Northern Arizona University

Design Specification

CS 486 - Localization

Blayne Kennedy, Kimi Oyama, Daren Rodhouse, Chihiro Sasaki

Contents

Introduction

MoneyClip Mobile (MCM) is a web-based payments infrastructure providing fee-free transactions to customers and merchants through the use of mobile smartphones. Mr. Joshua Cross of Hermes Commerce, Inc. is planning to make this mobile payment system more robust by providing targeted advertisements, coupons, and managing customer's royalty programs.

This project is sponsored by Mr. Joshua Cross, CTO of Hermes Commerce Inc. in Flagstaff, Arizona. Hermes Commerce Inc. produces an application called MoneyClip Mobile (MCM) that allows users to transfer funds to and from other users, merchants or customers, without involving physical money or credit and debit cards. Mr. Cross is responsible for the development of MCM as well as for the business aspects of the product. MCM addresses the need of users to exchange funds regardless of distance.

The general problem facing MCM is competition with popular technologies such as Google Wallet and Square. Mr. Cross' goal for overcoming this challenge is to combine all the functionalities of the competition into one application and find a niche for that application. In order to accomplish this, MCM application needs to be able to transfer funds between customers and merchants. In addition, the application needs to determine when a customer is making a purchase from a merchant in order for the merchant to push the charge to the customer's MCM account. Also, MCM will give customers control over the frequency and types of advertisements or coupons they receive, and it will give merchants options for when to push advertisements or coupons depending on customer trends.

This project involves implementing a localization function that notifies merchants when MCM customers are in their vicinity, allowing the merchant to push a payment request to the customer's MCM account if the customer makes a purchase. This functionality allows true moneyless transactions. Another piece of functionality we are adding is directed advertisements and coupons based on the customer's location and transaction history. The customers will have the ability to select the amounts and sources of advertisements that are pushed to their account via the iPhone application, Android application, or website. They should be able to customize their settings to choose differing levels of participation from the advertisements. Merchant settings should also be available for change on the website or either mobile application.

The key architectural elements that will be implemented for the localization feature are:

- Merchant interface (Android, iOS, website)
- Customer interface (Android, iOS, website)
- Web Service
- Database

The above elements will be discussed in the following sections. One of the key risks is the Google API limit, so each of these architectural elements will limit its use of the Google API to as little as possible.

Architecture

Overall System Architecture

The entire system consists of two types of user interactions: customer and merchant. Each interaction is supported by three different platforms, iOS 5 running on an iOS device, Jelly Bean 4.2 running on an Android device, and a website. Each of the six interfaces, as seen in Figure 1, connects to web services on the server, which then interact with the database. The interfaces do not directly interact with each other; the web services acts as an intermediary among different interfaces.

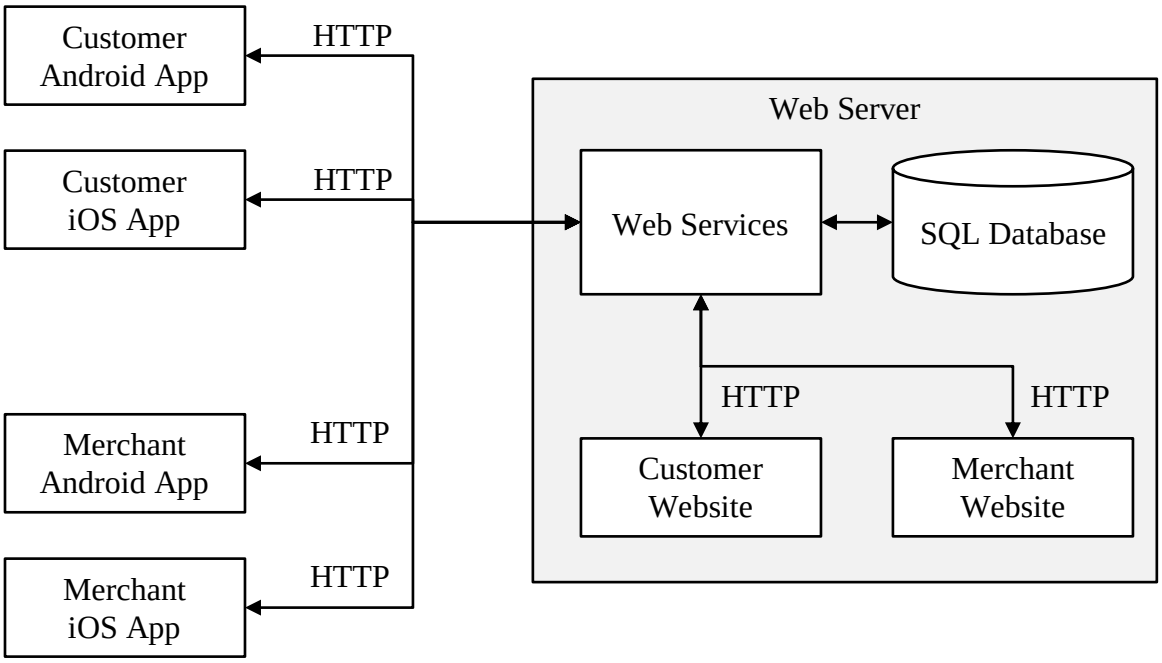


Figure 1: Overall system with its submodules and their interactions.

The four mobile device interfaces are placed external to the web server. The two website interfaces interact internally within the web server since the web services and the website live on the same server.

As seen in Figure 1, the main data flow between mobile devices and the web service happens through an HTTP request, where data passed is packaged in a JSON string. The customer and merchant website components also interact with the web service through an HTTP request. The web services interact with these interfaces in a RESTful way, using only POST, GET, PUT, and DELETE operations.

Customer Interfaces

The customer interfaces allow the customers to interact with the MoneyClip Mobile web service. The main features include viewing account details, changing account settings, accepting charges from a merchant, changing the localization settings, viewing nearby merchants, and receiving notifications of new advertisements and coupons.

Website

The customer interface for the website provides the basic functionality for the generalized customer interface, as seen in Figure 2. The difference between the mobile applications is its lack of updating the customer's current location using a network provider.

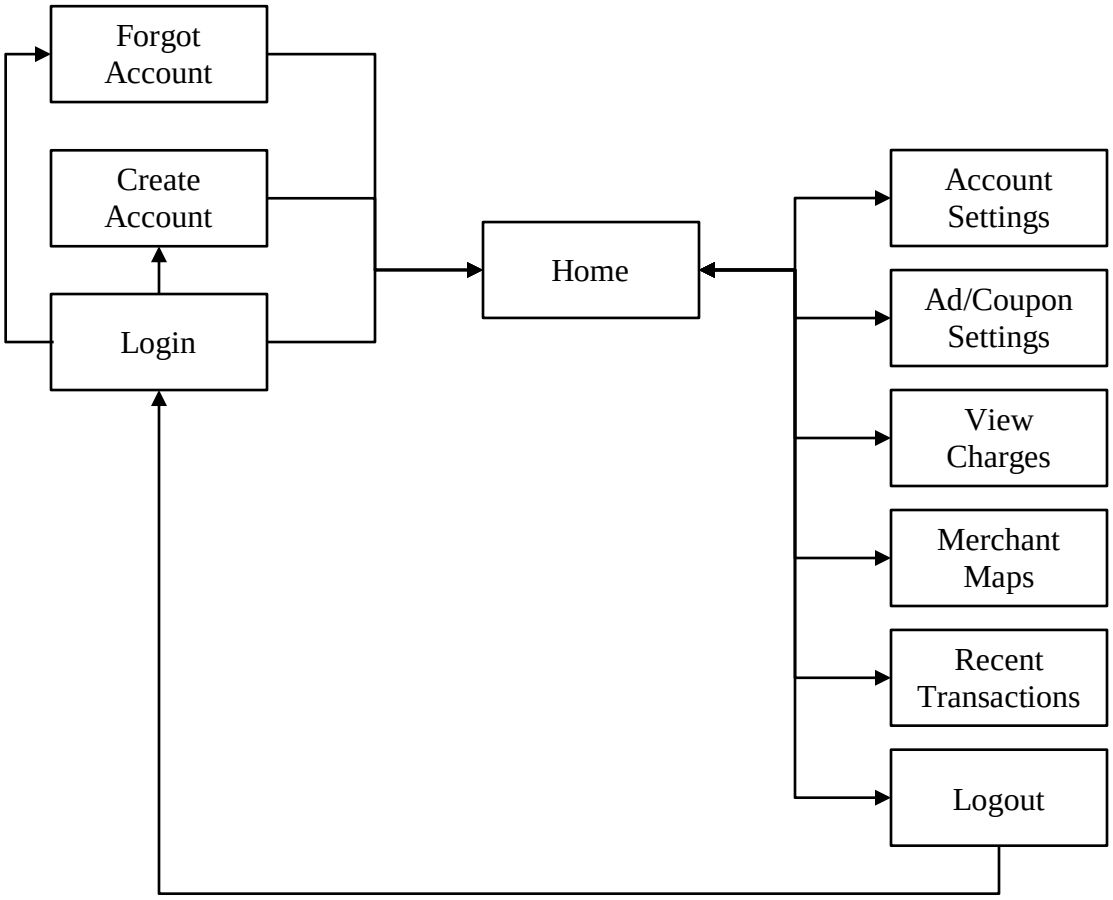


Figure 2: Customer website module with its submodules and their interactions.

Mobile Application

The customer application module consists of several submodules shown in Figure 3. When the app is first opened, the initial login page allows registered users to access the core submodules of the system. The login page also allows new users to create an account or for registered users to retrieve their account information.

When a notification is pushed to the device about nearby deals, clicking on the deal opens up the MCM app. This will allow the user to take advantage of nearby deals without having to click on multiple pages to get to the deal.

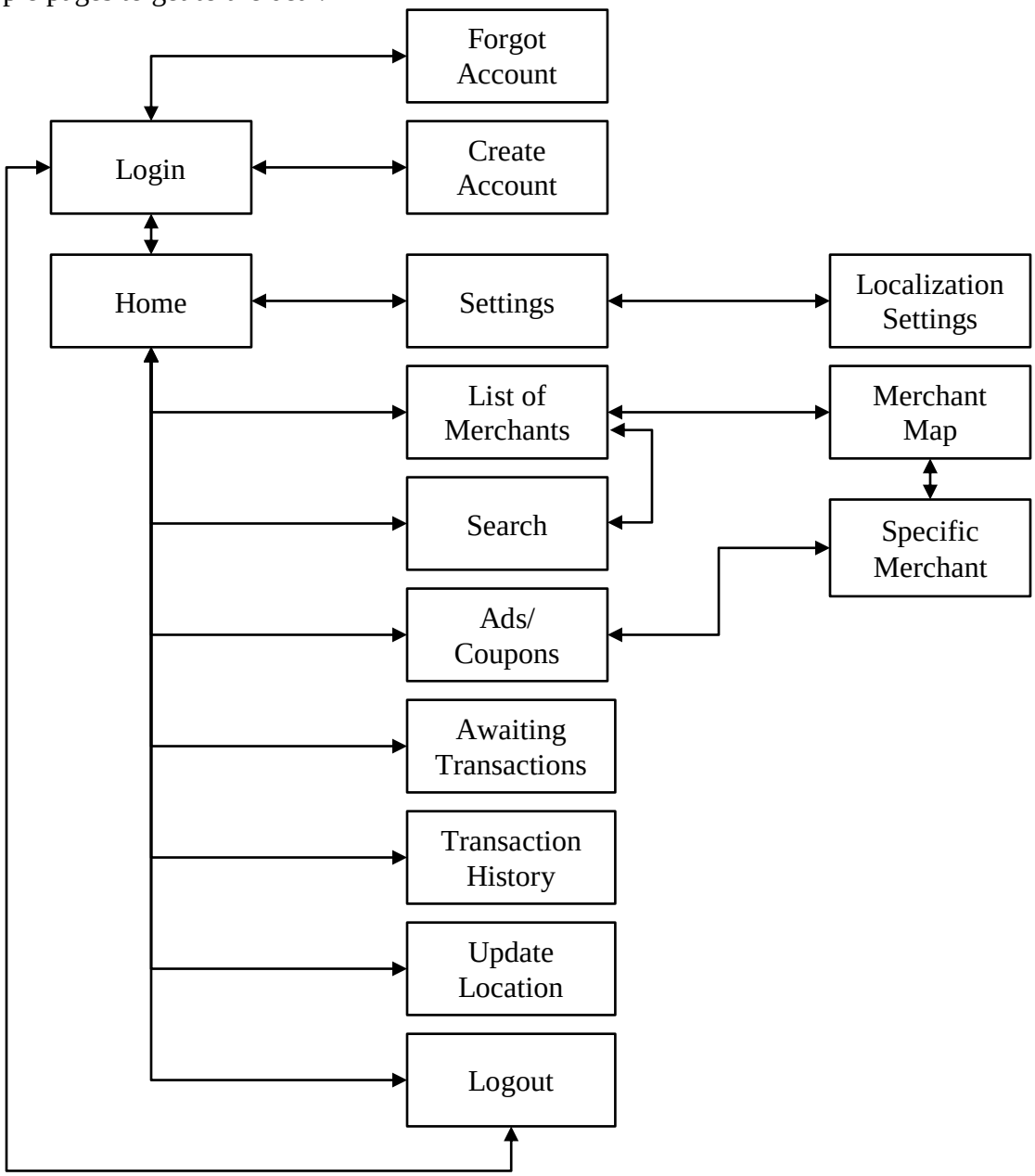


Figure 3: Customer app module with its submodules and their interactions.

Merchant Interfaces

The merchant interfaces allow merchants interact with the MCM web service. The main features include viewing nearby customers, charging customers within their vicinity, and viewing recent transactions.

Website

The website consists of all features of the merchant mobile application, in addition to several functionalities that are unique to the website. The merchant website architecture can be seen in Figure 4. The website has two pieces of functionality that the mobile applications do not allow: creating ads and creating coupons. These features are only implemented on the website because of the complexity of the forms; they would not translate to the screen size of mobile devices well.

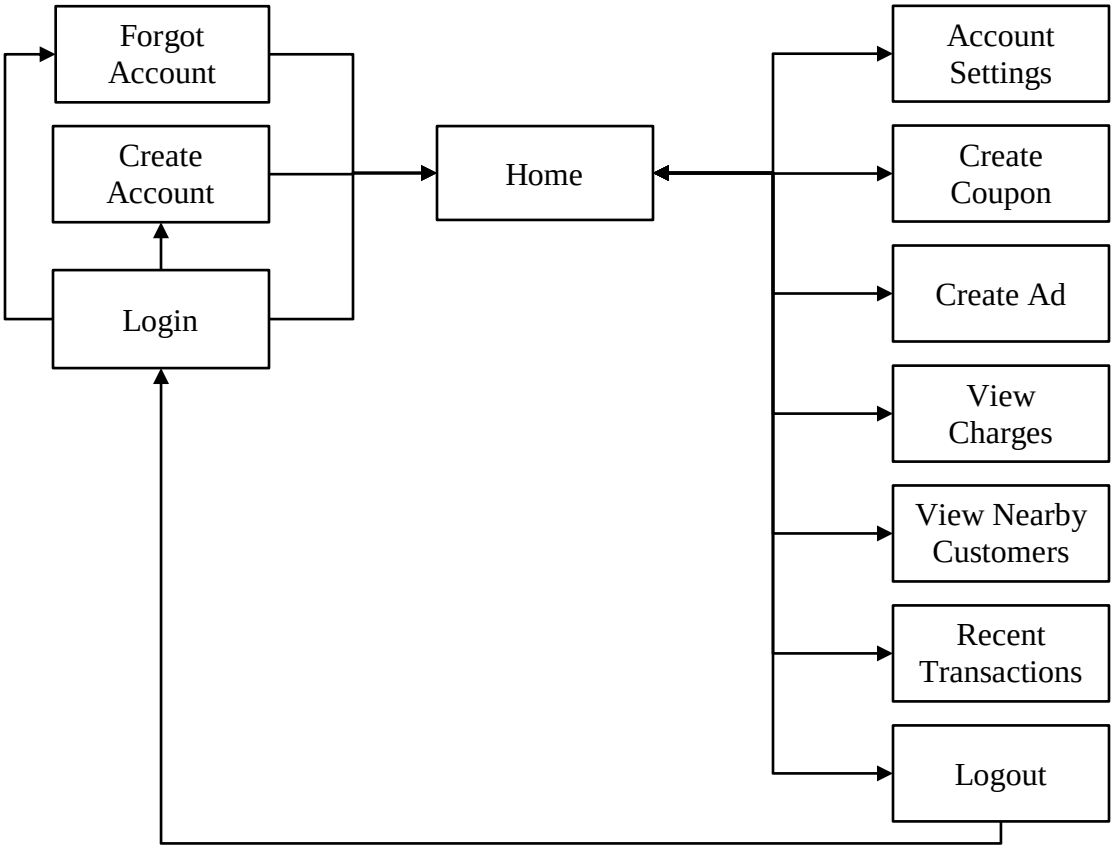


Figure 4: Merchant website module with its submodules and their interactions.

Mobile Application

The mobile application consists of the features available to the website, minus the feature to create ads and coupons. The mobile app architecture can be seen in Figure 5. The mobile application pulls a list of available ads and coupons from the web service, which they can then push to customers from their app. The ads and coupons are pre-approved before they are posted on the list, which addresses one of the risks of inappropriate ad content.

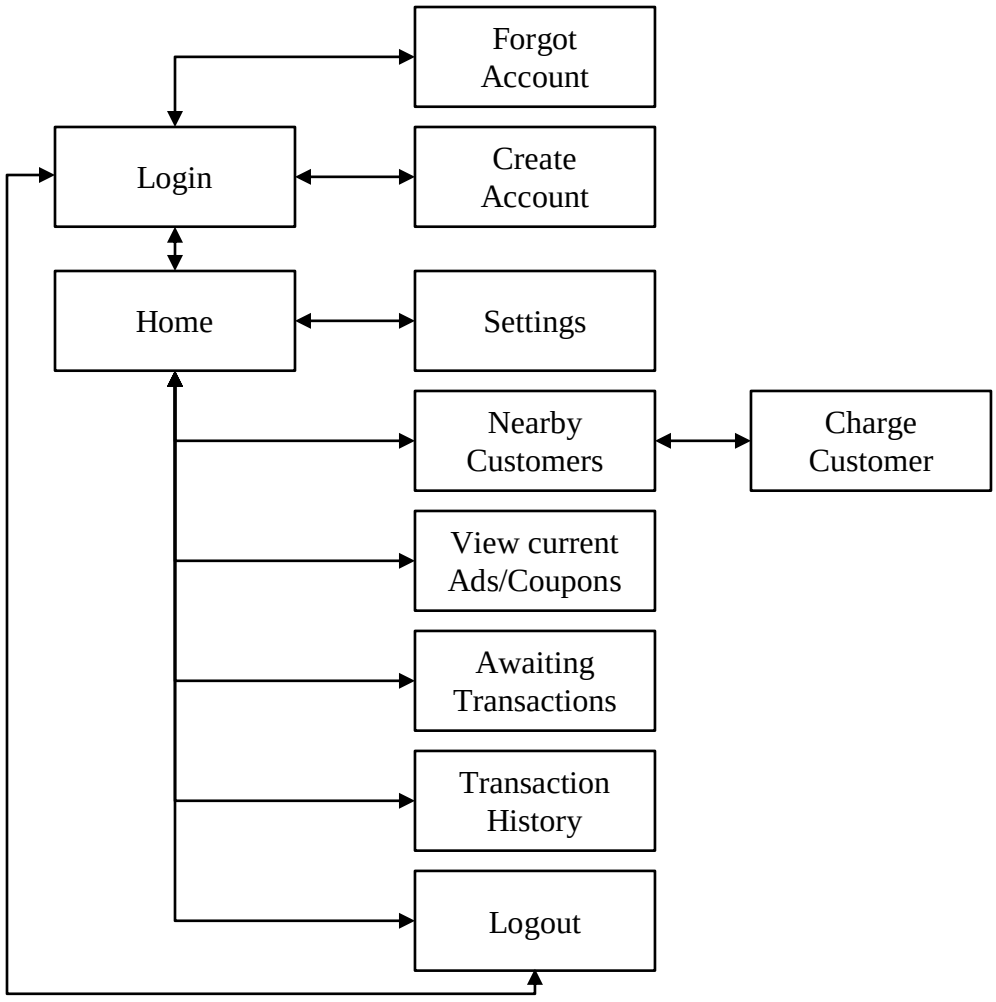


Figure 5: Merchant app module with its submodules and their interactions.

Server

The server(s) host the web services, database, and the websites, seen in Figure 6. The web services take requests from the user interfaces and query the database, performing the requested update or returning the requested information. The web service uses a RESTful architecture, which only uses the basic operations of POST, GET, PUT, DELETE. Information is passed from the user interfaces through an HTTP request, where parameters are passed as JSON strings.

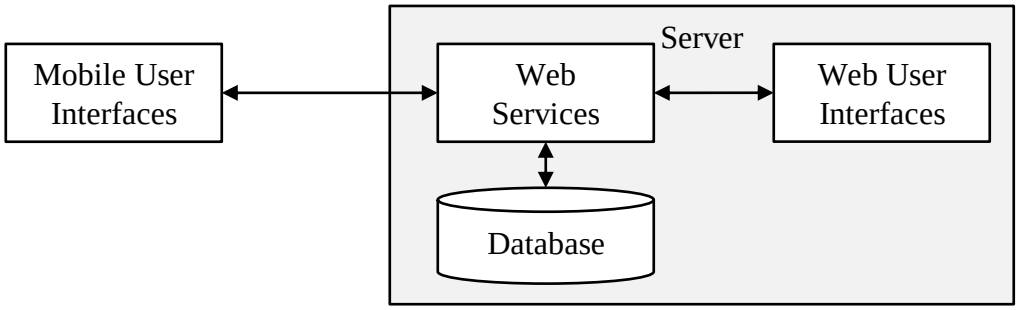


Figure 6: Server module with its submodules and their interactions.

Module and Interface Descriptions

This section provides a description of all modules included in all interfaces of the customer and the merchant, as well as the web service and the database.

Customer iOS App

Pending Charges

The **pendingCharges** view allows users to view all their charges that have yet to be accepted or declined. The details of the module can be seen in Table 1, with a list of all of its containing functions. This module works by loading all their transactions into a table view and passes the information on to the next view. This is done by using the database given a username to seed all the transactions.

Table 1: Module descriptions for pending charges page.

pendingChargesViewController
userCharges: NSMutableArray
initWithStyle (id) : self viewDidLoad (void) : void didReceiveMemoryWarning (void) : void numberOfSectionsInTableView (NSInteger) : NSInteger numberOfRowsInSection (NSInteger) : NSInteger cellForRowAtIndexPath (UITableViewCell) : cell didSelectRowAtIndexPath (void) : void

Table 2: Interface descriptions for a function in pending charges page.

initWithStyle
id : NSInteger : the id of the view
The function creates a controller object that manages a table view.

The **viewDidLoad** function is where the username is used to supply the table view page all the user's current transactions, seen in Table 3, along with its corresponding function in Table 4. This is where a user can pick one of the transactions to be directed to a more detailed view of charges. The mutable array **userCharges** contains all the users charges pulled from the database.

Table 3: Interface descriptions for a function in pending charges page.

viewDidLoad
The function uses the username to pull all the userCharges from the database. Also loads the table view controller.

Table 4: Interface descriptions for a function in pending charges page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 5: Interface descriptions for a function in pending charges page.

numberOfSectionsInTableView
numberOfSections : NSInteger : The number of sections in the table view.
The function gives the total number of sections to the tableview during viewDidLoad.

The number of rows in the table view is pulled from the number of elements in the mutable array **userCharges** passed in during the **viewDidLoad** method, which happens before the table view is constructed.

Table 6: Interface descriptions for a function in pending charges page.

numberOfRowsInSection
numberOfRows : NSInteger : The number of rows in the table view.
The function gives the total number of rows to the tableview during viewDidLoad.

Table 7: Interface descriptions for a function in pending charges page.

cellForRowAtIndexPath
cellForRowAtIndexPath : UITableViewCell : Gives the cell that was selected for a given cell.
The function asks the data source for a cell to insert in a particular location of the table view.

Table 8: Interface descriptions for a function in pending charges page.

didSelectRowAtIndexPath
Tells the delegate that the specified row is now selected.

History

The history view, shown in Table 9, is a module showing a table view of all current and past used deals. This page is created by connecting to the database and pulling all deals based on the user settings, as well as past deals that has been used by the customer.

Table 9: Module descriptions for history view page.

HistoryViewController
deals:NSMutableArray indexNumber:NSMutableArray

viewDidLoad (void):void viewDidUnload (void):void shouldAutorotateToInterfaceOrientation (bool):bool numberOfRowsInSection (NSInteger):NSInteger cellForRowAtIndexPath (UITableViewCell):cell prepareForSegue (void):void

The **viewDidLoad** function uses the username to receive all deals that the user currently can use as well as past deals. This is done by passing the logged in user to the database and pulling all the current data for said given user, and the table view is constructed to include all deals. This is done by having all deals into the mutable array **deals** and having the index number of the database into the mutable array **indexNumber**.

Table 10: Interface descriptions for a function in history view page.

viewDidLoad
The function uses the username to pull all the user's deals from the database. Also loads the table view controller.

Table 11: Interface descriptions for a function in history view page.

viewDidUnload
Returns all resourced allocated to the view.

Table 12: Interface descriptions for a function in history view page.

shouldAutorotateToInterfaceOrientation
Returns a Boolean value indicating whether the view controller supports the specified orientation.

The number of rows in the table view is pulled from the number of elements in the mutable array **deals** passed in during the **viewDidLoad** method but before the table view is constructed.

Table 13: Interface descriptions for a function in history view page.

numberOfRowsInSection
numberOfRows : NSInteger : The number of rows in the table view.
The function gives the total number of rows to the tableview during viewDidLoad.

Table 14: Interface descriptions for a function in history view page.

cellForRowAtIndexPath
cellForRowAtIndexPath : UITableViewCell : Gives the cell that was selected for a given cell.

The function asks the data source for a cell to insert in a particular location of the table view.

The mutable array **indexNumber** is passed onto the new view, which is the **historyDetail-ViewController**. The condition to move to this view is that a user has a deal that can be accessed.

Table 15: Interface descriptions for a function in history view page.

prepareForSegue
Give a condition and/or a variable to the next view.

History Detail

The **historyDetailViewController** is where the detailed information of a given deal is displayed to the customer. This is done by connecting to the database using both the username and the mutable array **indexNumber** that was passed by the last view.

Table 16: Module descriptions deals page.

historyDetailViewController
deals:NSMutableArray indexNumber:NSMutableArray
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void

Table 17: Interface descriptions for a function in deals page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The **viewDidLoad** function is where the view controller is set up to include the detailed information of a given deal. This is done by passing both the username and **indexNumber** to the database and receiving the detailed deal that is then displayed on the view controller for the user.

Table 18: Interface descriptions for a function in deals page.

viewDidLoad
The function uses the username and indexNumber to pull a deal from the database. Also loads the table view controller.

Table 19: Interface descriptions for a function in deals page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

New User Registration

The **registerViewController** is where the user would set up a new account. In this module, the app pulls information from the form fields, which include the username, password, confirmed password, and the name of the user. This information is sent to the database, and if there is no one with the same username, a conformation is given to the user. If there is any error creating the new account, an alert is sent informing them that that username is taken or any other errors that may occur.

Table 20: Module descriptions for new account page.

registerViewController
userText:UITextField passwordText:UITextField repasswordText:UITextField firstNameText:UITextField lastNameText:UITextField
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void viewDidUnload(void):void textFieldShouldReturn(bool):bool textFieldDidBeginEditing(void):void textFieldDidEndEditing(void):void animateTextField(void):void

Table 21: Interface descriptions for a function in new account page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The **viewDidLoad** function is where the request to the database is performed, given the user filled out all of the fields. It is also where a confirmation or an alert is given depending on whether or not the registration was successful. Registration cannot succeed if the requested username is already taken or if the fields are not filled out correctly. If registration succeeds, the account is created and the database is populated with the new user and corresponding information.

Table 22: Interface descriptions for a function in new account page.

viewDidLoad
The function connects to the database and depending on conditions creates a new user or gives an alert. Also loads the table view controller.

Table 23: Interface descriptions for a function in new account page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 24: Interface descriptions for a function in new account page.

viewDidUnload
Returns all resourced allocated to the view.

Table 25: Interface descriptions for a function in new account page.

textFieldShouldReturn
Asks the delegate if the text field should process the pressing of the return button.

Table 26: Interface descriptions for a function in new account page.

textFieldDidBeginEditing
Tells the delegate that editing began for the specified text field.

Table 27: Interface descriptions for a function in new account page.

textFieldDidEndEditing
Tells the delegate that editing stopped for the specified text field.

Table 28: Interface descriptions for a function in new account page.

animateTextField
Allows for the textfield to update.

Login

The **loginViewController** is where the user logs in. The two fields required are the username and the password. If the username and password combination is validated through the web service, the user is allowed access to the rest of the application. This is also where a valid login global variable is set to the current user's username.

Table 29: Module descriptions for a login page.

loginViewController
userText:UITextField passwordText:UITextField loginButton:UIButton
initWithNibName (id) : self viewDidLoad (void) : void didReceiveMemoryWarning (void) : void viewDidUnload(void) : void LoginButtonClicked(IBAction) : void textFieldShouldReturn(bool) : bool

Table 30: Interface descriptions for a function in login page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The function **viewDidLoad** sets up all the textfields and also sets up what keyboard and keyboard functions for each text field. For both textfields, **userText** and **passwordText** a normal keyboard is used with a done button to release the keyboard from the view.

Table 31: Interface descriptions for a function in login page.

viewDidLoad
The function connects to the database and depending on conditions logs a user in or alerts them on a bad login. Also loads the table view controller.

Table 32: Interface descriptions for a function in login page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 33: Interface descriptions for a function in login page.

viewDidUnload
Returns all resourced allocated to the view.

The **LoginButtonClicked** function is where both the UITextFields, **userText** and **passwordText** is passed to the database and if there is a match then the user is logged in and the global variable is set for the user's username. If there is not a match, an alert is triggered stating that the username and/or password is wrong and to try again.

Table 34: Interface descriptions for a function in login page.

LoginButtonClicked
The function connects to the database and depending on conditions logs a user in or alerts them on a bad login.
textFieldShouldReturn
textField : Bool : if the textfield should be selected or not.
Asks the delegate if the text field should process the pressing of the return button.

App Delegate

The app delegate is a built-in Apple class for application start-up and other actions that can happen to the application.

Table 35: Module descriptions for built-in application start-up page.

capstoneAppDelegate
application(bool):bool applicationWillResignActive(void):void applicationDidEnterBackground(void):void applicationWillEnterForeground(void):void applicationDidBecomeActive(void):void applicationWillTerminate(void):void

Table 36: Interface descriptions for a function in built-in application start-up page.

application
Application : Bool : starting up or not.
Tells the delegate that the application is about to start up.

Table 37: Interface descriptions for a function in built-in application start-up page.

applicationWillResignActive
Tells the delegate that the application is about to become inactive.

Table 38: Interface descriptions for a function in built-in application start-up page.

applicationDidEnterBackground
Tells the delegate that the application is now in the background.

Table 39: Interface descriptions for a function in built-in application start-up page.

applicationWillEnterForeground

Tells the delegate that the application is about to enter the foreground.

Table 40: Interface descriptions for a function in built-in application start-up page.

applicationDidBecomeActive
Tells the delegate that the application has become active.

Table 41: Interface descriptions for a function in built-in application start-up page.

applicationWillTerminate
Tells the delegate when the application is about to terminate.

Merchant Map

The mapViewController is where the user can see all merchants within a given area, this is done by placing pins on every near merchant that include some information such as the merchants name, location and other items. The pins are made using the **MyAnnotation** class to supply the basic set up.

Table 42: Module descriptions for the merchant map page.

mapViewController
initWithNibName (id):self viewDidLoad (void):void didUpdateUserLocation(void):void shouldAutorotateToInterfaceOrientation(bool):bool UpdateLocation(IBAction):void

Table 43: Interface descriptions for a function in the merchant map page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The **viewDidLoad** is where the map connects to the database and gives it the current location in lat and long cords to get all current merchants within the area of the user. It then also connects to the database to supply the pins and information related to each pin.

Table 44: Interface descriptions for a function in the merchant map page.

viewDidLoad
The function connects to the database and supplies the map with the requested information. Also loads the map view controller.

Table 45: Interface descriptions for a function in the merchant map page.

didUpdateUserLocation
Tells the delegate that the location of the user was updated.

Table 46: Interface descriptions for a function in the merchant map page.

shouldAutorotateToInterfaceOrientation
Returns a Boolean value indicating whether the view controller supports the specified orientation.

The function **UpdateLocation** has the same functionality as **viewDidLoad**, with the addition of a manual update of the current user location.

Table 47: Interface descriptions for a function in the merchant map page.

UpdateLocation
Update Location : IBAction : button press event
Tells the delegate that the location of the user was updated.

MyAnnotation is a simple declaration for adding an object to the map view. The only items it requires are the coordinates, a title, and a sub-title for each pin.

Table 48: Interface descriptions for a function in the merchant map page.

MyAnnotation
coordinate:CLLocationCoordinate2D title:NSString subtitle:NSString
dealloc (void):void

Table 49: Interface descriptions for a function in the merchant map page.

dealloc
Called to clear data allocated by the class.

Project Detail Apple Built-In

The following function, **capstoneDetail**, is another built-in Apple class.

Table 50: Module description for another built-in Apple class.

capstoneDetailViewController
initWithStyle (id) : self

setDetailItem (void):void configureView (void):void viewDidLoad (void):void didReceiveMemoryWarning (void):void

Table 51: Interface description for function in built-in Apple class.

initWithStyle
id : NSInteger : the id of the view
The function creates a controller object that manages a view.

Table 52: Interface description for function in built-in Apple class.

setDetailItem
Method to help you create a custom setter for the sighting property.

Table 53: Interface description for function in built-in Apple class.

configureView
Method is a custom setter method for the detailItem property that was provided by the template.

Table 54: Interface description for function in built-in Apple class.

viewDidLoad
Sets up the view for the application.

Table 55: Interface description for function in built-in Apple class.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Customer Android App

Localization Activity

Localization Activity is called when the application is first created. It will be the first page the user sees and acts as a login screen. There will be fields for the user to enter their username and password, as well as buttons that allow the user to attempt to login, to retrieve their password, or to register as a user.

Table 56: Module description for login page.

LocalizationActivity
inputUserName: EditText inputPassword: EditText
onCreate(Bundle) login()

Table 57: Interface description of XML file for login page.

main.xml
RelativeLayout
MCMLogo: ImageView username: TextView username: EditText password: TextView password: EditText login: Button register: Button forgotAccount: Button

The **onCreate** method is called when the class is first created. It sets the layout of the page to '*main.xml'* and finds all of the components that will be used from the xml file. An onClickListener is set for the *'login'* button which will call the *login()* method when the button is pressed.

Table 58: Interface description for the onCreate method in the login page.

LocalizationActivity – onCreate
Called when class is first created.
1) Sets up the main.xml file. 2) Gives functionality to the login button - login() is called when the button is pressed.

The **login** method sends the data that was input for *username* and *password* to the database and reacts to the databases response. If the combination was found in the database, the *HomeActivity* is set up. If not, a dialogue window is created that lets the user know that the *username* and *password* that they entered was not found.

Table 59: Interface description for the login method in the login page.

LocalizationActivity – login
Called when login button is pressed.

1) Grabs input from username/password fields. 2) Sends username/password to database 3) Reacts to databases response a. If positive, sends user to HomeActivity b. If negative, dialogue window displays error

HomeActivity

HomeActivity provides basic accessibility to all other pages on the application, except pages reserved for administrators and/or merchants.

Table 60: Module description for home page.

HomeActivity
notificationMessage: String
onCreate(Bundle)

Table 61: Interface description of XML file for home page.

home.xml
RelativeLayout
MakePayments: Button ViewHistory: Button GetLocation: Button Merchants: Button Coupons: Button LogOut: Button MCMLogo: ImageView

Table 62: Interface description for the onCreate method in the login page.

HomeActivity - onCreate
Called when class is first created.
1) Sets up the home.xml file. 2) Gives functionality to the MakePayments – user is sent to the MakePayments screen when the button is pressed. 3) Gives functionality to the Merchants button - user is sent to the Merchants screen when the button is pressed. 4) Gives functionality to the GetLocation button – user latitude and longitude is displayed when the button is pressed. 5) Gives functionality to the Coupons button - user is sent to the coupons page when the button is pressed. 6) Gives functionality to the LogOut button – user is sent to the login screen when the button is pressed.

Merchant Map

Table 63: Module description for merchant map page.

MerchantMap
latitude: double longitude: double mapView: MapView itemizedOverlay: MerchantMapOverlay
onCreate(Bundle) getLocation()

Table 64: Interface description of XML file for merchant map page.

map.xml
GoogleMap: MapView

Table 65: Interface description for the map item overlay method in the merchant map page.

MerchantMapItemOverlay
MerchantMapItemOverlay(Drawable, Context) addOverlay() createItem(int): Overlayitem size(): int onTap(int): boolean

List of Merchants

Table 66: Module description for list of merchants page.

Merchants
username: Textview result: TextView dbresult: String listContents: List<String> myListView: ListView
onCreate(Bundle) getData() getMerchants()

Table 67: Interface description of XML file for login page.

merchant.xml
LinearLayout
MerchantListLabel: TextView

MerchantMap: Button MerchantList: ListView

Payments History

Table 68: Module description for payment history page.

PaymentHistory
paymentHistoryLayout: LinearLayout
onCreate(Bundle) getData()

Table 69: Interface description of XML file for payment history page.

history.xml
LinearLayout
PaymentLabel: TextView PaymentList: ListView

Coupons

Table 70: Module description for coupons page.

Coupons
username: TextView result: TextView dbresult: String listContents: List<String> ListView myListView adapter: ArrayAdapter<String>
onCreate(Bundle) getData() getCoupons()

Table 71: Interface description of XML file for coupons page.

coupons.xml
LinearLayout
CouponsLabel: TextView CouponList: ListView

Make Payments

Table 72: Module description for make payments page.

MakePayments
paymentLayout: LinearLayout
onCreate(Bundle) getData()

Table 73: Interface description of XML file for make payments page.

payments.xml
LinearLayout
PaymentsListLabel: TextView
PaymentsList: ListView

Customer Website

The customer website provides a different interface from the mobile devices, but it provides mostly the same functionality. The customer website is to be written in PHP and HTML, where pages on the customer's website are populated according to information retrieved from the database.

Table 74: Module description for customer website.

Customer Website
createAccount (String, String, String, String, String, Bool) : Bool login (String, String) home ( ) settings (String) merchantListMap (int, int) searchDeals (int, int, String) recentActivites (String)

Create Account

Table 75: Interface description for the create account function.

createAccount
username : String : Requested username password : String : Requested password confirmPassword : String : Confirmation of requested password which has to match the password exactly

email : String : Validated email of the user birthdate : String : Validated birthdate of the user student : Bool : Indicates whether or not the user is a student
The createAccount function takes in the requested username, password, confirmed password, email, birthdate, and student status. This sends a query to the database to add this information. On this page, there are several checks to see if the information filled out on the form is valid. The password and the confirmed password must match exactly, the username cannot already exist in the database, and none of the fields can be missing. When the account has been successfully created, the function returns true, and false otherwise.
success : Bool : Indicates whether or not the registration was successful. If anything goes wrong in the function, such as database connection issues, then it returns false.

Login

Table 76: Interface description for the login function.

login
username : String : Username password : String : Password
The login page takes in the username and password and validates the user according to the PHP function in validateLogin. Once the credentials match what is stored in the database, the user is taken to the home page. If the credentials are denied, the user is able to attempt logging in again.
success : Bool : Indicates whether or not the login validation was successful. If anything goes wrong in the function, such as database connection issues, then it returns false.

Home Page

Table 77: Interface description for home page.

home
The home page provides basic accessibility to all other pages on the site, except pages reserved for administrators and/or merchants.

Settings

Table 78: Interface descriptions for settings page.

settings

username : String : Username
The settings page provides user setting options for the specific customer logged in the current session. The page takes in the username and provides customized pages.

List and Map of Merchants

Table 79: Interface descriptions for merchant map page.

merchantListMap
latitude : int : User's current latitude longitude : int : User's current longitude
The merchantListMap page shows a list of merchants in the nearby area. The user is able to update their current location on the merchant map, and the map shows a list of merchants that the user is subscribed to in the area.

Search Deals

Table 80: Interface descriptions for search deals page.

searchDeals
latitude : int : User's current latitude longitude : int : User's current longitude username : String : Username
The searchDeals page allows the user to search deals based on merchant name and product category. The page will then show a list of relevant results.

Recent Activities

Table 81: Interface descriptions for recent activities page.

recentActivities
username : String : Username
The recentActivities page shows a listing of activities performed by the user in the past week or up to ten activities, depending on which is less. If the user wishes to see a more comprehensive list of activities from the past, they are able to click a button that allows them to have increased views of their recent activities.

Merchant iOS App

Nearby Customers

The NearViewController is where a merchant is given a table view of all the current users within a given area. This is done by passing the location of the merchant to the database and receiving a response of all current users that meet the requirements. This is then constructed into a table view.

Table 82: Module description for nearby customers.

NearViewController
near: NSMutableArray indexNumber: NSMutableArray
viewDidLoad (void):void viewDidUnload (void):void shouldAutorotateToInterfaceOrientation (bool):bool numberOfRowsInSection (NSInteger):int cellForRowAtIndexPath (UITableViewCell):cell prepareForSegue (void):void

The **viewDidLoad** function is where the static merchant location is used and connecting to the database pulls all the users who's current location are within a given area. From here the merchant can chose one of the users running the customer application and it will direct them to the **chargeUserDetailView**.

Table 83: Interface descriptions for a function in nearby customers page.

viewDidLoad
The function connects to the database and using the static merchant location pulls all current users who are close to them. Also loads the table view controller.

Table 84: Interface descriptions for a function in nearby customers page.

viewDidUnload
Returns all resourced allocated to the view.

Table 85: Interface descriptions for a function in nearby customers page.

shouldAutorotateToInterfaceOrientation
Returns a Boolean value indicating whether the view controller supports the specified orientation.

The number of rows in the table view is pulled from the number of elements in the mutable array **near** passed in during the **viewDidLoad** method but before the table view is constructed.

Table 86: Interface descriptions for a function in nearby customers page.

numberOfRowsInSection
numberOfRows : NSInteger : The number of rows in the table view.
The function gives the total number of rows to the tableview during viewDidLoad.

Table 87: Interface descriptions for a function in nearby customers page.

cellForRowAtIndexPath
cellForRowAtIndexPath : UITableViewCell : Gives the cell that was selected for a given cell.
The function asks the data source for a cell to insert in a particular location of the table view.

The mutable array **near** is passed into the new view, that being **chargeUserDetailView**. The condition to move to this view is that a merchant must have a user within the given area.

Table 88: Interface descriptions for a function in nearby customers page.

prepareForSegue
Give a condition and/or a variable to the next view.

Charge Customer

The **chargeUserDetailViewController** is where the merchant is able to send a charge to a user. This is done by connecting to the database and passing the selected index that was passed with the mutable array **near** passed from the last view, **NearViewController**.

Table 89: Module description for charge customer page.

chargeUserDetailViewController
chargeField:UITextField userLabel:UILabel userName:NSString
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void viewDidUnload (void):void chargeButtonPressed (IBAction):void

Table 90: Interface descriptions for a function in charge customer page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

Using the passed values of the mutable array **near** passed from the last view, **viewDidLoad** sets up the view controller with the user selected, sets focus to the **chargeField** UITextField, and then auto views the keyboard which is a number pad with a decimal.

Table 91: Interface descriptions for a function in charge customer page.

viewDidLoad
The function sets up the view with the passed user name and focuses the textfield.

Table 92: Interface descriptions for a function in charge customer page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 93: Interface descriptions for a function in charge customer page.

viewDidUnload
Returns all resourced allocated to the view.

The **chargeButtonPressed** function is where the **chargeField** UITextField is sent to the database along with the selected user from the last view using the mutable array **near**. If the UITextField is a number and is not blank then the request will be good. There is also a check on if the user dose exist in the database. If all is good then they message is sent to the database and then to the customer to approve the charge. If the request is bad then there is a alert triggered stating a bad user, a blank field, or a invalid text in the field.

Table 94: Interface descriptions for a function in charge customer page.

chargeButtonPressed
The function connects to the database and depending on conditions sends a charge to the selected user.

Home Page

The **homeViewController** is where the application's main view after the login view. This is where the merchant can go to any of the core functionality views, such as **NearViewController**, the settings page, or any other key view.

Table 95: Module description for home page.

homeViewController
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void

Table 96: Interface descriptions for a function in home page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The **viewDidLoad** is where the view is set up, all links are static and the only data is the global variable of the current merchant logged in.

Table 97: Interface descriptions for a function in home page.

viewDidLoad
The function sets up the view controller.

Table 98: Interface descriptions for a function in home page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Register New Merchant User

The **registerViewController** is where the merchant would set up a new account, with this the database needs a username, a password as well as the password retyped and the name of the user. This is then sent to the database and if there is no one with the same username then a conformation is given to the user, and if not an alert is sent informing them that that username is taken. There are also added steps for merchants such as approval and location.

Table 99: Module description for register new user page.

registerViewController
userText:UITextField passwordText:UITextField repasswordText:UITextField firstNameText:UITextField lastNameText:UITextField
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void viewDidUnload (void):void textFieldShouldReturn (bool):bool textFieldDidBeginEditing (void):void textFieldDidEndEditing (void):void animateTextField (void):void

Table 100: Interface descriptions for a function in register new user page.

initWithNibName
id : NSInteger : the id of the view
Returns a newly initialized view controller with the nib file in the specified bundle.

The **viewDidLoad** function is where the request to the database is given the user filled out all the fields. It is also where a conformation is given or an alert is given depending on if the user filled in a username already taken, they did not fill in all the fields, or if everything was filled out and the username was not taken. If the latter then the account is created and the database populated with the new user and their information.

Table 101: Interface descriptions for a function in register new user page.

viewDidLoad
The function connects to the database and depending on conditions creates a new user or gives an alert. Also loads the table view controller.

Table 102: Interface descriptions for a function in register new user page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 103: Interface descriptions for a function in register new user page.

viewDidUnload
Returns all resourced allocated to the view.

Table 104: Interface descriptions for a function in register new user page.

textFieldShouldReturn
Asks the delegate if the text field should process the pressing of the return button.

Table 105: Interface descriptions for a function in register new user page.

textFieldDidBeginEditing
Tells the delegate that editing began for the specified text field.

Table 106: Interface descriptions for a function in register new user page.

textFieldDidEndEditing
Tells the delegate that editing stopped for the specified text field.

Table 107: Interface descriptions for a function in register new user page.

animateTextField
Allows for the textfield to update.

Login

The **loginViewController** is where the merchant logs in, the two fields required are the username and the password. If they are good then the user is allowed access to the rest of the application. This is also where when a good login the global variable is set to the current user's username.

Table 108: Module description for login page.

loginViewController
userText:UITextField passwordText:UITextField loginButton:UIButton
initWithNibName (id):self viewDidLoad (void):void didReceiveMemoryWarning (void):void viewDidUnload (void):void LoginButtonClicked (IBAction):void textFieldShouldReturn (bool):bool

Table 109: Interface description for a function in the login page.

initWithNibName
id : NSInteger : the id of the view

Returns a newly initialized view controller with the nib file in the specified bundle.

The function **viewDidLoad** sets up all the textfields and also sets up what keyboard and keyboard functions each text field has. For both textfields, **userText** and **passwordText** a normal keyboard is used with a done button to release the keyboard from the view.

Table 110: Interface description for a function in the login page.

viewDidLoad
The function connects to the database and depending on conditions logs a user in or alerts them on a bad login. Also loads the table view controller.

Table 111: Interface description for a function in the login page.

didReceiveMemoryWarning
Sent to the view controller when the app receives a memory warning.

Table 112: Interface description for a function in the login page.

viewDidUnload
Returns all resourced allocated to the view.

The **LoginButtonClicked** function is where both the UITextFields, **userText** and **passwordText** is passed to the database and if there is a match then the user is logged in and the global variable is set for the user's username. If there is not a match then a alert is triggered stating that the username and/or password is wrong and to try again.

Table 113: Interface description for a function in the login page.

LoginButtonClicked
The function connects to the database and depending on conditions logs a user in or alerts them on a bad login.

Table 114: Interface description for a function in the login page.

textFieldShouldReturn
textField : Bool : if the textfield should be selected or not.
Asks the delegate if the text field should process the pressing of the return button.

App Delegate Apple Built-In

Built-in apple class for application start-up and other actions that can happen to the application.

Table 115: Module description for built-in application for Apple.

capstoneAppDelegate
application(Bool):Bool applicationWillResignActive(void):void applicationDidEnterBackground(void):void applicationWillEnterForeground(void):void applicationDidBecomeActive(void):void applicationWillTerminate(void):void

Table 116: Interface description for a function in the built-in class.

application
Application : Bool : starting up or not.
Tells the delegate that the application is about to start up.

Table 117: Interface description for a function in the built-in class.

applicationWillResignActive
Tells the delegate that the application is about to become inactive.

Table 118: Interface description for a function in the built-in class.

applicationDidEnterBackground
Tells the delegate that the application is now in the background.

Table 119: Interface description for a function in the built-in class.

applicationWillEnterForeground
Tells the delegate that the application is about to enter the foreground.

Table 120: Interface description for a function in the built-in class.

applicationDidBecomeActive
Tells the delegate that the application has become active.

Table 121: Interface description for a function in the built-in class.

applicationWillTerminate
Tells the delegate when the application is about to terminate.

Merchant Android App

Login Page

Table 122: Module description for login page.

MerchantLocalizationActivity
inputUserName: EditText inputPassword: EditText
onCreate(Bundle) login()

Table 123: Interface description of XML file for login page.

main.xml
LinearLayout
Username: TextView Username: EditText Password: TextView Password: EditText Login: Button ForgotPassword: Button

Home Page

Table 124: Module description for home page.

Home
onCreate(Bundle)

Table 125: Interface description of XML file for home page.

home.xml
LinearLayout
ManageAds: Button Customers: Button LogOut: Button

List of Customers

Table 126: Module description for list of customers page

Customers
onCreate(Bundle) getData()

createList()

Table 127: Interface description of XML file for list of customers page.

customers.xml
LinearLayout
CustomerListLabel: TextView
CustomerList:ListView

Manage Ads

Table 128: Module description for manage ads page.

ManageAds
adsLayout: LinearLayout
onCreate(Bundle) getData()

Table 129: Interface description of XML file for manage ads page.

manageads.xml
LinearLayout
Advertisements: TextView AdList: ListView

Merchant Website

Table 130: Module description for merchant website.

Merchant Website
Username Password
HomePage () accountSettings (String, String, String, String, Date, Bool, String) createAd (int, Bool, String, String, String, Date) createCoupon (int, Bool, String, String, String, Date) viewCharges(String) viewNearbyCustomers () recentTransactions (Date) logout (String) : Bool login (String, String) : Bool createAccount (String, String, String, String, String, Bool) : Bool

Home Page

Table 131: Interface descriptions for home page.

Home Page
Session array
Displays the navigation menu for the merchant website.
menu : a list of links to the rest of the merchant website pages.

Account Settings

Table 132: Interface descriptions for account settings.

Account Settings
Username : string Password : string confirmPassword : string address : string birthday : date student : bool email : string
Update the row in the database associated with this user.
success : Bool : Indicates whether or not the update was successful. If anything goes wrong in the function, such as database connection issues, then it returns false.

Create Coupon

Table 133: Interface descriptions for create coupon page.

Create Coupon
targetAge: int targetGender: string targetTime: date studentTarget: bool title : string content : string
Submit a request for approval to MCM.

Create Ad

Table 134: Interface descriptions for create ads page.

Create Ad
targetAge: int

targetGender: string targetTime: date studentTarget: bool title : string content : string
Submit a request for approval to MCM.

View Charges

Table 135: Interface descriptions for view charges page.

View Charges
Username : String
Query the database for the charges associated with the user.
Display the list of charges.

View Nearby Customers

Table 136: Interface descriptions for view nearby customers page.

View Nearby Customers
Latitude : Int Longitude : Int
Query the database for the customers near the user.
Display list of nearby customers

Recent Transactions

Table 137: Interface descriptions for recent transactions page.

Recent Transactions
Username : String
Query the database for the customers near the user.
Display list of nearby customers

Logout

Table 138: Interface descriptions for logout page.

Logout
Username : String
Close the user's session and return to the login screen

Login

Table 139: Interface descriptions for login page.

Login
Username : string Password : string
Validate the login information with the database. If it is correct, call the homepage.

Create Account

Table 140: Interface descriptions for create account page.

Create Account
username : String : Requested username password : String : Requested password confirmPassword : String : Confirmation of requested password which has to match the password exactly email : String : Validated email of the user address : String : merchants primary location productTypes : enum : The list of product types associated with the business
The function takes the required parameters from the new user and sends a database query to add a new row to the database.
success : Bool : Indicates whether or not the registration was successful. If anything goes wrong in the function, such as database connection issues, then it returns false.

Forgot Login Info

Table 141: Interface descriptions for forgot login info page.

Forgot Login Info
Username : string Email : string
Validate the combination and send an email with the forgotten password.

Web Service

The web service provides an interface for information passed between different devices and the database. The functions within the web service are written to be compatible with Android, iOS, and websites written in PHP and HTML.

Table 142: Module description for the web service.

Web Service
DB_HOST: String DB_USER: String DB_PASSWORD: String DB_DATABASE: String GOOGLE_API_KEY: String
config ( ) connect ( ) : database handler close ( ) sendNotification (int, String, String) : String getCoupons (String, int, int) : String getNearbyCustomers (int, int) : String getNearbyMerchants (String) : String getUserHistory (String) : String registerNewUser (String, String, String, Date, Bool, String, String, int) : Bool chargeUser (String, double, String) updatePayment (String, Bool, Bool, int) updateUserLocation (String, int, int) validateLogin (String, String) : Bool

Configure Variables

Table 143: Interface description for the configuration function.

config
The function config links the variables of the web service to their actual values. The database host is the host storing the database, the database user and password are the variables used to log into the database, and the DB_DATABASE is the name of the actual database being used. This function also uses the GOOGLE_API_KEY, which is a unique key given to a project by Google to keep track of total Google API usage.

Connect with Database

Table 144: Interface description for the connect function.

connect
In the function connect, the web service establishes a connection with the database configured in config. This file is configurable to different databases, which would be necessary once the functions in this prototype are ported over to the real MCM database. This function returns a connection handle which can be used when making queries to the database.
handler : databaseHandler : Handle of connected database.

Close Database Connection

Table 145: Interface description for the close function.

close
In the function close, the web service closes the current database connection established in the connect function.

Send Notification

Table 146: Interface description for the send notification function.

sendNotification
type : int : Type of notification registrationId : String : Registration id of a specific device message : String : Message to be sent to the device
The function sendNotification sends a push notification service to a device. Depending on the type of device, which is passed as a parameter, different services are used. The Android push notification will be hosted by Google Cloud Messaging (GCM) service, and the iOS push notification will be hosted by the Apple Push Notification Service. The registration ID is a unique string associated with each device, and the message is the type of message that is going to be sent in the notification. The function results whether or not the push notification has been successful on the server side.
result : String : Result of the notification sending service.

Get Coupons

Table 147: Interface description for the get coupons function.

getCoupons
username : String : Username latitude : int : User's current latitude longitude : int : User's current longitude
This function retrieves a list of relevant coupons based on the user's current location as well as their localization user settings. The relevant coupons will query several database tables to retrieve deals that are currently nearby and most relevant to the user. The function will return a JSON string composed of the relevant deals queried from the database.
result : String : JSON string of relevant coupons

Get Nearby Customers

Table 148: Interface description for the get nearby customers function.

getNearbyCustomers
latitude : int : User's current latitude longitude : int : User's current longitude
The function getNearbyCustomers returns a list of customers within the perimeter of 0.10 km. The function uses a flat-earth approximation, which is accurate enough for a small distance such as merchant store perimeters. This function returns a JSON string of nearby customer usernames.
result : String : JSON string of nearby customers

Get Nearby Merchants

Table 149: Interface description for the get nearby merchants function.

getNearbyMerchants
username : String : Username
The function getNearbyMerchants returns a list of merchants within the perimeter of 0.20 km. This function also uses the flat-earth approximation to estimate a circular radius containing the merchants. The function returns a JSON string of nearby merchant usernames and their information.
result : String : JSON string of nearby merchants

Get User History

Table 150: Interface description for the get user history function.

getUserHistory
username : String : Username
The function getUserHistory takes in the username of the requester and returns a list of user history up to one week or ten transactions at a time, depending on which criteria returns less items. If the user requests a prolonged history, they are able to call this function again through the web service. This function returns a JSON string of user history.
result : String : JSON string of user history

Register New User

Table 151: Interface description for the register new user function.

registerNewUser
username : String : Username password : String : Requested password email : String : User's email address birthdate : String : User's birthdate student : Bool : Indicator for user's student status regId : String : User's device registration ID type : int : User's type of interface
The function registerNewUser takes in relevant parameters needed to register a new user into the database. It takes in the requested username, password, email, birthdate, and a Boolean for whether or not the customer is a student or not. It also takes in the device type and registration ID, which is a unique ID assigned to each device.
result : Bool : Indicates registration success

Charge User

Table 152: Interface description for the charge user function.

chargeUser
username : String : Username amount : Double : Amount being charged message : String : Details on the charge
The function chargeUser charges a customer by a specific amount. There is the option of creating messages, so the customer knows exactly what type of product is going to be purchased. This function uses the push notification service to notify the customer right after a charge request goes through the database.

Update Payment

Table 153: Interface description for the update payment function.

updatePayment
username : String : Username paid : Bool : Indicates if user approved the charge cancelled : Bool : Indicates if user cancelled the charge productIndex : int : Unique index assigned to each product
The function updatePayment updates the user response to the charge being made. Based on the customer's username and the product index that has been bought, the function updates whether the customer approved or cancelled the charge. This sends a notification to the merchant to confirm the customer reply to a recent charge.

Update User Location

Table 154: Interface description for the update user location function.

updateUserLocation
username : String : Username latitude : int : User's current latitude longitude : int : User's current longitude
The function updateUserLocation updates the user's current latitude and longitude in the database. The latitude and longitude is sent as a length-six integer, where the actual latitude and longitude values have been multiplied by one million. This is going to update the database, which will affect the functions getNearbyMerchants, getNearbyCustomers, and getCoupons.

Validate Login

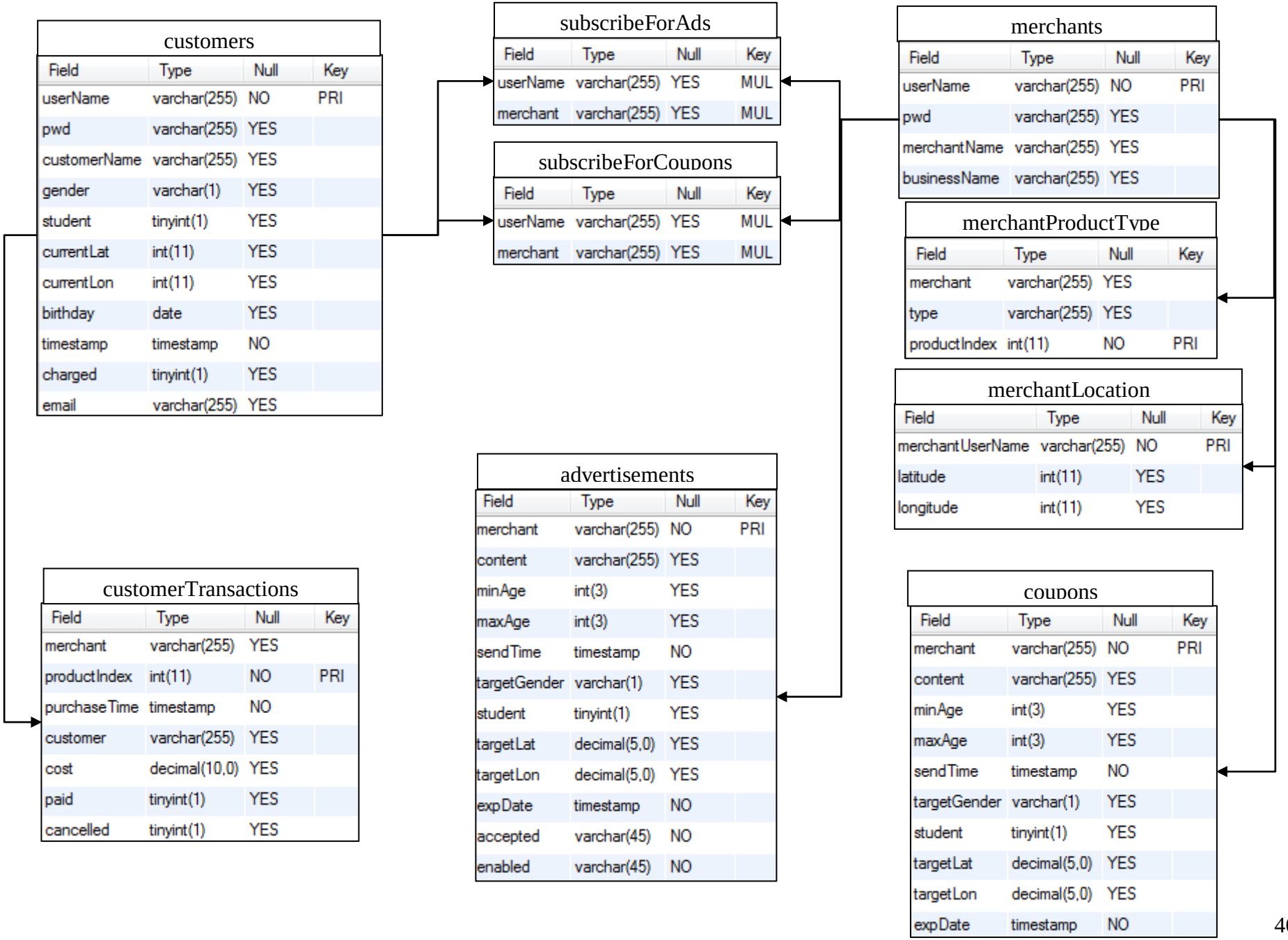
Table 155: Interface description for the login validation function.

validateLogin
username : String : Username, case-sensitive password : String : Password, case-sensitive
The function validateLogin checks the input username and password to the customer information stored in the database. It returns true if the username and the password pairs match exactly in a case-sensitive manner. It returns false if the username and password do not match, the username does not exist in the list of registered users, or the username and password cases don't match exactly as stored.
valid : Bool : Indicates if the username/password match exists.

Database

The database has nine tables to hold the necessary customer and merchant information:

1. Customers
2. Merchants
3. subscribeForAds
4. subscribeForCoupons
5. merchantLocation
6. advertisements
7. coupons
8. customerTransactions
9. merchantProductType


customers
Field Type Null Key
userName varchar(255) NO PRI
pwd varchar(255) YES
customerName varchar(255) YES
gender varchar(1) YES
student tinyint(1) YES
currentLat int(11) YES
currentLon int(11) YES
birthday date YES
timestamp timestamp NO
charged tinyint(1) YES
email varchar(255) YES
subscribeForAds
Field Type Null Key
userName varchar(255) YES MUL
merchant varchar(255) YES MUL
subscribeForCoupons
Field Type Null Key
userName varchar(255) YES MUL
merchant varchar(255) YES MUL
merchants
Field Type Null Key
userName varchar(255) NO PRI
pwd varchar(255) YES
merchantName varchar(255) YES
businessName varchar(255) YES
merchantProductType
Field Type Null Key
merchant varchar(255) YES
type varchar(255) YES
productIndex int(11) NO PRI
merchantLocation
Field Type Null Key
merchantUserName varchar(255) NO PRI
latitude int(11) YES
longitude int(11) YES
advertisements
Field Type Null Key
merchant varchar(255) NO PRI
content varchar(255) YES
minAge int(3) YES
maxAge int(3) YES
sendTime timestamp NO
targetGender varchar(1) YES
student tinyint(1) YES
targetLat decimal(5,0) YES
targetLon decimal(5,0) YES
expDate timestamp NO
accepted varchar(45) NO
enabled varchar(45) NO
customerTransactions
Field Type Null Key
merchant varchar(255) YES
productIndex int(11) NO PRI
purchaseTime timestamp NO
customer varchar(255) YES
cost decimal(10,0) YES
paid tinyint(1) YES
cancelled tinyint(1) YES
coupons
Field Type Null Key
merchant varchar(255) NO PRI
content varchar(255) YES
minAge int(3) YES
maxAge int(3) YES
sendTime timestamp NO
targetGender varchar(1) YES
student tinyint(1) YES
targetLat decimal(5,0) YES
targetLon decimal(5,0) YES
expDate timestamp NO

The customer table holds all of the user information necessary for the ad and coupon targeting functionality. The timestamp field holds the time of the most recent location update. The charged field is a temporary field that we used for testing.

Table 156: Customer Table in Database

Field	Type	Null	Key
userName	varchar(255)	NO	PRI
pwd	varchar(255)	YES	
customerName	varchar(255)	YES	
gender	varchar(1)	YES	
student	tinyint(1)	YES	
currentLat	int(11)	YES	
currentLon	int(11)	YES	
birthday	date	YES	
timestamp	timestamp	NO	
charged	tinyint(1)	YES	
email	varchar(255)	YES	

The merchant table holds the merchant user name and password as well as the merchant name and a more specific business name (i.e. MCM and Hermes Commerce Inc.). The user name serves as the table's primary key. The business location is stored in another table because businesses may have multiple locations.

Table 157: Merchant fields, types, and keys.

Field	Type	Null	Key
userName	varchar(255)	NO	PRI
pwd	varchar(255)	YES	
merchantName	varchar(255)	YES	
businessName	varchar(255)	YES	

The subscribeForAds table holds the merchants that each user wants to get ads from.

Table 158: subscribeForAds fields, types, and keys.

Field	Type	Null	Key
userName	varchar(255)	YES	MUL
merchant	varchar(255)	YES	MUL

The subscribeForCoupons table holds the merchants that each user wants to get coupons from.

Table 159: subscribeForCoupons fields, types, and keys.

Field	Type	Null	Key
userName	varchar(255)	YES	MUL
merchant	varchar(255)	YES	MUL

The customerTansactions table holds all of the transactions for each customer. The primary key is the customer.

Table 160: customerTransactions fields, types, and keys.

Field	Type	Null	Key
merchant	varchar(255)	YES	
productIndex	int(11)	NO	PRI
purchaseTime	timestamp	NO	
customer	varchar(255)	YES	
cost	decimal(10,0)	YES	
paid	tinyint(1)	YES	
cancelled	tinyint(1)	YES	

The merchantProductType table lists all of the product types of the different merchants and has an index for each type as the primary key.

Table 161: merchantProductType fields, types, and keys.

Field	Type	Null	Key
merchant	varchar(255)	YES	
type	varchar(255)	YES	
productIndex	int(11)	NO	PRI

The coupons table stores all of the coupons as well as the information needed for direct advertising.

Table 162: coupon fields, types, and keys.

Field	Type	Null	Key
merchant	varchar(255)	NO	PRI
content	varchar(255)	YES	
minAge	int(3)	YES	
maxAge	int(3)	YES	
sendTime	timestamp	NO	
targetGender	varchar(1)	YES	
student	tinyint(1)	YES	
targetLat	decimal(5,0)	YES	
targetLon	decimal(5,0)	YES	
expDate	timestamp	NO	

The advertisements table stores all of the ads as well as the information needed for direct advertising.

Table 163: advertisement fields, types, and keys.

Field	Type	Null	Key
merchant	varchar(255)	NO	PRI
content	varchar(255)	YES	
minAge	int(3)	YES	
maxAge	int(3)	YES	
sendTime	timestamp	NO	
targetGender	varchar(1)	YES	
student	tinyint(1)	YES	
targetLat	decimal(5,0)	YES	
targetLon	decimal(5,0)	YES	
expDate	timestamp	NO	
accepted	varchar(45)	NO	
enabled	varchar(45)	NO	

The merchantLocation table stores the different addresses of the merchants. This is a separate table because it is possible for one business to have two or more locations.

Table 164: merchantLocation fields, types, and keys.

Field	Type	Null	Key
merchantUserName	varchar(255)	NO	PRI
latitude	int(11)	YES	
longitude	int(11)	YES	

Technical Project Plan

The technical project plan is based on smaller components of the architecture. The Gantt chart can be seen in Figure 7.

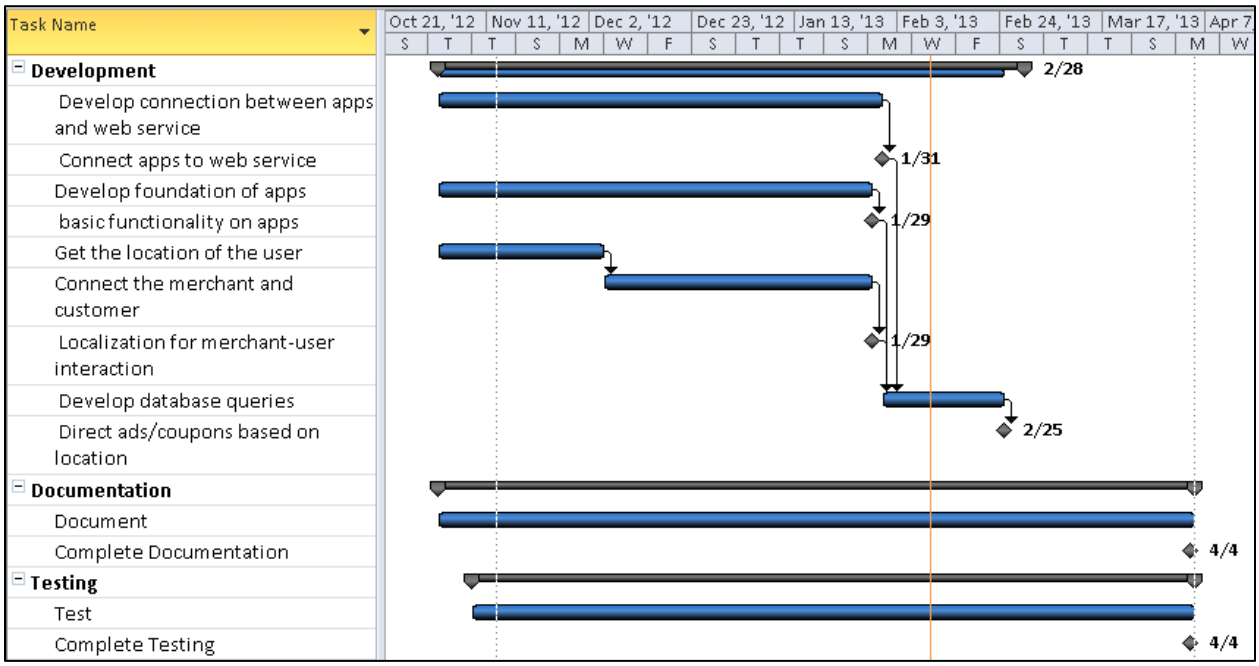


Figure 7: Gantt Chart of scheduled timeline.