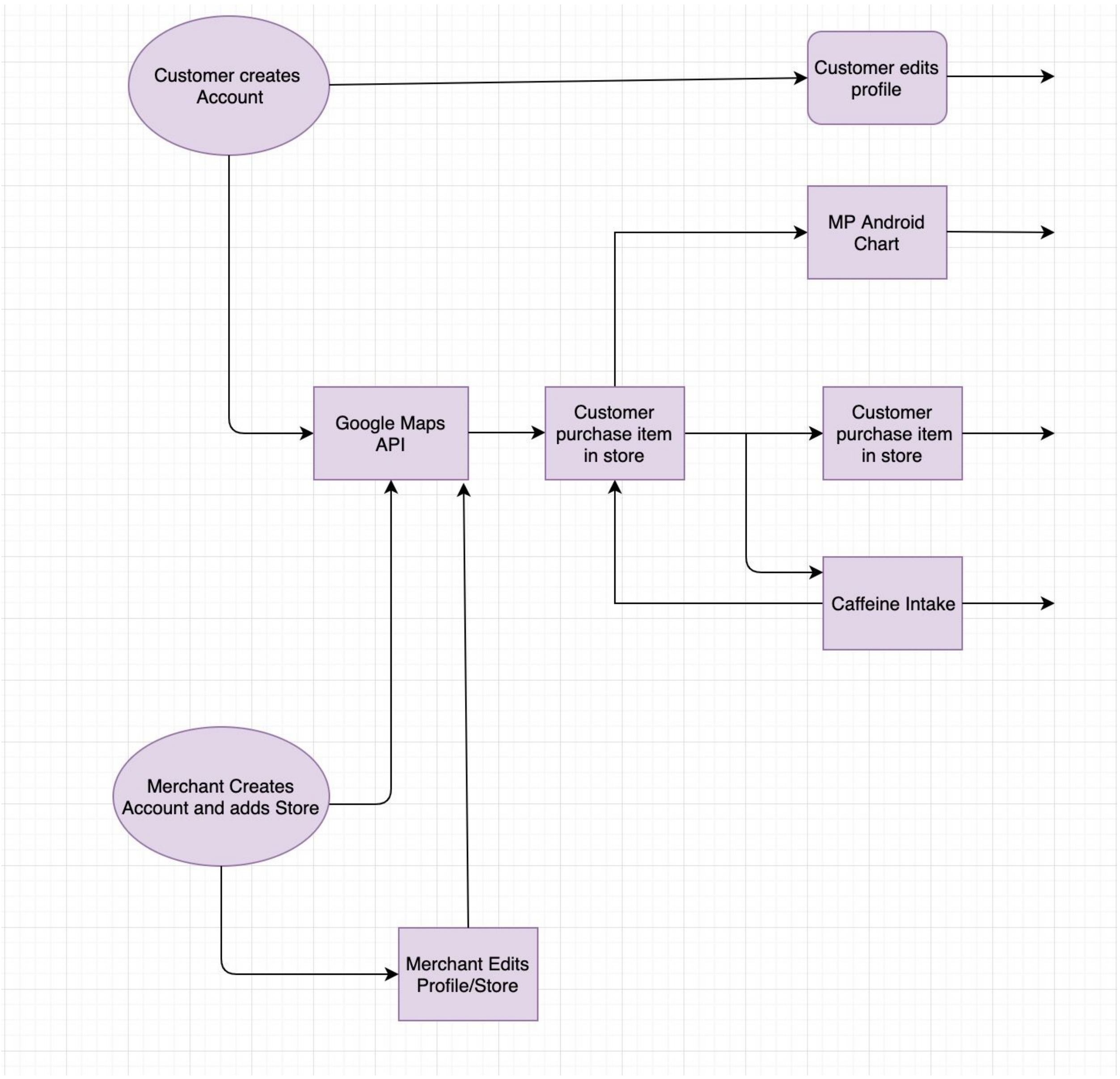
During the implementation of the application, we made adjustments to our architectural and detail design and requirements. One of our major changes to our architectural design was to utilize a systems location through permissions rather than use the geolocation API to get a user's current location. The geolocation api would have added an extra layer of complexity that wasn't required for the functionality we needed in Bean&Leaf to find surrounding coffee and tea shops. In regards to the detail design, most of our class definitions had minor changes that included adding additional getter and setter functions except our DBAccess class and the Order class. Our database class had to be restructured as we decided not to use mySQL and integrate SQLite instead since it was more time efficient. Our order class had to be adjusted as well since we realized we wanted information about an entire order to be easily accessible. Also, there were minor changes to our user interface such as the caffeine intake notification to the user, having clickable markers for the coffee and tea shops, and the way in which we displayed directions to a customer.

4. Architectural Design Change

- **Block Diagram**
    - We decided not to use the geolocation api for the user to see a map with his/her location. We instead extracted location through permissions and system location.

We simply needed to find the user's current location which can be accessed by the system itself in conjunction with asking for the correct permissions from the user and the Google Maps API. After the user creates his/her account or logs in, the user will be shown his/her location and nearby cafes using the Google Maps API.

- Additionally, instead of the Google Maps API we used the MPAndroidChart API to display the charts for the history screens for the merchant and customer.



Customer creates Account
Customer edits profile
MP Android Chart
Google Maps API
Customer purchase item in store
Customer purchase item in store
Caffeine Intake
Merchant Creates Account and adds Store
Merchant Edits Profile/Store

- ***Changes to Major Components***

Component	*Input*	*Output*
Google Maps API	User Creates a Profile (Gmaps get current location)	User Purchases an Item in Store (sends notification that the user is in the store)
MP Android Chart API	User Purchases an Item in Store (user's purchase history)	Charts for the user to view in the app
Caffeine Intake	User Purchases an Item in Store	User can see their caffeine intake for that day by clicking on the caffeine button

5. Detailed Design Change

- **Class Diagram Changes**
  - *Database Helper Class*: adjusted for SQLite database, parameters and return types for getters and setters, and added more functions as needed
    - We found it more efficient to integrate SQLite vs. MySQL because it's prebuilt into Android Studio therefore, we no longer needed the Connection, Prepared Statement, or ResultSet variables within this class while adding onCreate() and onUpgrade() functions to instantiate and manage our database. In addition, we decided to pass in specific information about a user rather than the entire User object since we made the User Class abstract. This caused us to added functions such as getUserId(), getUserGender(), getUserName(), getUserPassword(), updateUserId(), updateUserName(), updateEmail(), updateGender(), and updateUserPassword(). These functions allow us to get and update a single piece of information from a user's profile without having to reaccess other information that user may not want to change. Additionally, we added other functions like updateStoreName(), updateStoreLat(), updateStoreLong() along with all of the getters and setters for our Customer and MenuItem class as needed.
  - *Order Class*: restructured our order class
    - We added orderID, itemID, quantity, caffeine, calories, price, name, and orderTime as member variables along with getters and setters for each rather than the itemsPurchased map in our design documentation. We realized we wanted to track the price, calories, caffeine, etc. of an entire

        - order instead of one menu item. In order to access one item's information we utilized an itemID to access the item's information from our menuItem class.
    - *Customer Class:* no longer needed to track order history but instead an array of trip objects
        - We removed the userHistory map and implemented an array of trip objects. Each trip object then has an array of order objects associated with it that track the menu items ordered by a user at a shop. This was a more organized approach to storing customer order information because it was difficult to track several different menu items, which store the user was located in, and information about an entire order.
    - *Store Class*: changed the location member variable
        - We adjusted location member variable to two floats in order to store longitude and latitude because we found it to be unnecessary to have the location as its own object.

6. Requirements Change

- Caffeine tracking
    - We added a caffeine button as opposed to a pop-up. This changes the design slightly because we have to add a button that navigates to a display that has the caffeine info, instead of it only being popped up on whichever page you were on, once you reach a determined threshold of caffeine intake.
- Coffee and tea shop location markers
    - The markers that signify a coffee or tea shop at a given location are now clickable. A click on a store marker triggers a pop-up display that shows the name of the store, and clicking the store's name will link to the menu of the store. This will only slight change the design because we have to create a small display when the user clicks on the store to display the store's name and link the name to the store's menu.
- Location based Logging Orders
    - Instead of a pop up occuring when a customer is close to a store asking them to log their order, the customer will manually log their order by clicking the store they visited and clicking "Log Order." This will change the design because we have to create additional buttons to log the order instead of using the customer's current location to generate a pop up.
- Directions to a coffee or tea shop
    - Decided to display directions on a google map interface instead of listing the directions in our app itself. This decision doesn't change the design of the

    application because all the exit options after the user gets directed to the map is the same. Only the display itself on that screen changes.

- Merchant verification
    - We added a merchant verification process as a new requirement in order to keep our application as secure as possible. When a merchant creates a new store they are required to submit a photo of their store for verification. Their marker for their store location will not display to users until their store is verified. An admin must login through through our landing page to approve a certain store's verification photo with a universal admin password. This will then direct them to a page where they can view all stores in the process of verification and select the necessary store for the marker to display to all users.
- Admin user types
    - In our original design, we only accounted for customers and merchants as our user types. However, we realized in order to have an effective merchant verification process it was necessary to include a universal admin to ensure the validity of the stores within our application.
- Customer order history charts
    - Instead of customers viewing a table with all of their previous orders, our team decided to display a users' order history through three charts and stats. Within our order history tab, there's a dropdown with "Today","Past Week", and "All Time" and these charts and stats will dynamically update when a time interval is selected. For a customer, they can see money spent, calories consumed, and caffeine consumed (for the interval they choose; today is selected by default). A customer can also see a pie chart with the drinks they've purchased, a bar chart displaying the number of visits to different stores, and another bar chart of the money they've spent per day in the past week by day of the week. The last chart updates only when "This Week" or "All Time" is selected. For a merchant, they can see money earned and orders placed (for the interval they choose; today by default). A merchant can see a pie chart with the breakdown of menu items purchased (similar functionality to the customer one), a bar chart with money earned per day of the week (similar functionality to the customer one), and another pie chart that displays the number of visits a customer made to the store, which is updated by a given time interval.

7. Demo Application

Pull from the following Github repository:
https://github.com/Sandeepsuresh1998/GrindHouse.git

The instructions to run the application is in the README.md