Team 23 Design Document

Customer Loyalty Program for Small Businesses

Clients - Jay Namboor
Adviser - Dr. Govindarasu

Members:
Christopher Waters
Van Nguyen
William Tran

Contents

System Requirements

Functional Requirements

FR-1	The system shall allow the user to login and logout.
FR-2	The system shall allow the user to create an account
FR-3	The system shall allow merchants to create loyalty cards
FR-4	The system shall allow merchants to expire existing loyalty cards
FR-5	The system shall allow merchants to verify customer punches
FR-6	The system shall allow merchants to verify customer rewards
FR-7	The system shall allow customers to search for local businesses
FR-8	The system shall allow customers to select loyalty cards
FR-9	The system shall allow customers to download loyalty cards
FR-10	The system shall allow customers to redeem punches
FR-11	The system shall allow customers to redeem rewards
FR-12	The system shall allow the database to store a history of completed punch cards
FR-13	The system shall allow the merchant to view card history for their business
	The system shall allow the customer to view personal card history for completed rewards

Non-functional requirements

NFR-1	The system shall be developed using the Appcelerator SDK
NFR-2	The system shall use an encrypted SQLite database for local storage to prevent other applications from interacting with a user's card data
NFR-3	The system shall utilize hashing for punch a reward verification

Functional Decomposition

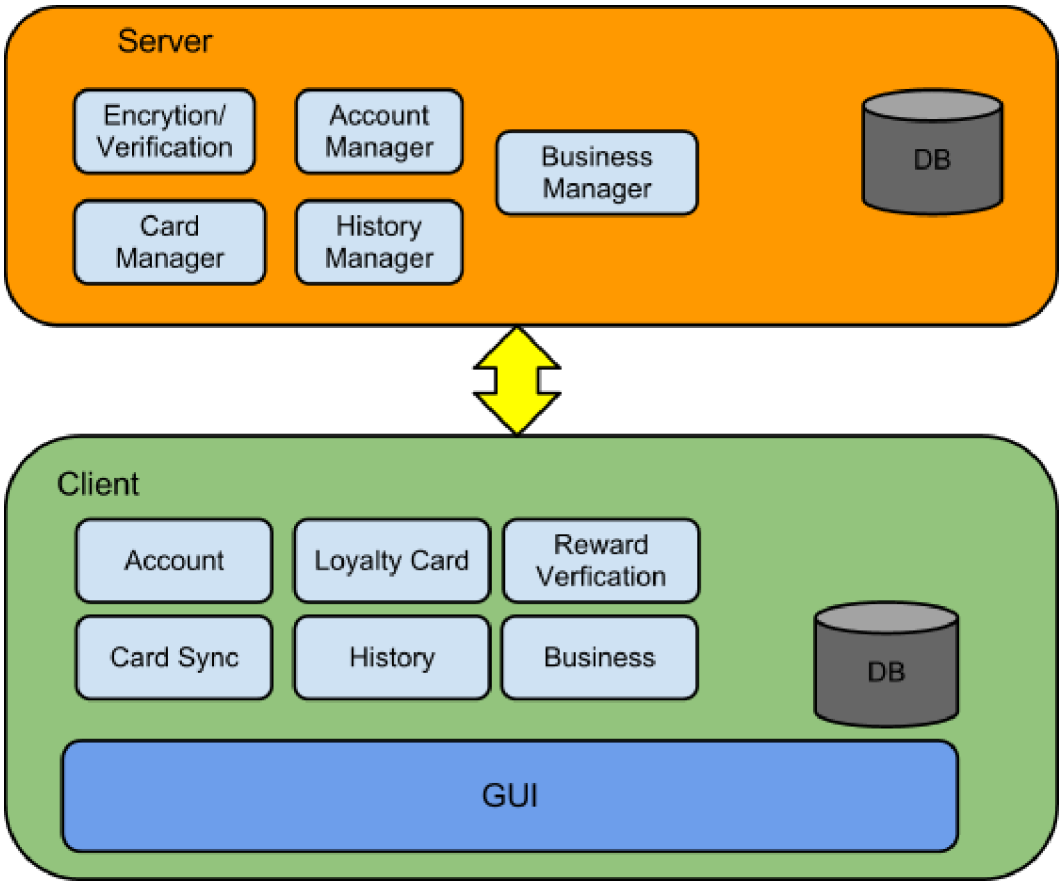


Figure 1 - Above shows the client and server sides interact.

System Analysis

While our system is mainly a client-server architecture, it is however, somewhat unique in that both the server and client side will house some of its own data. The reason for this is to provide some offline functionality. The offline functionality is needed when the user is “punching” the card. If we were to depend on an internet connection, that may slow the checkout process for the customer and the merchant. We do not want the application to cause the business to lose customers because it is trying to connect to the internet.

That is why cards can be downloaded on to the client locally. The client itself will be responsible for keeping track of the punch cards. It will handle everything from punching the card to redeeming the reward.

Because of the offline capability, we proposed a cloud synchronization solution. If the user were to ever lose their phone, they would not lose all of their punch cards. The Card sync module will interact with the History Manager on the server side to ensure data on both the client and the server are up to date.

Server Side

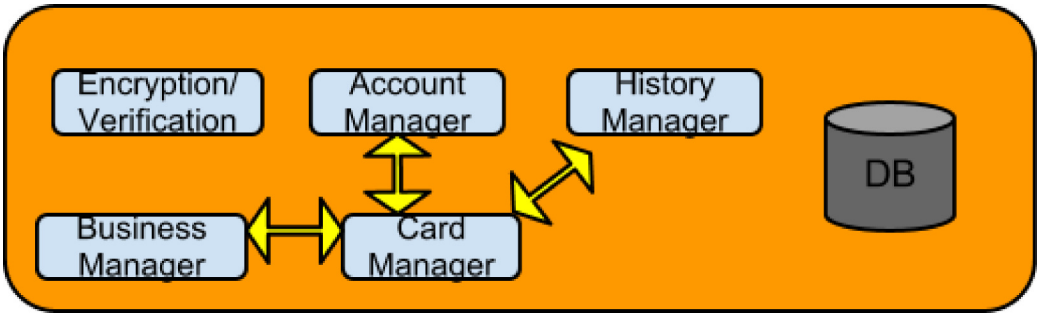


Figure 2 - Notice how every module within the same layer can interact with one another. Same goes for the client layer. For example if Business Manager needs to get the card that is owned by it, it can call on the card manager to get that card from the database.

The responsibilities of the server include storing all of the user and the businesses. The way each module is setup is based on their specific responsibilities. For example if a user wants to download a card, it will eventually have to go through the card manager to get that card. For more information on a module, please refer to the input/output specifications.

Interaction between client and server Side

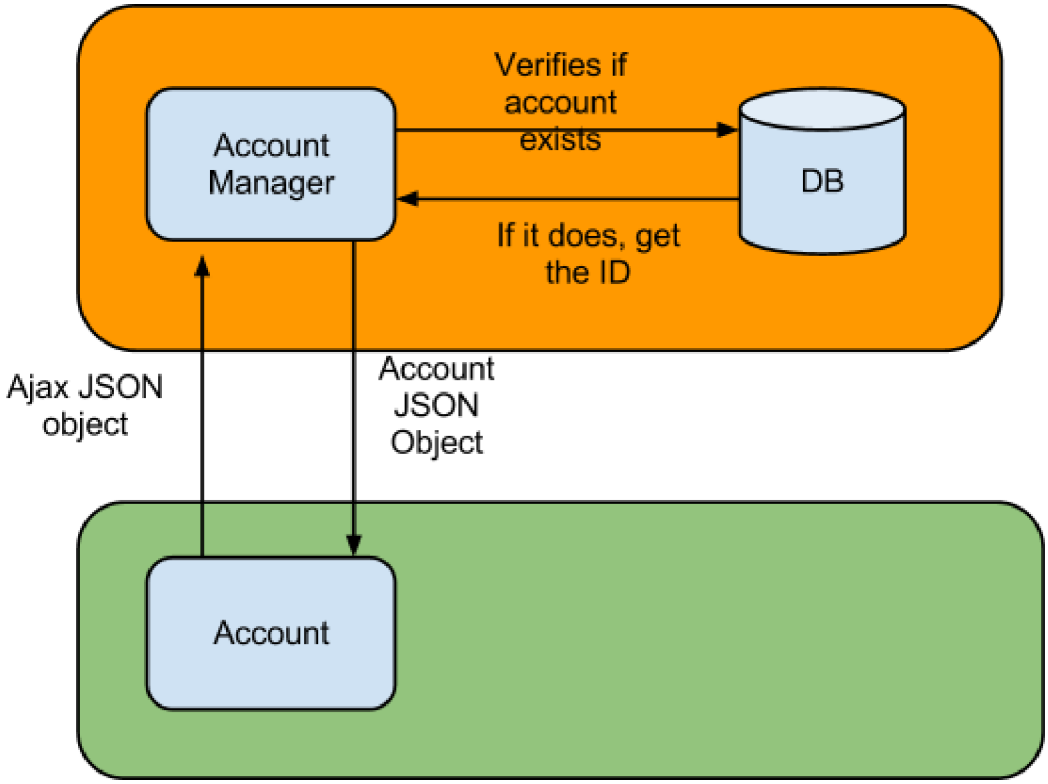


Figure 3 - Shows interaction between server and client

Interactions between client and server will done through JSON.In the example above, a user is logging in. The account module on the client side will ajax a GET request with an JSON param object { action : <method> , object : <data>} to the Account Manager on the server side. The action tells which method to call from the controller, and the data in this case is an Account Object with a user name and password. The Account Manager will parse out the Account object and verify if it exists in the database. If it does, the Account Manager will return an Account Object with a id > 1 and -1 otherwise. All communication with the server will be done in a similar fashion.

Software Design Patterns

1. Model–view–presenter

We are using the model-view-presenter design pattern in order to separate logic from the view from the model. A picture below describes the mvp structure.

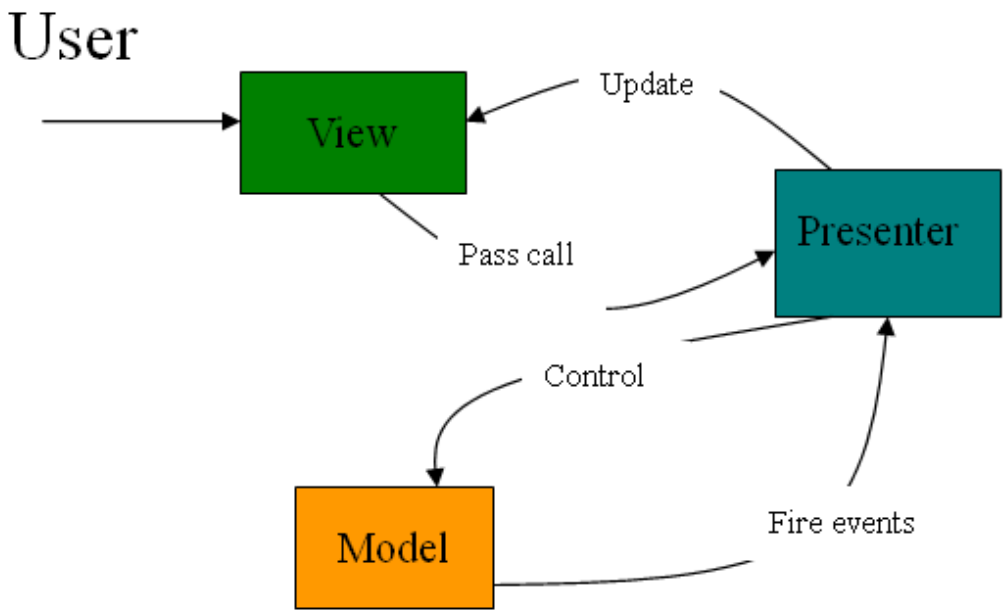


Figure 4 - Shows the MVP pattern (Source : http://dotnetslackers.com/articles/silverlight/A-WaspKiller-Game-with-Silverlight-3-NET-RIA-Services-MVP-and-MVVM-Patterns-Part-1.aspx)

In this case, the view is all of our UI. Whenever their UI needs to update/request information, they will need to call a specific controller(presenter) assigned for handling certain jobs. That controller will then update/retrieve the data and do any logic/format changes to the data before it is passed back to the View. The Model 's only job is to update/retrieve data. The Views job is only to display data. The controller will handle all the logic. Once again because of the client-server architecture along with the offline capability, we almost have two-part controllers in this case. Think about this, when the user attempts to create an account, the GUI will need to interact with the account controller on the client side. The account controller on the client side will then quickly do input checking ...etc , before sending the data to the Account manager controller on the server side to store and verify the account has been created.

Input/Output Module Specification

Server Side

Each input for the modules shall be wrapped around a JSON Object and converted to a string before it is sent over to a server in this format.

JSON Object - Which contains the data .
For example account object would have
object = { username : <value> , password <value> }

Which is then wrapped around another object to be sent over with another action attribute.
The action key specifies which function to call within the controller.
param = { action : <value> , object <data object> }

and finally this is passed to a ajax call by { param : Stringify(param) }

Example : A call to the Account Manager Controller for a login

```
var account = { username : "hello" , password : "goodbye"};
        var param = { action : 'Login', object : account };
        $.ajax({
                type: 'GET',
                url : 'http://localhost:81/loyaltyprogram/AccountManager.php',
                data : { param  : JSON.stringify(param)},
                success : function ( data){
                        alert(data);
                }
        });
```

Card Manager
The card manager module will handle creation of cards and also retrieving cards.

Module Function	Input	Output
Store created loyalty card onto the server	Title/Description/Number of punches/Level/Expiration Date/Place ID	Verification that loyalty card is created and added it to the server

Retrieving loyalty cards from server	Place ID	List of loyalty cards information that match with the Place ID

Test Case	Success	Fail
User(merchant) Create loyalty cards for their business	Loyalty card does not already exist for specific business in the database.	Loyalty card already existed in the database
User(Customer) Selected loyalty card from specific business	Loyalty card is not expired and it placeID match with specific	Loyalty card exists or placeID does not match with specific business ID.

Account Manager

The account manager will contain code to create, lookup accounts, & reset passwords. When creating an account, users will input an email and password combination. If a user already has an account, they can login by submitting their email and password combination.

Module Function	Input	Output
User login	Email/Password	Verification that a user is logged in (*Session?)*
User creates account	Email/Password	Stores account in database and notifies user.
User resets password	Reset password	Password is reset and sent to user's username

Test Case	Success	Fail
User creates account.	email does not already exist in the database.	email already exists in the database
User login	email and password combination exists in the user database	email and password combination does not exist in the user
User resets password	user password resets	

History Manager

The History Manager will be responsible for maintaining a history of cards that have been downloaded to a user's phone. When the card is redeemed, the database entry corresponding to that card will be marked as completed. It will also be able to sync up with a user's phone to keep track of partially complete cards so a user can download their cards in case they lose access to their phone (e.g. phone breaks and they get a replacement phone or they upgrade phones).

Module Function	**Input**	**Output**
Add Card	Card Object	Confirmation Card was Added
Update Card	Card Object	Confirmation Card was Updated
Retrieve Card	*Account Object (id)*	Uncompleted Cards
View History	Search Criteria ID Search by Date Creation Search by last punch Search by Business id	Cards Matching Search Criteria

Test Case	**Success**	**Fail**
Add Card	Card is added to Database	Card is not added to Database
Update Card	Card is updated in Database	Card is not updated in Database
Retrieve Card	Uncompleted cards are downloaded to a user's phone	Uncompleted cards are not downloaded to a user's phone
View History	User receives a list of cards matching search criteria	User does not receive a list of cards matching search criteria

Business Manager

The business manager module will handle creating and linking up business to users. It will also handle request for anything information involving businesses.

This module must :

1. Create a business and link up to user account

2. Link up other accounts to business

3. Retrieve businesses of a user

4. Search request for business based on name

10 | Loyalty Program

5. Search for businesses based on location

Module Function	Input	Output
Create a business and link up to user account, and generate qrcode and hash to business and email qr code to user.	UserID/Place Name/Location	Verification that business is created and linked up to an account with qrcode created. (Business JSON with id) -1 -business already exists
Save an edited business	Business info	Verification that business information has changed
Link up other accounts to business	UserIDs/Place/BusinessID	Verification that the business is linked up to those accounts
Retrieve businesses of a user	UserID	List of business information linked up to the UserID
Search request for business based on name	Place name	List of businesses matching with place name with locations.
Search request for business based on location	GPS location	List of businesses near GPS location.

Test Case	Success	Fail
User(Merchant) Creates a business	Business must be validated before it is created. User will be linked up to business Qrcode is generated and emailed to user.	Business creation request is declined.
Merchant edits and saves a business	Business information must changed	Business creation request is declined.
User(Merchant) Links up other users to help manage business.	User has the role(linked to business already) to manage the business.	User does not have the role(linked to business already) to manage the business.
User(Merchant) retrieves	User is linked to any	User is not linked to any

11 | Loyalty Program

his/her businesses to manage.	businesses.	businesses.
User Search request for business based on name	Business matches that name with in the database.	Business does not match the name with in the database.
User searches for business based on a his/her location.	There are businesses within a range of that location	There does not exists not any businesses with a range of that location.

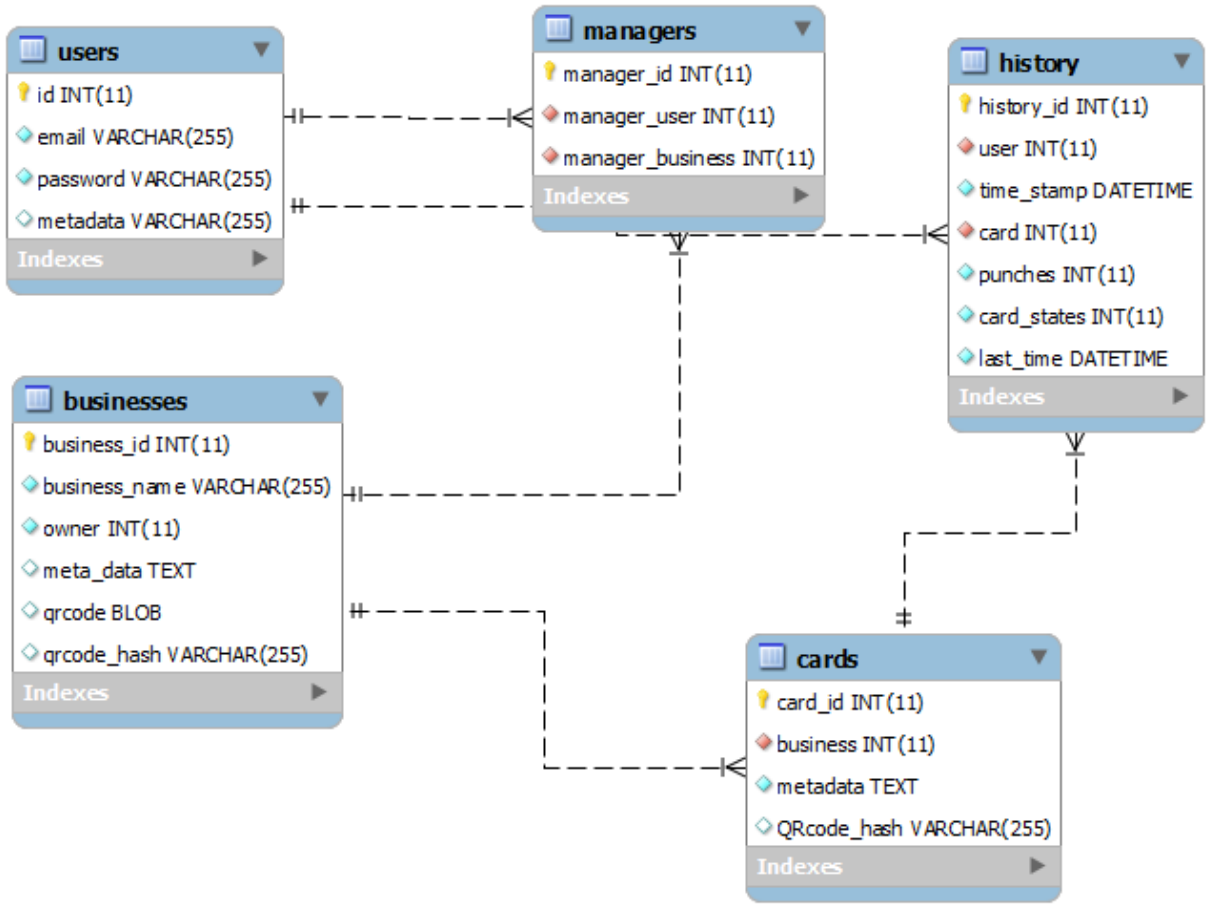
Encryption/Verification

This module is responsible for hashing (sha-256) of input from the user such as passwords. It will also have the responsibility of generating a QR code (with http://phpqrcode.sourceforge.net/) for a specific card for a merchant to do punches.

Module Function	**Input**	**Output**
Hash a user password	User password	Hashed (password)
Generate a QR code	Merchant Name + Merchant ID	QR code of hashed Merchant Name + Merchant ID (salt) = merchantId + name + id

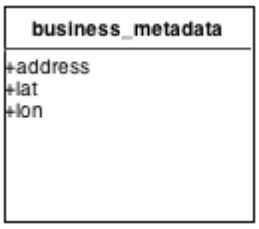
Test Case	**Success**	**Fail**
User password input	Password is hashed	none
Generate QR code	QR code is generated from hashed(Merchant Name+Merchant Id)	none

Database



**Each Database will have a corresponding object relating to it.*

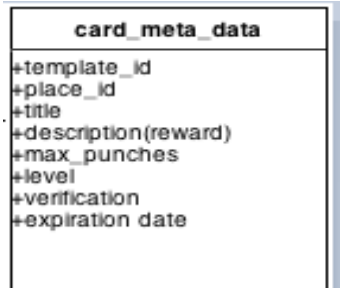
Business_meta_data JSON



address- address of the business
lat - latitude location of the business
lon - longitude location of the business

Card_meta_data JSON

13 | Loyalty Program



- template_id - the id of the downloaded card
- place_id - the id of the card's business
- title - title of the card
- description - description of the reward
- max_punches - max number of punches needed to redeem a reward
- level - level of the card
- verification - the verification code for this card
- expiration Date - the expiration date of this card

Client Side

Account

The account module will be responsible for the login/logout process of the user. It will also handle the creation of accounts.

1. Login
2. Logout
3. Create Accounts

Module Function	Input	Output
User login	username and password	Verification if user is successful or not, and open a session to the user
User logout	user clicks logout	logs user out by closing a session to the user
Create accounts	email, password, and confirm password	send information to server side to create an account

Test Case	Success	Fail
User log into the system	username and password match with username and	mismatching username and password

	password on the server	
User logout of the system	the system close user session	user session still run
Create new account	email and password doesn’t exist on the server	email and password exist on the server

Loyalty Card

This module will be use when the customer downloads a selected loyalty card. Once the download card is complete, it will be store onto the phone database. The module will then be required to handle punches of the a card. It will also mark the card is completed when punches have been filled.

Cards have have only have one of four states

1. Not completed - 0
2. Completed - 1
3. Redeemed -2
4. Expired - 3

Module Function	Input	Output
Download Loyalty Card	Card Template ID	Download the new loyalty card and store it the phone
Punches card	QR code and timestamp of last punch.	Loyalty point to the card and update the phone database if QR code matches.
Last punch of card	QR code,	Update history module and and mark card as completed in database.
Get current loyalty cards	Selected loyalty card	Gets a list of current loyalty cards.
Get completed loyalty cards	Selected completed loyalty card	Gets a completed list of loyalty cards.

Test Case	Success	Fail
QR code verification	hashed value of QR code matches the one in the database with the specified card id.	hashed value of QR code doesn't match the one in the database.

Reward Verification

This module is responsible for the verification of a punch. When the user scans a QR code, this module check the QR code's value against the one in the database.

Module Function	Input	Output
Scan and Verify QR code	Card id, hashed value from QR code	Success or fail

Test Case	Success	Fail
Scan QR code verification	hashed value of QR code matches the one in the database with the specified card id.	hashed value of QR code doesn't match the one in the database.

Business

The business module will mainly interact with the business manager module on the server side to get requested information based on a search criteria.

Module Function	Input	Output
Create a business	UserID, Place name, and location	Verification that business is created and linked up to an account
Retrieve businesses of a user	UserID	List of business information linked up to the UserID
Search request for business	Place name	List of businesses matching with place name with locations

Test Case	Success	Fail
User(merchant) creates a business	Business must be validated before it is created. User will be linked up to business	Business creation request is declined
User(merchant) links up other users to help manage businesses	User has the role to manage the business	User does not have the rule to manage the business
User(merchant) retrieves his/her businesses to manage	User is linked to any businesses	User is not linked to any businesses
User search request for business based on name	Business matched that name within the database	Business does not match the name within the database

History

The history module will interact with the history manager on the server to add and update cards stored in the server's database, download non-completed cards, and view a user's history.

Module Function	Input	Output
Add Card	Card Object	Confirmation Card was Added
Update Card	Card Object	Confirmation Card was Updated
Retrieve Card	*<none>*	Uncompleted Cards
View History	Search Criteria	Cards Matching Search Criteria

Test Case	Success	Fail
Add Card	Card is added to Database	Card is not added to Database
Update Card	Card is updated in Database	Card is not updated in Database
Retrieve Card	Uncompleted cards are downloaded to a user's phone	Uncompleted cards are not downloaded to a user's phone
View History	User receives a list of cards matching search criteria	User does not receive a list of cards matching search criteria

Client Database

The client database will consist of :

loyalty card table

- time_stamp - When the card was downloaded
- card_meta_data - refer below
- punches - Number of punches a card (Default of 0)
- last_punched - timestamp of last punch
- card_state
  0 - not complete
  1 - completed
  2 - reward redeemed
  3 - expired

card_meta_data will be a string version of this JSON object - explained in server database part

Card Sync

This module shall be responsible for keeping the data on the phone and the server up to date. That way, if the user were to switch to a new phone, the application should automatically download the information they had when using the old phone to their new phone. To accomplish this the module must:

1. Keep server up to date with local information.
2. It must also handle updating even if there is no connection to the internet.
3. Check at login to see if local information matches with server. (i.e. check number of current cards and number of completed cards match). If it doesn't match, it must download information from the server.

4. Must update when:
    - When they complete a card.
    - When they punch a card.
    - When they redeem a reward.
    - When they login.
    - When they log out.
        - -If not internet request to them to save info
    - When they close the program.
    - When they download a card.

Module Function	Input	Output
Update server with local information	User ID, # of punches, Timestamp of Last Punch(YYYY:MM:DD hh:mm:ss) and, reward claimed (True/False)	Send request to history manager with punch card transaction information
Update even without internet connection	Current and previous punch card transactions.	Send request to history manager with punch card transaction information
Check for matching information with server at login	Current cards	Send over local information such as number of current cards and number of completed cards

Test Case	Success	Fail
Update server with local	Punch card transaction is	Punch card transaction is

information	logged on to server	not logged onto server
Update event without internet connection	card transactions saved locally when there is no internet, all local saved transactions will be log onto server database when there is a internet connection	card transactions not saved locally without internet, or all local saved transactions is not logged onto server database with internet connection
Check for matching information with server at login	Local information from phone and server matches	mismatching information

22 | Loyalty Program

User interface specification

This section will show all the different mock ups for each screen.

Shows the login page for a user when they first open the application

The user may create an account if they are not a user.

First time login verification - Users can go ahead and create a business to link

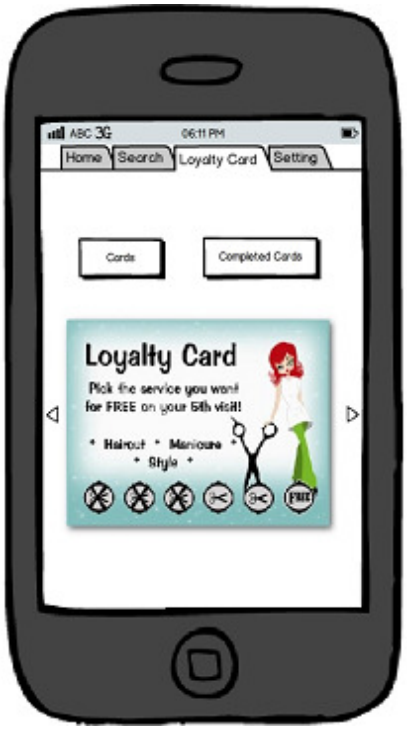
.

If they are a merchant they will proceed to this screen.

24 | Loyalty Program

Users will be shown a homescreen once they are logged in. The home screen can contain featured ads as well as various other tabs.

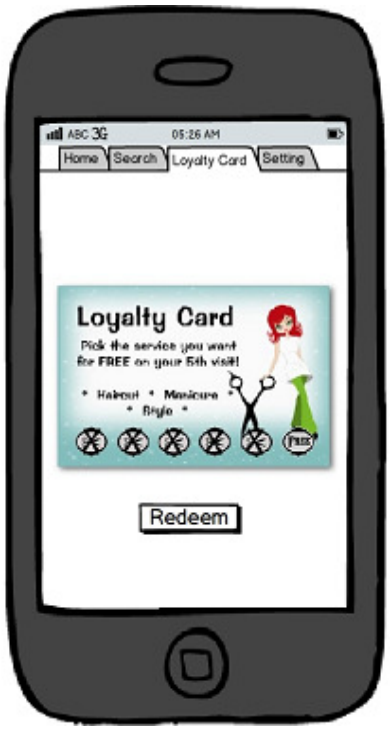
Search for businesses



This screen will allow the user to select a card they want to use.

Once selected, the user can punch the card by clicking on collect.

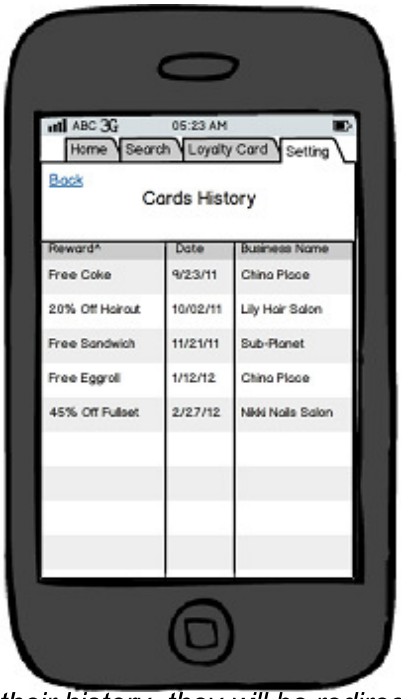
When the user hits collect, the QR scanner will automatically pop up. If it is successful, the card will get punched.



When the card is full, the user may try to redeem the reward once again by QR code.

The QR scanner will once again pop up for redeeming a reward.

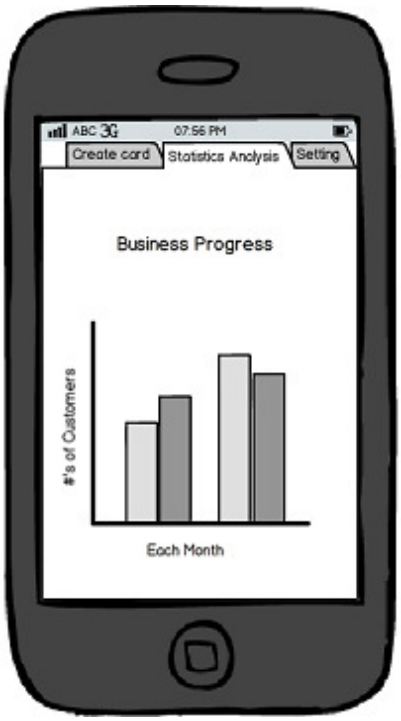
The setting page will allow the user to view previous completed cards or change email/password. (Chris will change)



If the user wants to view their history, they will be redirected to this screen.

This screen will allow the user to change their email/password. (Chris)

A merchant will have different options than a regular user. This screen shows how they can create a punch card. (Chris)



A merchant can also see track their progress each month based on the number of purchases, rewards redeemed … etc.

Software Specification

Technologies

Languages: PHP, JavaScript, MySQL, and sqlite.
Libraries: php qr code, phpMailer,JSON
SDK: Appcelerator Titanium

Server : Apache, MySQL, Amazon EC2

Appcelerator Titanium SDK
Using appcelerator, we are able to code once yet have it available for multiple platforms such as android and iOS. We write our client side code in JavaScript and it is compiled to native android and iOS code.

System Test Plan

Mainly done through use case scenarios

Use Case: UC1*-Create Account*
Trigger: *User requests to create an account*
Precondition: *None*
Interested Stakeholder: *Customers and merchants*
Actor: *Customers and merchants*
1. *The user will have to input their email, password, and confirmation password.*
2. *When they hit sign up, their name and password will be logged into the database.*

E1.1 The account is already existed in the database
E1.1 The error message will be displayed to the user that the account is already existed.
Output: *New account is created and it's going to be added to the system.*

Use Case: UC 2*-Login/Logout*
Trigger: *User start the application*
Precondition: *User must have signed up for an account.*
Interested Stakeholder: *homeowner and merchant*
Actor: *homeowner and merchant*
1. *The application will prompt the user to a login screen to enter their email and*

password.
2. The user will then hit login, which will then verify if their email and password exists in the database.
E2.1 Name and password combination do not exist
E2.1 The error message will be displayed to the user that login information is incorrect.
Output: Login information is verified and the application is displayed to the user.

Use case: UC 3-Create loyalty card
Trigger: Merchant hit the create card button
Precondition: Must be login
Interested Stakeholder: Merchant
Actor: Merchant
1. The application will prompt the merchant to a create card screen where they have to enter their new card information.
2. The merchant will then hit create, the system will then verify if their new card information exists in the database.
E3.1 Loyalty card already existed in the database.
E3.1 The error message will be displayed to the merchant that this card is already existed.
Output: New loyalty card is created/edited into system.

Use Case: UC 4-Expire loyalty card
Trigger: Merchant edit their loyalty card
Precondition: Must be logged in.
Interested Stakeholder: Merchant
Actor: Merchant
1. The merchant selected loyalty they want to edit.
2. The merchant update the expiration date for the selected card.
Output: Loyalty card is no longer valid.

Use Case: UC 5-Verify Punches
Trigger: Merchant scans QR Code
Precondition: Customer has downloaded card.
Interested Stakeholder: Merchant
Actor: Merchant
1. The customer gives phone to merchant to scan QR code
2. The merchant scans the QR code
Output: A punch is recorded for the loyalty card

Use Case: UC 6-Verify Reward
Trigger: Merchant scans QR Code

Precondition: *Customer has downloaded card and has the required number of punches*
Interested Stakeholder: *Merchant*
Actor: *Merchant*
1. The customer gives phone to merchant to verify the customer has earned the reward
2. The merchant scans the QR code
Output: *Loyalty card is marked as completed, no longer redeemable*

Use Case: UC 7-*Search Businesses*
Trigger: *Customer enters search criteria*
Precondition: *None*
Interested Stakeholder: *Customer*
Actor: *Customer*
1. The customer specifies search criteria
2. Businesses matching search criteria are displayed
Output: *List of businesses*

Use Case: UC 8-*Select a card*
Trigger: *Customer finds a card on a business's page*
Precondition: *Must be logged in.*
Interested Stakeholder: *Customer*
Actor: *Customer*
1. The customer selects a card from a business's page
2. Information about the card is displayed
Output: *Card information*

Use Case: UC 9-*Download Card*
Trigger: *Customer presses a button to download card*
Precondition: *Must be logged in, have a card selected*
Interested Stakeholder: *Card*
Actor: *Card*
1. The customer presses a 'download card' button
2. Card information is downloaded from server and saved on the phone
Output: *Card information*

Use case : UC 10-*Redeem punch*
Trigger : *User punches card*
Precondition : *Card must not be completed or expired.*
Actors : *User, local database, server database.*
1. *The hash value of the qr code is compared with the card's local hash value.*
E1 *Hash value does not match, card is not updated with punch.*
E1 *User is notified qr code is not valid.*
3. *Card's number of punches is incremented .*
A1 *If card's number of punches matches the max punches assigned to card, card is*

changed to a completed state.

4. *Update local database.*
5. *Update server data base if internet is available.*
Outcome : *User's punch has been verified,card is updated with a new punch, and card is updated in server side database if there is internet.*

Use case : *UC-11 Redeem reward*
Trigger : *User collects reward*
Precondition : *Card must be in a completed state and not expired*
Actors : *User, local database, server database.*
1. *The hash value of the qr code is compared with the card's local hash value.*
E1 *Hash value does not match.*
E1 *User is notified qr code is not valid and no reward is given.*
2. *Qr Code is verified and card is transitioned to a completed state.*
3. *Update local database.*
4. *Update server data base if internet is available.*
Outcome : *User has redeemed reward, information is updated locally and on the server.*

Use case : *UC-12 Store history*
Trigger : *Card information has been updated*
Precondition :
Actors : *Server database.*
1. *Take new card information and updated it in the server database.*
Outcome : *Card information is updated in the server side database, allowing for merchants to see user history.*

Use case : *UC-13 Merchant query history*
Trigger : *Merchant requests user history*
Precondition :
Actors : *Merchant,Server database*
1. *Return users and their information of whom are using merchant's cards.*
Outcome : *Merchant will have a list of users who currently have downloaded their cards.*

Use case : *UC-14 Customer query history*
Trigger : *User requests user history*
Precondition :
Actors : *User*
1. *Return list of completed cards for this user.*
Outcome : *User's will be able to view a list of completed cards for him/her self.*