PAyTEL

Marsol Daman, Lauren Lingeman, Axel Van Hoyweghen,
Alexander Dapoz, Anthony Calandra , Noah McGivern

CSI-4999-Senior-Design

Fall 2018

1. Project Vision

1.1. Backgrounds

PAyTEL was originally thought of by the predefined project description given to us by Professor Patel. Here is the background is given in the project description: “Credit card fraud isn’t new to any of us. Fraudulent use of credit cards is at all time high with the growth of cyber retail of goods and services where identity check of the buyer is an issue. To cope with this, credit card issuers are constantly in search of better methods of identity check for an online retail transaction. One recent addition to this system is SMS message based code verification as an extra step for verifying buyer’s identity. In this project you will design a new system for Secure Transaction. The system consists of three pieces, a web portal that manages all transactions, a buyer mobile app for approval of the transaction and a seller mobile app for issuing a charge. The front-facing mobile phone camera will be used for authenticating the biometric identity of the buyer when approving the transaction. The server will mediate the messaging, manage user profile and transactions.”

PAyTEL is a multichannel payment system for secure transactions for buyers and sellers.

1.2. Socio-economic Impact, Business Objectives, and Gap Analysis

The Socio-economic Impact for PAyTEL is going to impact the buyers and sellers in the economic marketplace and allow merchants and consumers to check out and pay per transaction. It will impact the mobile app payment marketplace and allow more secure banking for mobile applications for merchants and consumers. Our business objectives are to allow merchants and consumers to have secure mobile banking as well and have the ability to issue charges as well as customize user profiles. Another business objective will be for merchants and consumers to search transaction history. Our desire for PAyTEL is to lead the mobile banking app industry to a new frontier, with clean, efficient, user interface, along with in-depth user profile customization. The actual performance of PAyTEL will be determined by the volume of transaction requests created by merchants and consumers that find the application and utilize its full potential.

1.3. Security and ethical concerns

Our focus for PAyTEL will be fast but secure transactions for merchants and consumers. It will have encrypted data along the entire transaction process. Ethically, we will not allow the average user to access any transaction apart from their own as well as limiting displayed.

1.4. Glossary of Key Terms

1.4.1. Transaction: A request or offer of payment.
1.4.2. User: A person who is using the PAyTEL app.
1.4.3. Facial recognition: An algorithm that detects a face or faces in a photo or video.
1.4.4. AWS: Amazon Web Services.

2. Project Execution and Planning

2.1. Team information

2.1.1. Alexander Dapoz

Front-end development and documentation.

2.1.2. Anthony Calandra

Application development and documentation.

2.1.3. Axel Van Hoyweghen

Back-end (server, database, etc.) design and implementation, application development and documentation.

2.1.4. Lauren Lingeman

Application development and managing documentation.

2.1.5. Marsol Damon

Application development and documentation.

2.1.6. Noah McGivern

Front-end development and documentation.

2.2. Tools and Technology

2.2.1. Android Studio

Android Studio is Android's official IDE. It is purpose-built for Android to accelerate your development and help you build the highest-quality apps for every Android device.

2.2.2. Java

Java is an object-oriented programming language

2.2.3. React

A JavaScript library for building user interfaces for web applications.

2.2.3.1. HTML and CSS

HyperText Markup language (HTML) and Cascading Style Sheets are the basic languages used to create a website or web application.

2.2.3.2. Javascript

JavaScript is a high-level, interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web.

2.2.4. Node.JS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

2.2.5. Amazon Web Services (AWS)

2.2.5.1. API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

2.2.5.2. Cognito

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0.

2.2.5.3. DynamoDB

Amazon DynamoDB is a nonrelational database that delivers reliable performance at any scale.

2.2.5.4. Lambda

AWS Lambda lets you run code without provisioning or managing servers and scales as your application grows.

2.2.5.5. Mobile Hub

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

2.2.5.6. Rekognition

Rekognition allows you to automatically identify objects, people, text, scenes, and activities, as well as detect any inappropriate content.

2.2.5.7. S3

Object storage built to store and retrieve any amount of data from anywhere.

2.3. Project plan

2.3.1. Sprint 1 (9/6 - 9/13)

2.3.1.1. Define requirements
2.3.1.2. Layout basic sprint schedule
2.3.1.3. Setup collaboration tools (GitHub, Slack, Google Drive)
2.3.1.4. Select development environment, languages, tools, etc. and get all team members access (Android Studio, AWS DynamoDB, Bootstrap, Java, Android SDK 8, Bootstrap, Groovy, Kotlin, Java JDK, CSS, HTML, Javascript)
2.3.1.5. Begin project documentation

2.3.2. Sprint 2 (9/13 - 9/27)

2.3.2.1. Setup AWS (Cognito, IAM, Mobile Hub) for user account creation and authentication
2.3.2.2. Design consistent UI/UX for application
2.3.2.3. Setup QA for testing for user authentication system
2.3.2.4. Research and test phone facial recognition system for user authentication
2.3.2.5. Design database for storing transactions and user information
2.3.2.6. Write user stories for Sprint 3

2.3.3. Sprint 3 (9/27 - 10/13)

2.3.3.1. Implement UI/UX for application (login, registration, user account page, main landing page)
2.3.3.2. Connect application to login server and database storing user information
2.3.3.3. Design UI/UX for transaction portal
2.3.3.4. Setup transaction portal login at backend for admin
2.3.3.5. Setup QA for testing transaction portal login
2.3.3.6. Write user stories for Sprint 4

2.3.4. Sprint 4 (10/13 - 10/25)

2.3.4.1. Implement UI/UX for application (settings, anything else unfinished)
2.3.4.2. Implement UI/UX for transaction portal (login, main landing page, database queries)
2.3.4.3. Finalize database design
2.3.4.4. Setup QA for testing database queries and storage
2.3.4.5. Write user stories for Sprint 5

2.3.5. Sprint 5 (10/25 - 11/8)

2.3.5.1. Implement UI/UX for transaction portal (anything unfinished)
2.3.5.2. Debug and testing for whole application
2.3.5.3. Feature lock by end of this sprint

2.3.6. Sprint 6 (11/8 - 11/22)

2.3.6.1. Continue debug and testing

2.3.6.2. Finish final documentation
2.3.6.3. Prepare final presentation and poster board

2.4. Best standards and Practices

2.4.1. We followed a basic agile software development process with 2 week sprints and weekly in-person meetings. We collaborated online using Slack and GitHub. Frequent testing of the application occurred throughout the duration of the project.

3. System Requirement analysis

3.1. Functional requirements

3.1.1. User must be able to create a user account based on their Google login
3.1.2. User must be able to offer payment to another user (buyer role)
3.1.3. User must be able to accept payment from another user (seller role)
3.1.4. User must pass face scan to access their personal account
3.1.5. User must be able to see their transaction history
3.1.6. User must be able to change their payment method through the settings page
3.1.7. User must be able to generate a QR code
3.1.8. User must be able to scan a QR code
3.1.9. Admin must be able to access all transactions
3.1.10. Admin must be able to report malicious transactions to users

3.2. Non-functional requirements

3.2.1. User must have a valid Google login consisting of username and password
3.2.2. User must have a valid credit card
3.2.3. User must have an phone running Android API 26 or higher
3.2.4. Database must store user account information securely
3.2.5. Database must store all transactions
3.2.6. Server must be able to handle all incoming transactions real-time
3.2.7. Website and mobile app must both use the same color palette throughout.
3.2.8. Data must be encrypted when transferred between web-app / mobile-app and the server
3.2.9. Web-app interface must be responsive for variable screen sizes
3.2.10. Server must remove pending transactions that have timed out

3.3. On-screen Appearance of landing and other pages requirements

3.3.1. Initial Sign-In screen
- 3.3.1.1. A user must not be sign-in
- 3.3.1.2. Redirected from:
  - 3.3.1.2.1. Initial app startup
  - 3.3.1.2.2. Logout button on Settings page

3.3.2. Change Profile info.
- 3.3.2.1. A user must be signed-in
- 3.3.2.2. Redirected from:
  - 3.3.2.2.1. Sign-In screen
  - 3.3.2.2.2. Change Profile button on Setting page

3.3.3. Change Payment info.
- 3.3.3.1. A user must be signed-in
- 3.3.3.2. Redirected from:
  - 3.3.3.2.1. Change Profile screen
  - 3.3.3.2.2. Change Payment button on Setting page

3.3.4. Change Billing Address
- 3.3.4.1. A user must be signed-in
- 3.3.4.2. Redirected from:
  - 3.3.4.2.1. Change Payment screen
  - 3.3.4.2.2. Change Billing Address button on Setting page

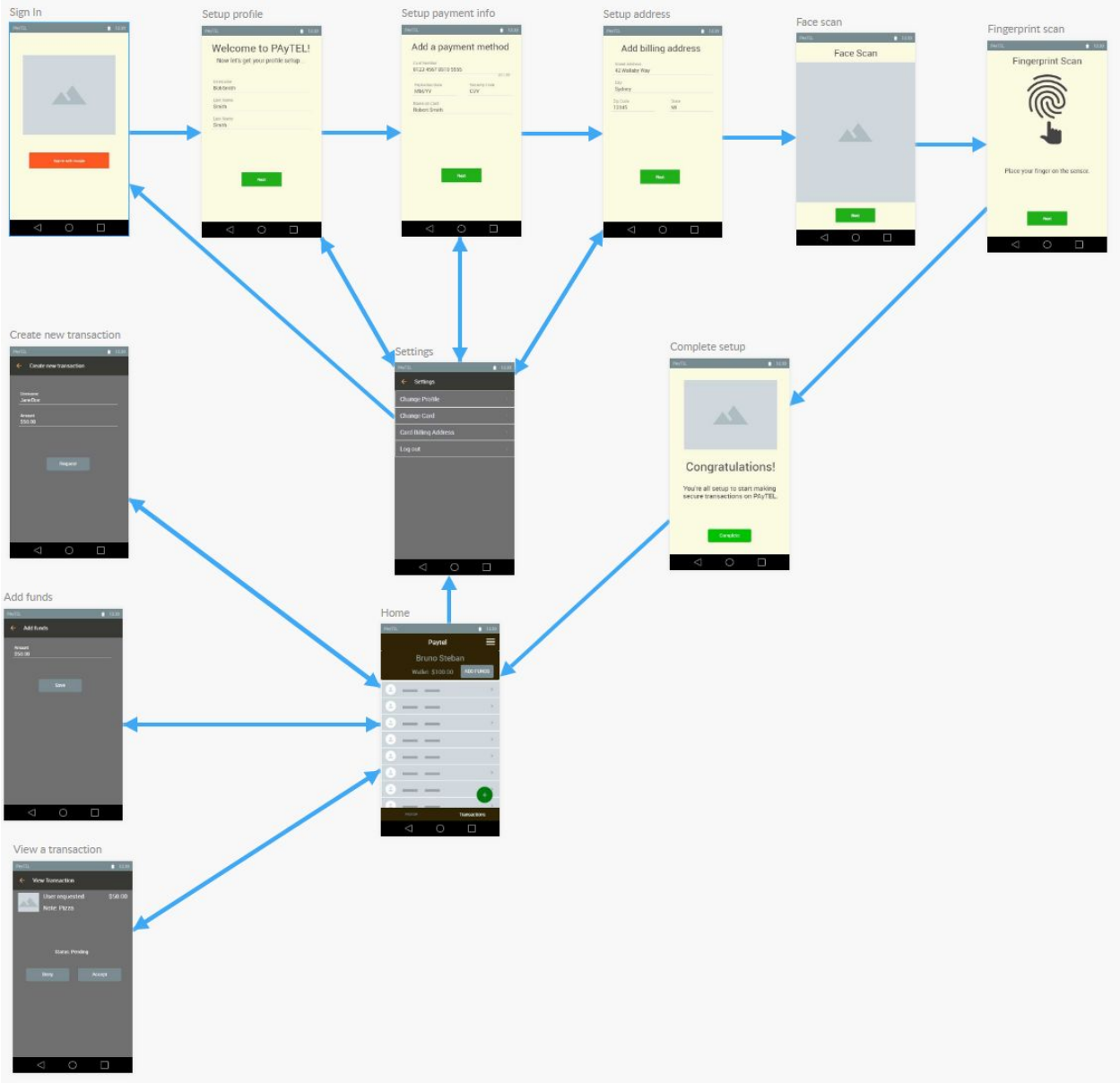
3.3.5. Face Scan
- 3.3.5.1. A user must be signed-in
- 3.3.5.2. Redirected from:
  - 3.3.5.2.1. Change Billing Address screen
  - 3.3.5.2.2. Approve button on View Transaction page

3.3.6. Fingerprint Scan
- 3.3.6.1. A user must be signed-in
- 3.3.6.2. The user’s phone must support the fingerprint sensor
- 3.3.6.3. Redirected from:
  - 3.3.6.3.1. Face Scan screen
  - 3.3.6.3.2. Approve button on View Transaction page

3.3.7. Home Screen
- 3.3.7.1. A user must be signed-in
- 3.3.7.2. Redirected from:
  - 3.3.7.2.1. Setup completed
  - 3.3.7.2.2. Settings page
  - 3.3.7.2.3. Add Funds page
  - 3.3.7.2.4. View Transaction page
  - 3.3.7.2.5. Create New Transaction page

- 3.3.8. Create Transaction
  - 3.3.8.1. A user must be signed-in
  - 3.3.8.2. Redirected from:
    - 3.3.8.2.1. Home page
- 3.3.9. View Transaction
  - 3.3.9.1. A user must be signed-in
  - 3.3.9.2. Redirected from:
    - 3.3.9.2.1. Home page
- 3.3.10. Add Funds
  - 3.3.10.1. A user must be signed-in
  - 3.3.10.2. Redirected from:
    - 3.3.10.2.1. Home page
- 3.3.11. Settings
  - 3.3.11.1. A user must be signed-in
  - 3.3.11.2. Redirected from:
    - 3.3.11.2.1. Home page

3.4. Wireframe designs



4. Functional Requirements Specification

4.1. Stakeholders

4.1.1. Project team members
4.1.2. Businesses which support our service as a form of payment
4.1.3. PAyTEL users
4.1.4. Local government agencies looking to prevent fraud and cybercrime

4.2. Actors and Goals

4.2.1. Actors

4.2.1.1. PayTel Application User
4.2.1.2. PayTel Administrator
4.2.1.3. Database

4.2.2. Goals

4.2.2.1. Reduce credit card fraud by creating system for secure transactions utilizing facial recognition.
4.2.2.2. Create user-friendly application to allow secure transactions for easy exchange of money.

4.3. User Stories, scenarios and use cases

4.3.1. User Stories

4.3.1.1. As a user, I want to be able to create an account on PayTEL that is tied to my personal google account.
4.3.1.2. As a user, I want to be able to login to my PayTEL account using my personal google account.
4.3.1.3. As a user, I want to make sure my account is secure by using facial recognition in addition to my login information.
4.3.1.4. As a user, I want to be able to add information to my PayTEL profile including credit card details and name.
4.3.1.5. As an administrator, I want to be able to access the PayTEL web portal to view transactions and user information.
4.3.1.6. As a user, I want to be able to login to my PAyTEL account on the PAyTEL android app.
4.3.1.7. As a user, I must be able to create a transaction through my PAyTEL account.
4.3.1.8. As a user, I must be able to view my transaction history on my account.
4.3.1.9. As a user, I should be able to update my profile information after account creation.
4.3.1.10. As a user, I want to make sure any initiated transaction is secure through facial recognition.
4.3.1.11. As an admin, I must be able to manage user accounts and transactions from the database, including being able to flag particular accounts or transactions.
4.3.1.12. As a user, I want to be sure my data is secure in the database.

4.3.2. Scenarios

4.3.2.1. Bob is out with Joe and wants to buy a popsicle from the popsicle stand. The popsicle stand only accepts cash, which Bob has none of. He does, however, have his phone. Joe offers to pay for Bob's popsicle if he pays him back. Bob agrees, but instead of having to go all the way to the bank, he wants to be able to be able to simply pull out his phone and transfer money to Joe directly.

4.3.2.2. Lucy is terrified of credit card fraud and will only use the most secure transaction applications. She wants to secure it with facial recognition because she never takes or allows anyone to take pictures of her so it would be a very secure method for her.

4.3.2.3. Joey plays betting games with his friends all the time and constantly loses. He would like an easy, quick way to make good on the bets. He doesn't want to have to enter credit card information all the time or carry cash. He just wants to be able to enter their usernames to send them the cash.

4.3.3. Use Cases

4.3.3.1. Use case: Create user account
Entry condition: On main app login page
Trigger: Enters google login information
Exit Condition: Clicks submit
Flow of Events:

a) User opens app
b) User enters valid google login
c) User clicks submit
d) Database sees user does not exist in database
e) User enters all required profile information
f) User takes required picture
g) User clicks submit
    i) Successful submit: user taken to landing page
    ii) Unsuccessful submit: user prompted to check entered info

4.3.3.2. Use case: Login to user account
Entry condition: On main app login page
Trigger: Enters google login information

Exit Condition: Clicks submit
Flow of Events:

a. User opens app
b. User enters valid google login
c. User clicks submit
d. Database sees user exists in database
e. User taken to landing page

4.3.3.3. Use case: Send transaction to other user
Entry condition: User is logged in
Trigger: User selects make transaction
Exit condition: User submits transaction request
Flow of Events:

a. User enters transaction information, including valid username
b. User submits transaction request
c. User is taken back to home page with pending transaction showing

4.3.3.4. Use case: Receive transaction from other user
Entry condition: User is on home page
Trigger: User selects the pending transaction
Exit condition: User accepts or denies transaction
Flow of Events:

a. User clicks accept or deny
b. If accepted, user must take a picture and pass facial recognition
c. If facial recognition is passed, transaction is completed
d. If denied, transaction does not go through
e. In either case transaction status is updated on transaction page
f. User is taken back to home page

4.3.3.5. Use case: Change user settings
Entry condition: User is on home page
Trigger: User selects settings
Exit condition: User clicks submit
Flow of Events:

a. User makes any desired changes to profile settings
b. User clicks submit when satisfied
c. User is taken back to home page

4.3.3.6. Use case: Admin log in

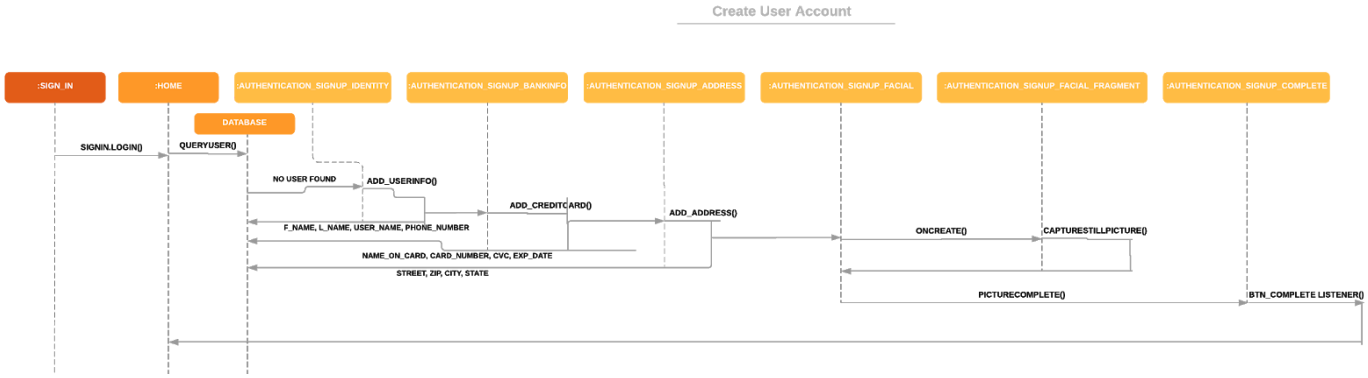
Entry condition: Admin is on web portal
Trigger: Enters admin login information
Exit condition: Admin clicks login
Flow of Events:

a. Admin enters login information when prompted at web portal
b. If successful, admin is taken to web portal home
c. If unsuccessful, admin is prompted to re-enter login information

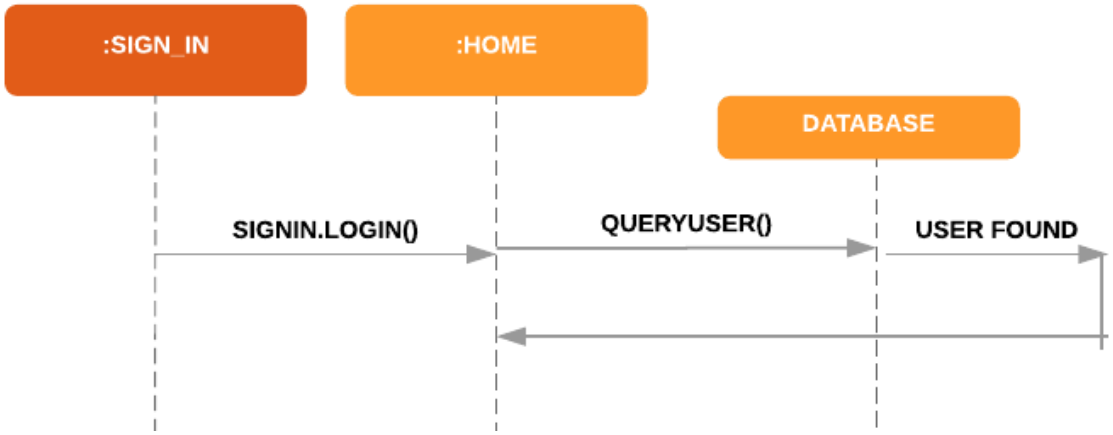
4.3.3.7. Use case: Flag transaction
Entry condition: Admin is logged into web portal
Trigger: Admin selects transaction to flag
Exit condition: Admin submits flag
Flow of Events:

a. Admin goes to transaction table in web portal
b. Admin selects one or many transactions
c. Admin submits transactions to flag
d. Selected transactions now appear as flagged

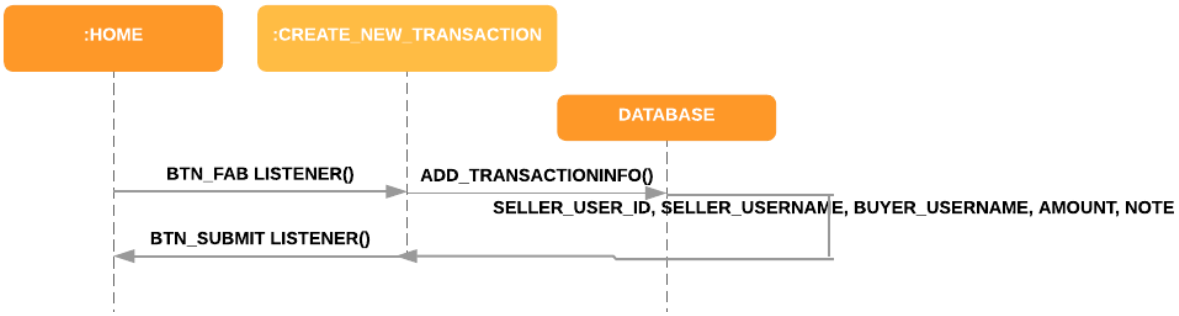
4.4. System Sequence / Activity Diagrams



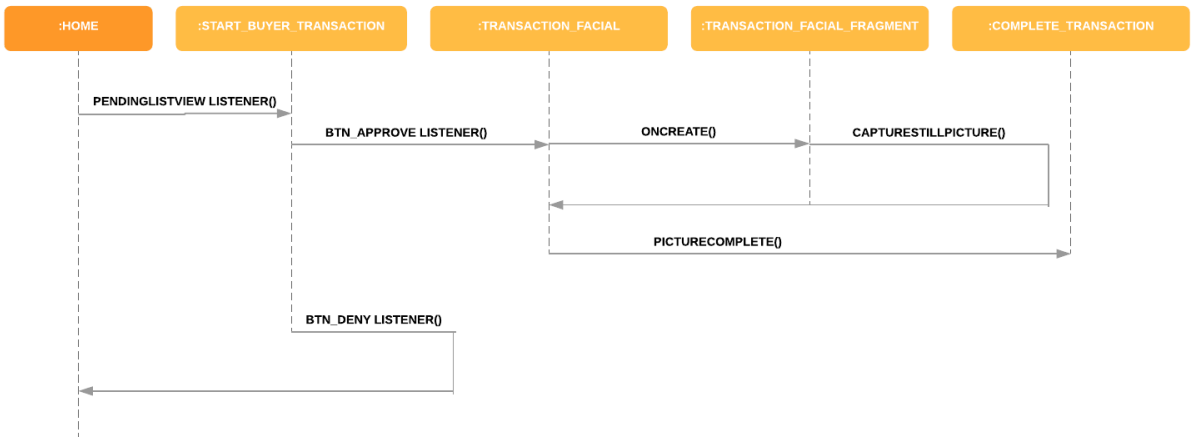
Login to User Account



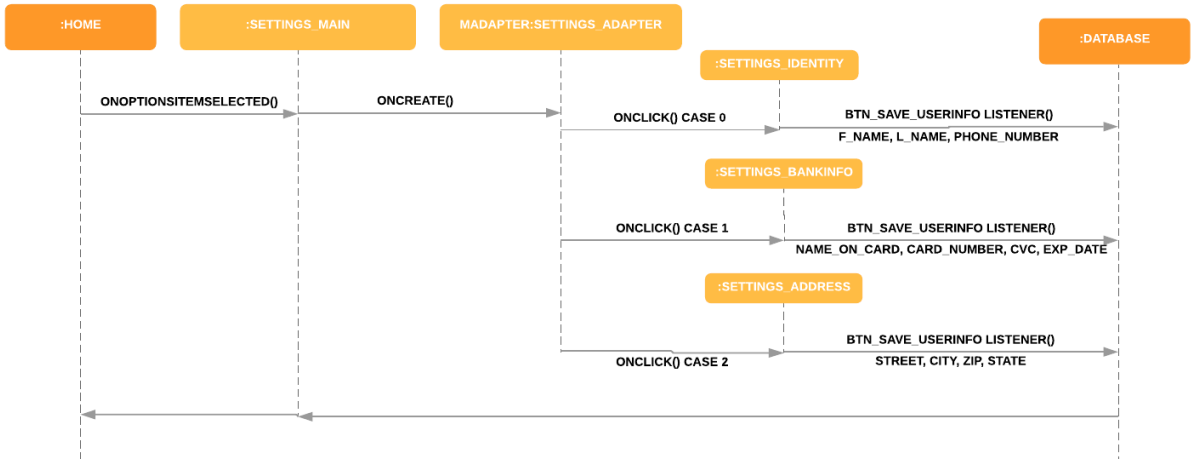
Send Transaction to Other User



Receive Transaction from User



Change User Settings



5. User Interface Specifications

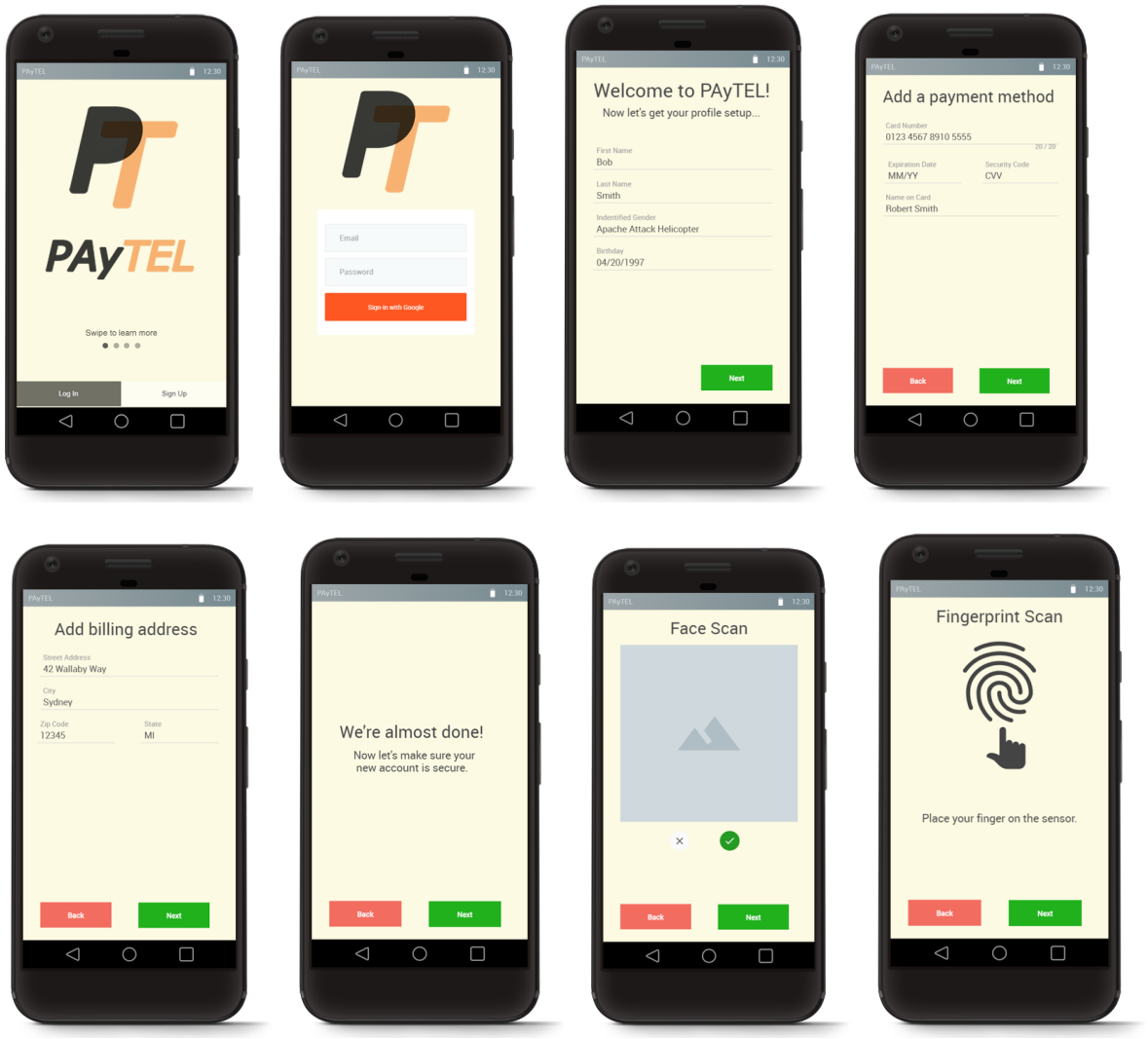
5.1. Preliminary Design

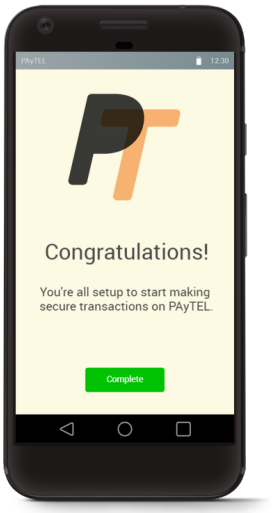
5.1.1. Tools used

5.1.1.1. Adobe Photoshop CC

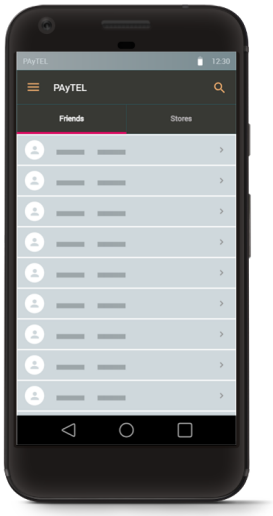
5.1.1.2. FluidUI

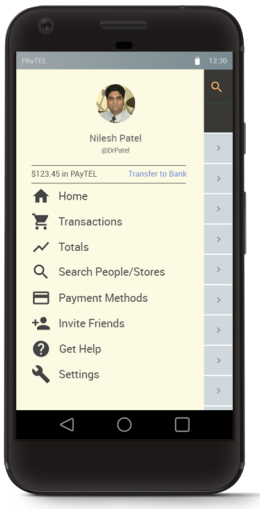
5.1.2. App mockup screen designs

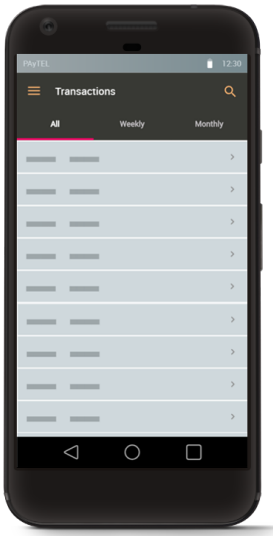


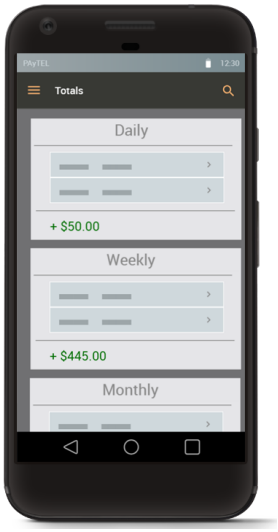

Congratulations!
You're all setup to start making secure transactions on PAyTEL.
Complete


PAyTEL


PAyTEL
Friends
Stores


Nilesh Patel
$123.45 in PAyTEL
Transfer to Bank
Home
Transactions
Totals
Search People/Stores
Payment Methods
Invite Friends
Get Help
Settings


Transactions
All
Weekly
Monthly

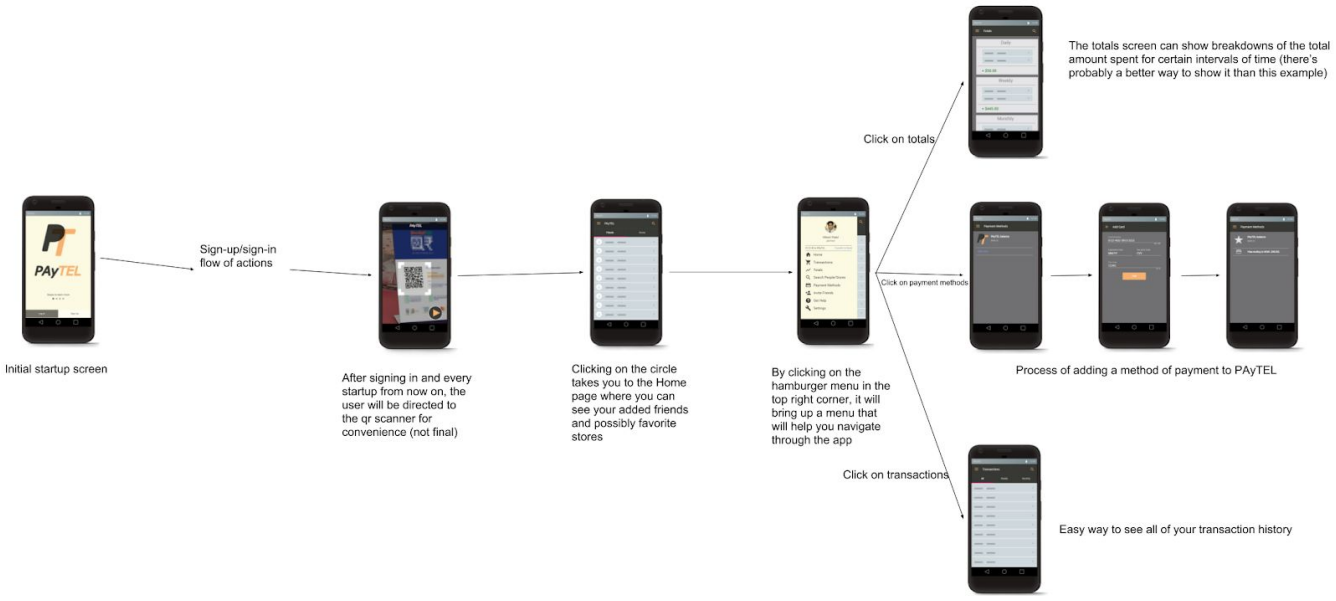

Totals
Daily
+ $50.00
Weekly
+ $445.00
Monthly


Payment Methods
PAyTEL balance
$429.33
Add card...


Add Card
Card Number
0123 4567 8910 5555
Expiration Date
MM/YY
Security Code
CVV
Zip Code
12345
Add


Payment Methods
PAyTEL balance
$429.33
Visa ending in 5555 (09/23)

5.1.3. App mockup flow



5.2. User Effort Estimation

5.2.1. Actors and Use Cases

Actor Name	Description	Complexity	Weight
PAyTEL User	Player interacts with system via GUI	Complex	3
PAyTEL Admin	User interacts with system via web portal	Complex	3
Database	Database is another system interacting through a protocol	Average	2

Use Case Name	Description	Complexity	Weight
Create user account (UC-1)	Simple UI. 7 steps required. 2 required actors (User, Database).	Average	10
Login to user account (UC-2)	Simple UI. 5 steps required. 2 required actors (Player, Database).	Average	10

Send transaction to other user (UC-3)	Simple UI. 3 steps required. 2 required actors (Player, Database).	Average	10
Receive transaction from other user (UC-4)	Simple UI. 6 steps required. 2 required actors (Player, Database).	Average	10
Change user settings (UC-5)	Moderate UI. 3 steps required. 2 required actors (Player, Database).	Average	10
Admin login (UC-6)	Simple UI. 3 steps required. 2 required actors (Admin, Database).	Average	10
Flag transaction (UC-7)	Simple UI. 4 steps required. 2 required actors (Admin, Database).	Simple	5

5.2.2. Calculate UUCP

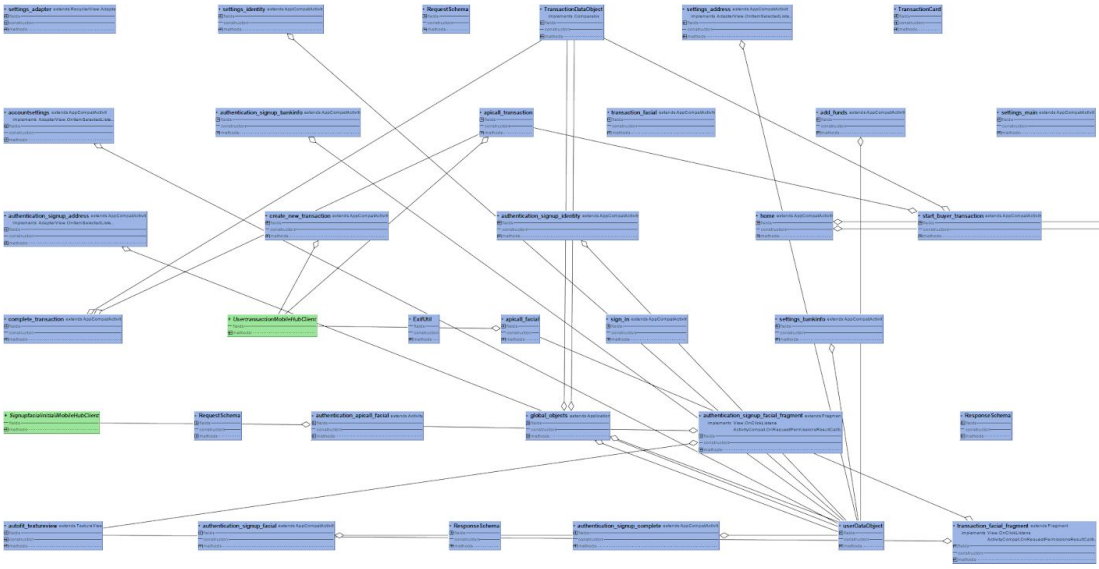
UAW = 2x3 + 1x2 = 6 + 2 = 8
UUCW = 1x5 + 6x10 + 0x15 = 5 + 60 = 65

UUCP = UAW + UUCW = 8 + 65 = **73**

6. Static Design

6.1. Class model

We also included a png image off the detailed uml diagram in the documentation folder submitted, because it was too large to add. Below you can see a class overview.



6.2. System Operation Contracts

Contract 1:	SignupUser
Operation:	authentication_signup_address(), authentication_signup_bankinfo, authentication_signup_facial, authentication_signup_facial_fragment, authentication_signup_identity, authentication_signup_complete
Cross References:	Use Case: Signup new user
Preconditions:	Open application and not already registered as a user
Postconditions:	A user was created with settings inputted as well as a facial picture stored in the database. Wallet initialized for $100.00

Contract 2:	FacialRecognitionRegister
Operation:	authentication_signup_facial_fragment(), authentication_signup_facial
Cross References:	Use case: Facial recognition for security
Preconditions:	User is signup up
Postconditions:	Facial picture registered and stored under that user in the database.

Contract 3:	create_new_transaction
Operation:	create_new_transaction(), apicall_transaction
Cross References:	Use Case: Create a transaction request
Preconditions:	User must be registered, and the user requested must be a registered user
Postconditions:	New transaction request created

Contract 4:	ModifySettings
Operation:	settings_adapter, settings_address, settings_bankinfo, settings_identity, settings_main
Cross References:	Use Case: Change user settings
Preconditions:	Must be a registered user
Postconditions:	New User changes are modified in the DB

Contract 5:	ApproveTransaction
Operation:	complete_transaction, transaction_facial, transaction_facial_fragment, apicall_transaction, apicall_facial, start_buyer_transaction
Cross References:	Use Case: Approve a transaction
Preconditions:	Must have a registered user, must be requested a transaction from another registered user. Must also be the correct, user for the facial recognition
Postconditions:	Transaction accepted, wallet impacted, and transaction info sent to database

Contract 6:	DenyTransaction
Operation:	complete_transaction, apicall_transaction, start_buyer_transaction
Cross References:	Use Case: Deny a transaction
Preconditions:	Must have a registered user, must be requested a transaction from another registered user.
Postconditions:	Transaction denied, depending choice, wallet impacted, and transaction info sent to database

Contract 7:	FacialCheck
Operation:	transaction_facial_fragment, transaction_facial, apicall_transaction
Cross References:	Use Case: Secure transactions
Preconditions:	User must be registered, have accepted a transaction, and must be the same user that signup
Postconditions:	Transaction verified

Contract 8:	FingerprintCheck
Operation:	transaction_facial_fragment, transaction_facial, apicall_transaction
Cross References:	Use Case: Secure transactions
Preconditions:	User must be registered, have accepted a transaction, and must be the same user that signup, and have fingerprint check.
Postconditions:	Transaction verified

Contract 9:	ViewTransaction
Operation:	start_buyer_transaction
Cross References:	Use Case: View transactions pending and completed
Preconditions:	User must have a transaction pending or completed
Postconditions:	User views transaction

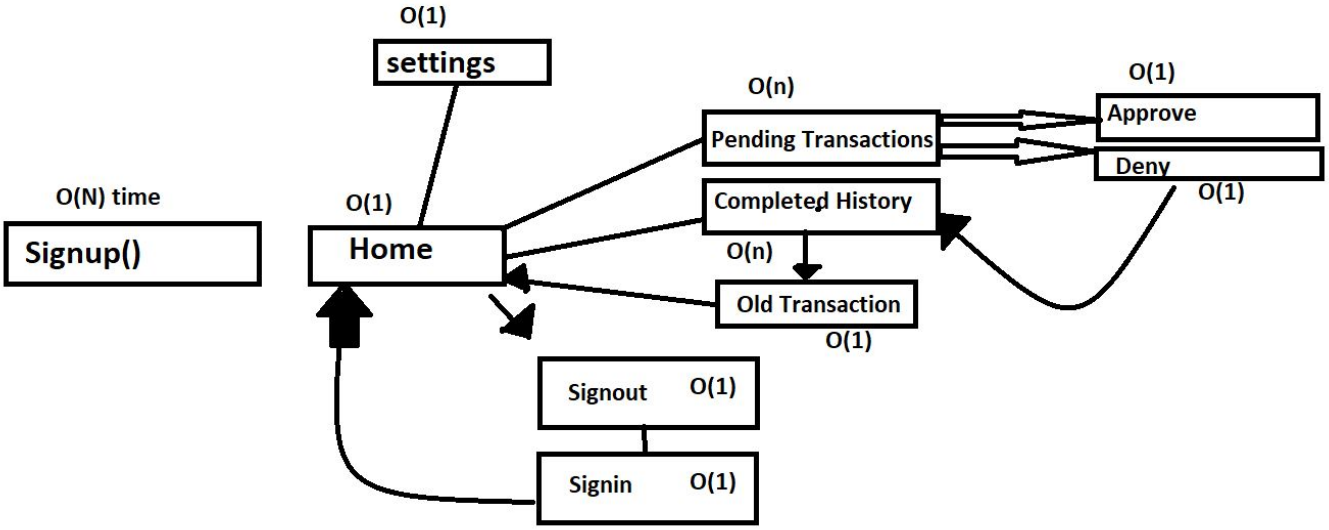
Contract 10:	Signout
Operation:	home
Cross References:	Use Case: Sign out of user account
Preconditions:	User must have a registered user to sign out
Postconditions:	User Signs out

Contract 11:	Signin
Operation:	home, sign_in
Cross References:	Use Case: Sign into registered account
Preconditions:	User must have a registered user to sign in
Postconditions:	User signs in

Contract 12:	AddWallet
Operation:	add_funds
Cross References:	Use Case: Add funds to user account
Preconditions:	Preconditions: User must be registered
Postconditions:	Postconditions: User adds money to wallet

6.3. Mathematical Model

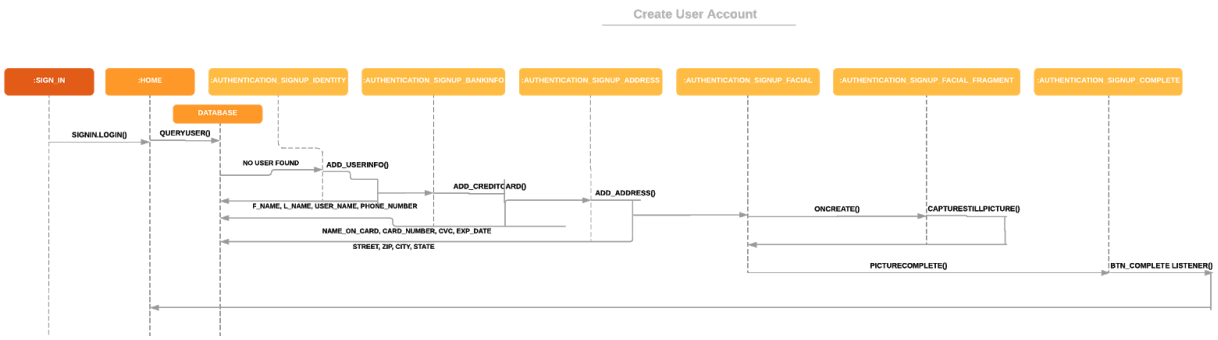
Mathematical Model



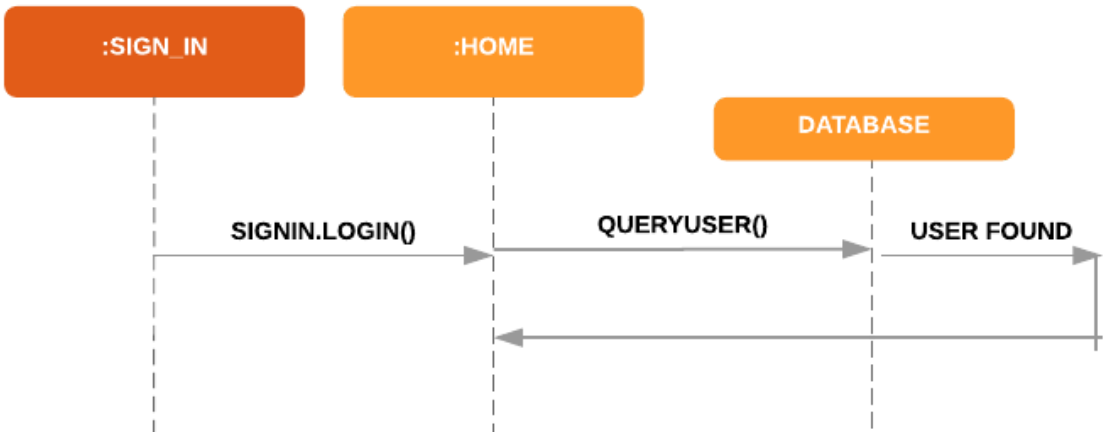
The Mathematical representation of our system is here. The application takes O(n) time to sign up based on the user's time inputting all the data for the signup process, including facial recognition. Once finished, accessing the home page takes 1 request based off O(1) time. This is the same for signing out, signing back in, and modified personal settings for the profile of the user. This is all based off O(1) processes because it takes one request to do any of these actions. The Pending transactions is O(n) for each transaction added in the list, therefore, indicating O(n). This is the same with the completed transaction history tab, which will add a transaction based on the users transactions so this is O(n). The approve or deny request takes 1 request so it is O(1).

7. Dynamic Design

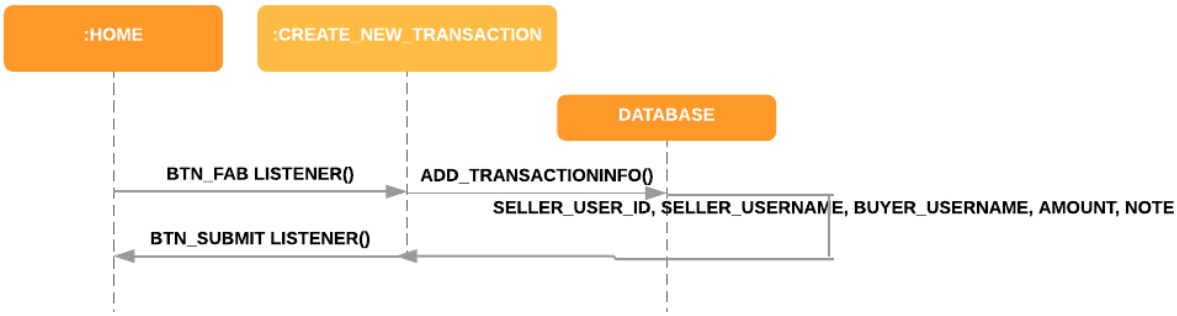
7.1. Sequence Diagrams



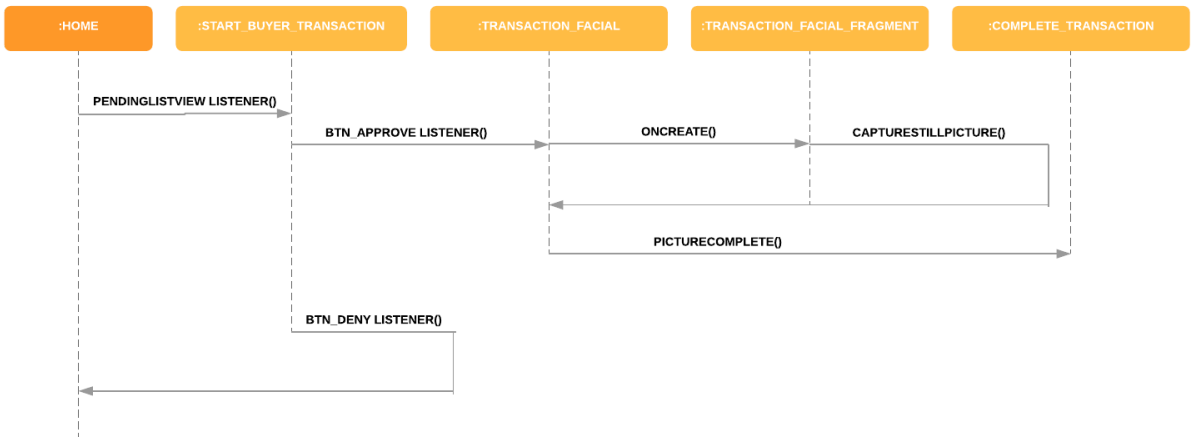
Login to User Account



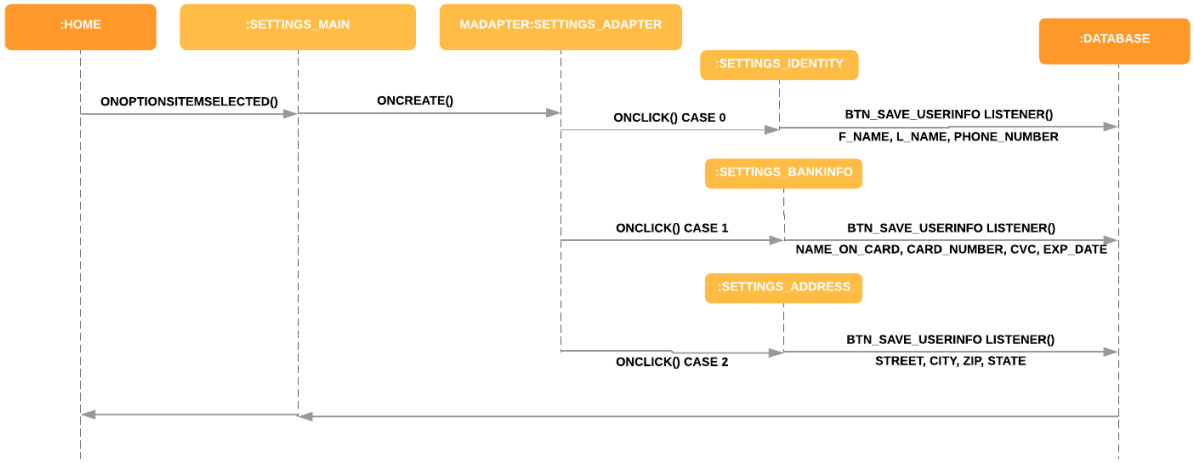
Send Transaction to Other User



Receive Transaction from User



Change User Settings



7.2. Interface Specifications

Function: Signup
Parameters: User Device ID, User Google login
Description: Allows user to register face and profile
Feedback: Allows User Creation
Errors: No device ID registered or no Google Login

Function: Sign in/out
Parameters: User Device ID, User Google login
Description: Allows user to sign in or sign out
Feedback: Allows User login ability
Errors: No registered users

Function: Request Transaction
Parameters: Requested User, User, DeviceID, wi-fi, amount, description
Description: Allows user to request money from another user with a note.
Feedback: Allows Fluid Transaction
Errors: No registered user or requested user in database

Function: Accept/Deny transaction
Parameters: Requested Transaction, User, DeviceID, Wi-fi, facial
Description: Allows user to accept or deny transaction
Feedback: Allows Responsive and secure facial recognition transactions
Errors: Incorrect facial recognition, no registered user in DB

Function: Change Profile Settings
Parameters: User's profile information, Device ID
Description: Allows user to modify user information
Feedback: Allows profile management
Errors: No currently registered user

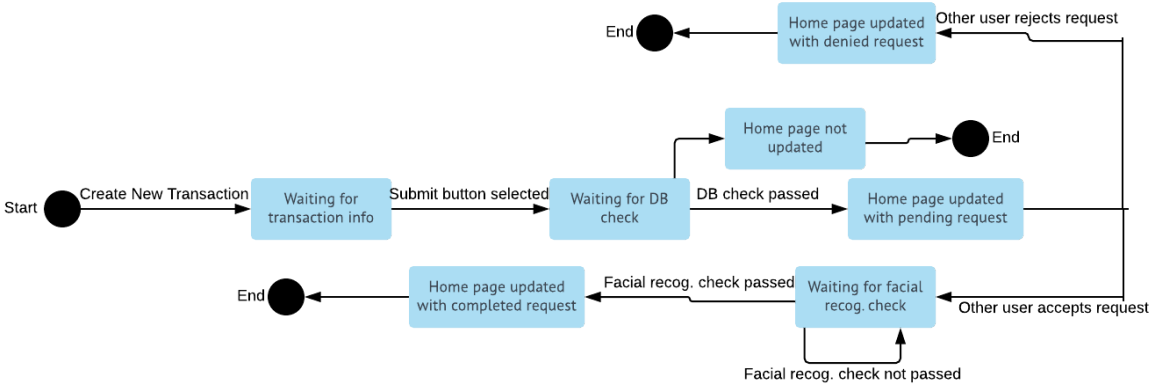
Function: View Old Transaction
Parameters: Transaction in history, User, DeviceID, wi-fi
Description: Allows user to view old completed transactions
Feedback: Allows transaction management and history
Errors: No transactions available or in database

Function: Facial Recognition
Parameters: Transaction(approve only) page, user, device, wi-fi
Description: Allows user to approve transactions with facial recognition
Feedback: Allows user to approve with an essential security feature
Errors: No face currently recognized or registered.

7.3. State Diagrams

7.3.1. New User

7.3.2. Transaction



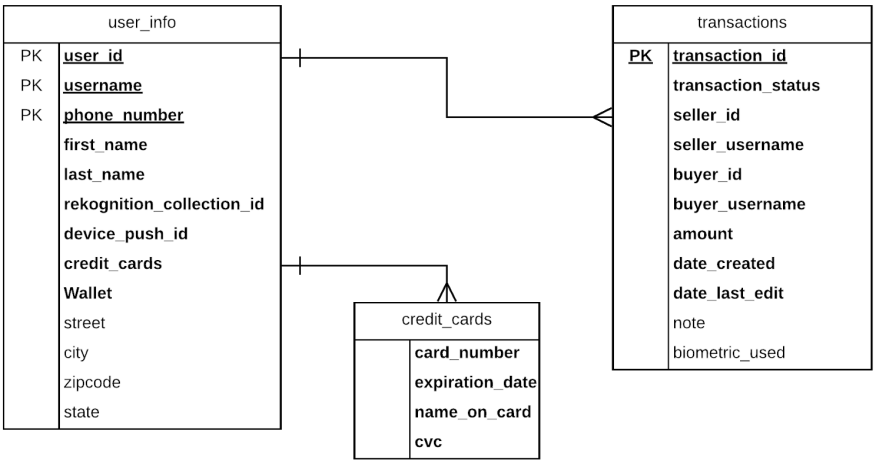
8. System Architecture and System Design

8.1. Subsystems / Component / Design pattern Identification

8.1.1. Database

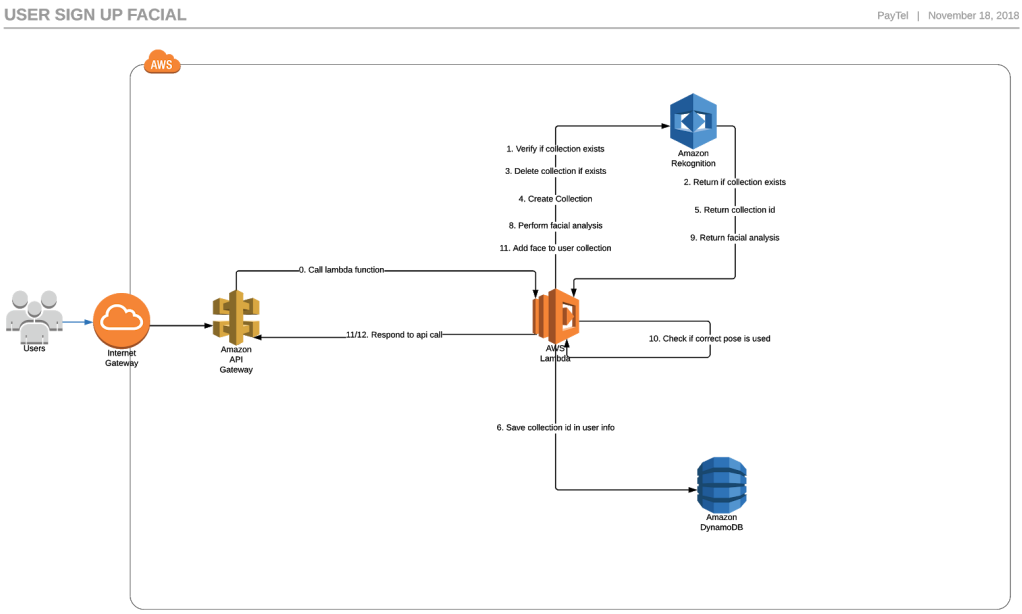
NoSQL database hosted on AWS DynamoDB. Utilizing a NoSQL database adds flexibility to the backend, so our data structure can evolve over time. Hosting the database on AWS DynamoDB will scale automatically when the number of users and transactions grow. The database utilizes 2 tables, user_info and transactions, which will store the

users information and transactions.

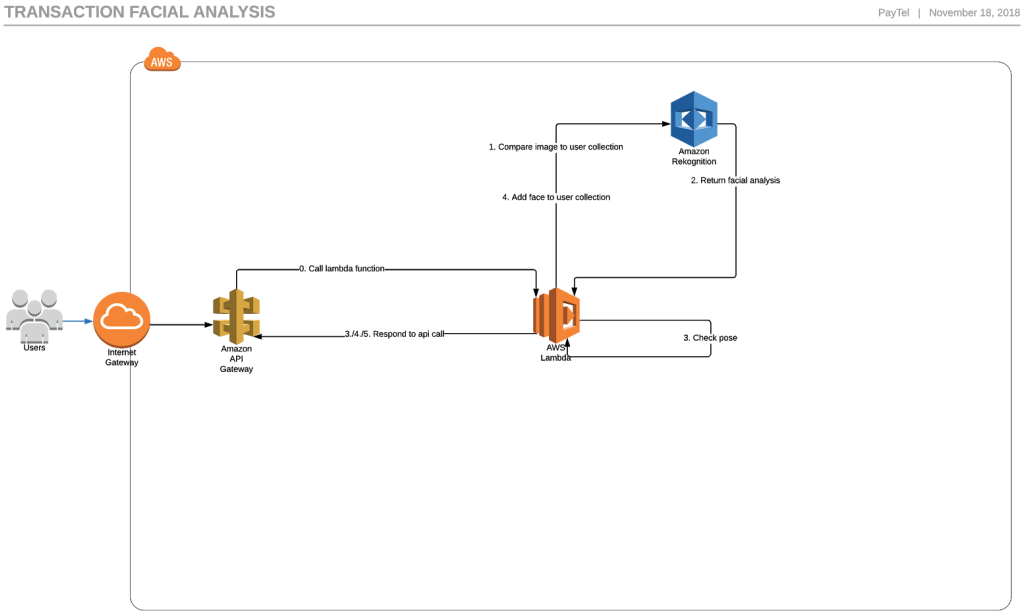


8.1.2. Backend

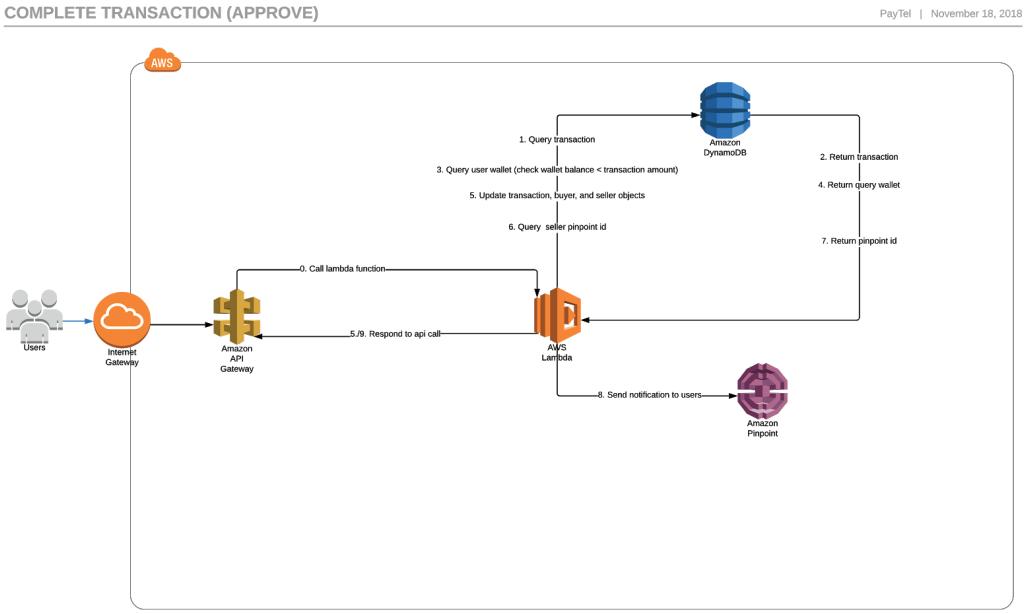
8.1.2.1. Facial verification on user sign up.

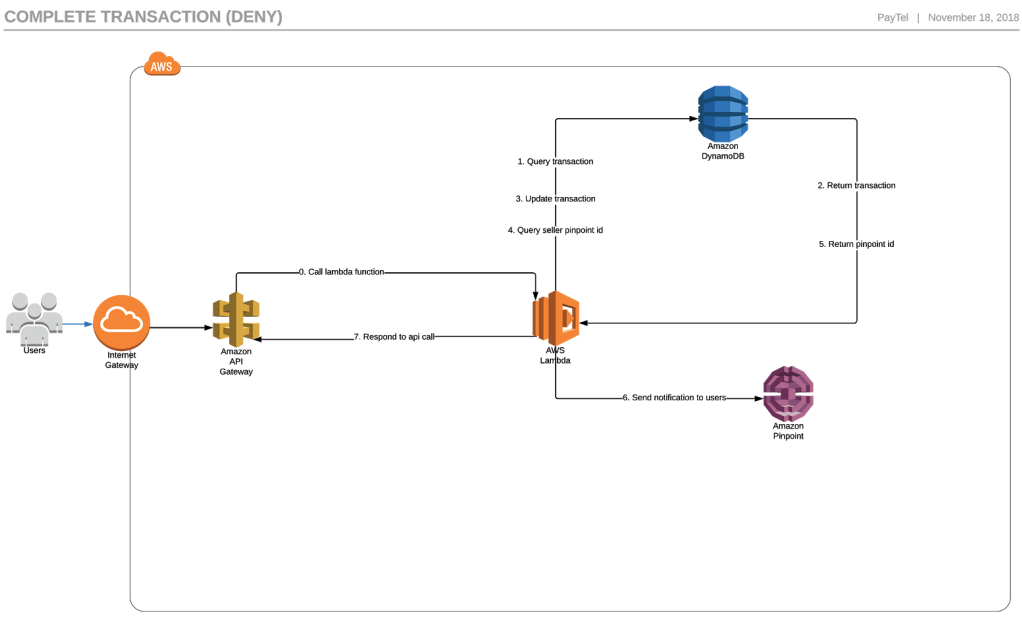


8.1.2.2. Facial verification when buyer approves transaction.



8.1.2.3. Complete transaction when buyer approves transaction and facial is verified.





8.2. Mapping Subsystems to Hardware (Deployment Diagram)

We do not have any deployment diagrams. The hardware (camera and fingerprint reader) we utilized in the application are abstracted by Android. We interfaced to the camera using the Camera API V2 and to the fingerprint reader using the Fingerprint API.

8.3. Persistent Data Storage

8.3.1. AWS S3

AWS S3 stores user images that are taken during transaction verification.

8.4. Network Protocol

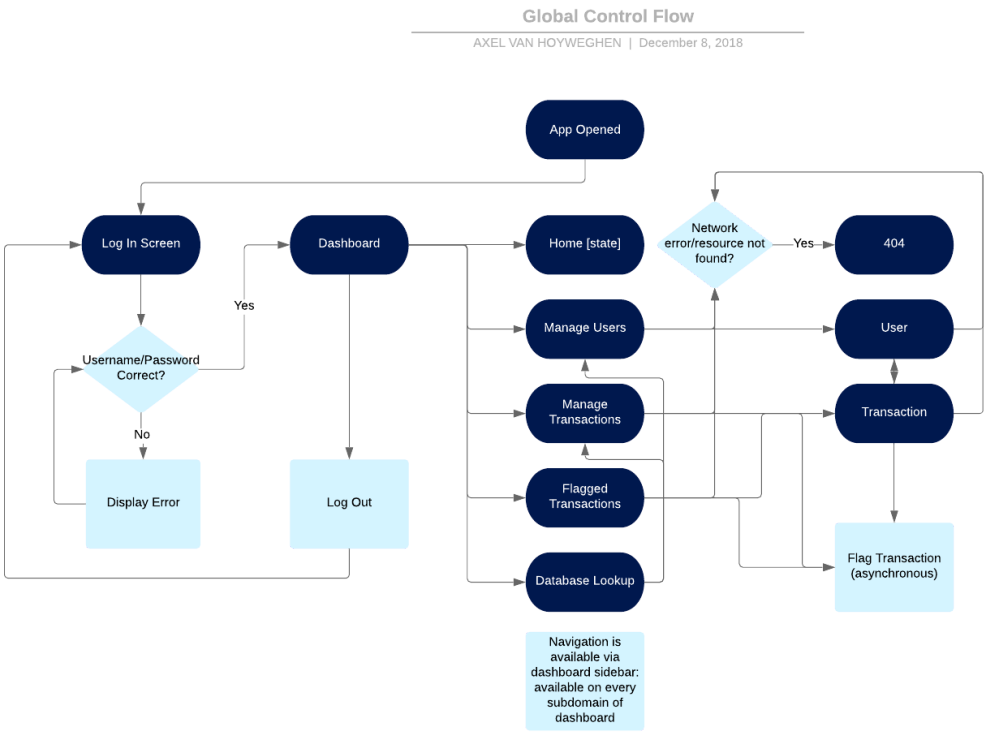
From Amazon: "The message broker supports the use of the MQTT protocol to publish and subscribe and the HTTPS protocol to publish. Both protocols are supported through IP version 4 and IP version 6. The message broker also supports MQTT over the WebSocket protocol."

By use of MQTT and HTTP through REST API (via Serverless API), PAyTEL is allowed to communicate with AWS servers securely and efficiently. AWS dashboard provides a manager for API calls; this allows for viewing of functions from a high-level observation, in-browser API code editing, organisation of API calls, a comprehensive API testing solution as well as many other tools to ensure the function calls you craft are secure, succinct and blazing fast.

In reference to §8.1 of PAyTEL's documentation, API calls are first handled by the Amazon API Gateway, ensuring authentication by cross-referencing AWS Cognito User Pool and IAM Permissions. The function call is then handled by AWS Lambda, which fires lambda functions to communicate with the appropriate services. A lambda function only uses up the resources it needs in order to execute the required code, in turn saving tremendous costs otherwise spent on running unused servers. How this works is that functions must first start ‘cold’, meaning that the API is not in active memory (consuming no resources concurrently), however, the API function will be kept alive for a predetermined amount of time after the first call from a cold start, meaning that services in constant use skip this startup process to become ‘warm’. Default or custom API return statements allow generating of feedback messages which may include a body of data, formatted in JSON for our Java and JavaScript based applications. Since these function calls may be made with either MQTT or HTTP protocols, it is easy to tailor a program to the individual need of heavy database querying through MQTT, or a quick image get via HTTP, all with the guarantee of security that AWS promises.

Ref: https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

8.5. Global Control Flow



8.6. Hardware Requirement

8.6.1. Android Application

8.6.1.1. Internet Access
8.6.1.2. Phone with Android OS (SDK 28 or higher)
8.6.1.3. Front-facing camera
8.6.1.4. Fingerprint Access

8.6.2. Admin panel

8.6.2.1. Internet Access

9. Algorithms and Data Structures

9.1. Algorithms

9.1.1. Autofitting the size of the TextureView for the camera fragment:

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    int width = MeasureSpec.getSize(widthMeasureSpec);
    int height = MeasureSpec.getSize(heightMeasureSpec);
    if (0 == mRatioWidth || 0 == mRatioHeight) {
        setMeasuredDimension(width, height);
    } else {
        if (width < height * mRatioWidth / mRatioHeight) {
            setMeasuredDimension(width, width * mRatioHeight / mRatioWidth);
        } else {
            setMeasuredDimension(height * mRatioWidth / mRatioHeight, height);
        }
    }
}
```

9.1.2. Round robin determination of which pose to ask for:

```
public String getPose() {
    Deque<String> myPoses = ((global_objects) getActivity().getApplication()).getPoseDeque();
    String[] poses = getResources().getStringArray(R.array.poses);
    if (myPoses.size() < 1) {
        for (int i = 0; i < poses.length; i++) {
            myPoses.add(poses[i]);
        }
    }
    String roundPose = myPoses.pop();
    myPoses.add(roundPose);

    System.out.println(myPoses);

    return roundPose;
}
```

9.2. Data Structures

9.2.1. Arrays for response objects, SparseIntArray to store orientations for camera setup, ArrayList to hold sizes for camera setup.

9.2.2. JSONObject and HashMap for ApiRequest when communicating with database.

9.2.3. Matrix to use for configuring the transform of the camera output.

9.2.4. Deque to cycle through poses.

10. User Interface Design and Implementation

10.1. User Interface Design Specification

10.1.1. Login

10.1.1.1. Logo, sign-in with Google button

10.1.2. Setup Profile

10.1.2.1. Simple, centered form - first name, last name, phone number

10.1.3. Add Payment Method

10.1.3.1. Simple, centered form - card number, expiration date, CVV, name on card

10.1.4. Add Billing Address

10.1.4.1. Simple, centered form - street address, city, state, zip code

10.1.5. Facial Recognition

10.1.5.1. Selfie camera preview

10.1.5.2. Action button to take a picture

10.1.6. Fingerprint verification

10.1.6.1. Prompt user to setup fingerprint verification

10.1.7. Finalize Signup

10.1.7.1. Logo, text telling user that sign-up is complete

10.1.8. Home Page

10.1.8.1. Header (logo, settings cogwheel -> go to settings page)

10.1.8.2. Wallet card (username, wallet amount, add funds button)

10.1.8.3. Scrollable list of transaction cards (icon indicating status [Pending, Completed, Cancelled, Flagged], username of sender, amount)
10.1.8.4. Bottom navigation bar to switch list between Pending and Completed transactions
10.1.9. Settings Page
10.1.9.1. Header (back button to Home screen, “Settings” title)
10.1.9.2. List of actions (Edit Profile, Change Payment Method, Change Billing Address, Sign Out)
10.1.10. Create Transaction
10.1.10.1. Header (back button to Home screen, “Make a payment” title)
10.1.10.2. Simple, centered form - username, amount, message
10.1.11. View Transaction
10.1.11.1. Header (back button to Home screen, “Make a payment” title)
10.1.11.2. Shows transaction icon, username, amount, message from requesting user, status
10.1.11.3. 2 Buttons to Accept/Deny transaction (if the current user is the buyer)
10.1.11.4. Verification picture from facial recognition (if the transaction is approved and if the current user is the buyer)
10.1.12. Add Funds
10.1.12.1. Current wallet amount
10.1.12.2. Simple, centered form - amount to be added to wallet

10.2. User Interface Implementation

10.2.1. User interface was implemented using Android Studio. Every view on the application has a separate layout xml file that corresponds to an activity. The correct xml is loaded upon calling of an intent to switch activities.

11. Testing

11.1. Unit Test Architecture and Strategy/Framework

11.2. Our strategy for testing was taking a test as you develop approach, much of our testing was done as we developed units and completed classes. This testing was done by hand testing the application by use of emulator or physical device. Tracking was done by use of the screen capture tool available with emulators.

11.3. We did also pair extensive manual unit testing with more automated tests. We utilized JUnit 4 and Espresso test suites to build unit and UI tests for our application.

11.4. Unit Test definition, test data selection

11.5. Be a application driven of user input data we took careful consideration in testing edge cases and “problematic” data as these are where our application could crash or even worse, present exploits.

11.6. So for data selection we tested data points directly outside and inside out data thresholds, such that if a dollar amount was limited to being less than or equal to

$10000 we would test at $9999.99 and $10000.01. Consideration was also given for “problematic data. Such as if we expected a dollar amount checking that any non dollar amount would be kicked back, such as $9999.999. Other consideration was taken for testing situations such as passing characters into a currency field.

11.7. System Test Specification

11.8. System testing was done with a black box approach. Ensuring that the system had correct outputs for any input, focusing on the test data defined above.

11.9. We also used our application mappings to find all application start and end points to verify all subsystems were tested thoroughly.

12. Project Management

12.1. Project Plan

12.2. Sprint 1 (9/6 - 9/13)

12.2.1. Define requirements
12.2.2. Layout basic sprint schedule
12.2.3. Setup collaboration tools (GitHub, Slack, Google Drive)
12.2.4. Select development environment, languages, tools, etc. and get all team members access (Android Studio, AWS DynamoDB, Bootstrap, Java, Android SDK 8, Bootstrap, Groovy, Kotlin, Java JDK, CSS, HTML, Javascript)
12.2.5. Begin project documentation

12.3. Sprint 2 (9/13 - 9/27)

12.3.1. Setup AWS (Cognito, IAM, Mobile Hub) for user account creation and authentication
12.3.2. Design consistent UI/UX for application
12.3.3. Setup QA for testing for user authentication system
12.3.4. Research and test phone facial recognition system for user authentication
12.3.5. Design database for storing transactions and user information
12.3.6. Write user stories for Sprint 3

12.4. Sprint 3 (9/27 - 10/13)

12.4.1. Implement UI/UX for application (login, registration, user account page, main landing page)
12.4.2. Connect application to login server and database storing user information
12.4.3. Design UI/UX for transaction portal

12.4.4. Setup transaction portal login at backend for admin
12.4.5. Setup QA for testing transaction portal login
12.4.6. Write user stories for Sprint 4

12.5. Sprint 4 (10/13 - 10/25)

12.5.1. Implement UI/UX for application (settings, anything else unfinished)
12.5.2. Implement UI/UX for transaction portal (login, main landing page, database queries)
12.5.3. Finalize database design
12.5.4. Setup QA for testing database queries and storage
12.5.5. Write user stories for Sprint 5

12.6. Sprint 5 (10/25 - 11/8)

12.6.1. Implement UI/UX for transaction portal (anything unfinished)
12.6.2. Debug and testing for whole application
12.6.3. Feature lock by end of this sprint

12.7. Sprint 6 (11/8 - 11/22)

12.7.1. Continue debug and testing
12.7.2. Finish final documentation
12.7.3. Prepare final presentation and poster board

12.8. Risk Management

12.8.1. Risk mitigation included weekly in-person group meetings to keep members on track for their individual tasks. Constant testing by all members occurred throughout the project so problems could be fixed as early as possible. Use of AWS helped to make sure user data was kept secure.

13. References

13.1. https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html
13.2. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
13.3. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
13.4. https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html
13.5. https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html
13.6. https://docs.aws.amazon.com/aws-mobile/latest/developerguide/reference-api-refs.html

13.7. https://developer.android.com/reference/android/hardware/camera2/package-summary