GMIT Cross-Platform Mobile Banking Application

Alan Niemiec

Shane Gleeson

Dara Starr

B.Sc.(Hons) in Software Development

April 17, 2017

Final Year Project

Advised by: Kevin O'Brien

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract GMIT (Galway-Mayo Institute of Technology) cross-platform mobile banking application is an app created using the Ionic framework, utilising technologies like Javascript, HTML5 and AngularJS. This is a mock banking app, designed for GMIT students where they can view their recent transactions, transfers, and manage the money in their account. This application focuses on security and transfer of data and has the AuthO framework implemented as the authentication standard. The use of open source API's such as Google Maps and their implementation in this app should also be noted.

The application is split into four main sections:

- Front End - The Ionic application that is downloaded to the user device. Utilising HTML pages as the GUI and AngularJS as the controller of the object. This is done in a Model-View-Controller fashion by linking to the database model of the application through HTTPS requests made to the middleware layer.
- Middleware - A REST API created in NodeJS and hosted on Heroku, it takes in HTTPS POST requests made by the user and calls a query to the MongoDB returning data objects.
- Database - The MongoDB database is hosted on the Mlab's servers, it provides us with the necessary data structures through a internet connection.
- Authorization - Auth0 provides a authentication widget for our application as well as a administrator option in it's online user control dashboard.

GitHub is used to manage all our changes to the project and from here Heroku takes the updated versions of the application and automatically builds and hosts new versions.

Our aim for this project was to produce an application that has the necessary needs for a student when it comes to banking. As students when it comes to finances there's only few things we do, and that is either saving, spending or transferring money so we aimed this application at the students out there just like ourselves. An application like ours would be of great benefit to students considering a lot of them will be living independently for the first time and a clear and concise banking app could help with their finances.

Authors

- **Alan Niemiec** - 4th Year Software Development (HONS), Galway-Mayo Institute of Technology
- **Shane Gleeson** - 4th Year Software Development (HONS), Galway-Mayo Institute of Technology
- **Dara Starr** - 4th Year Software Development (HONS), Galway-Mayo Institute of Technology

Chapter 1

Introduction

What is mobile banking? Mobile banking is a service provided by a bank or other financial institution that allows its customers to conduct financial transactions remotely using a mobile device such as a mobile phone or tablet[1]. It utilises software, usually called an app, provided by the financial institution to further decrease the gap between the sectors of Technology and Finance.[1]. Mobile banking provides users with an easier way of managing their financial affairs. It's faster, you can use it on the go and you don't have to wait in long lines in banks or travel to your nearest ATM.

The current technology sector is closely co-operating or in some cases competing with the finance sector creating a whole new industry breed of Fintech. This versatile environment blurs the lines between the two sectors and comes together to challenge the old way of handling finances by bringing the banking technology to the mobile world.[2]

Our main objective of this project was to create a cross platform application that we could use on Internet browsers, Android and IOS phones while focusing on the security learning aspects. We came up with the idea of a mock banking application for GMIT where students could have access to their account and manage their payments by transferring money and also looking at their recent transactions.

A banking application tackles a lot of the main aspects in programming in the form of user confidentiality, data transfer and mobile access. So we decided to design and implement an application covering most of these topics to some degree, this has created a unique learning environment and a steep research curve suitable for a final year project.

Our Banking Application is a cross platform app which means it can be exported to both Android and IOS phones, this is achieved by using the Ionic Framework. We decided to use Ionic v1 as we wanted to work with HTML5, Javascript and Angular. We have utilised the MongoDB database technology

so we can store and retrieve data and also protect the user's information.

Our idea at first was to aim our application at credit union customers but we then decided it would be a better idea to aim something at the students at GMIT so we set about creating a mobile banking application for students who attend GMIT. From here students would be able to manage their finances on the go and have their own savings account. They can even transfer money to other students across the college in the click of a button. With a simple email and password authentication process students can access their finances.

Mobile Banking is very appealing to the audience we chose to market this application to as they can monitor deposits and review their transaction history. They can also transfer money from their current account to their savings account. Because of this, students will also find it easier to save money and be able to access these savings whenever they need to. Students don't have a lot of money so being conscious of the way they handle their money is very important so incorporating a savings feature is very beneficial especially when students have to worry about paying bills, rent, college fees etc.

Chapter 2

Context

2.1 Chapters Explained

2.1.1 Methodologies

In this chapter we discuss Agile and Scrum methodologies and how we used these to help us in the foundation and progress of our project.

2.1.2 Technology Review

In this chapter we discuss each piece of technology we used in our application and how each of these technologies works. We give an in-depth, researched description on all frameworks and languages.

2.1.3 System Design

In this chapter we go about showing you how we implemented the technologies mentioned in the chapter above giving screen-shots and code from our application and showing you how the application functions. The architectural diagram is also included here.

2.1.4 System Evaluation

In this chapter we explain why we used these technologies, we discuss the limitations of the application and we talk about the testing of the app. We also discuss what also we may have done differently throughout our time together doing this project.

2.1.5 Conclusion

In our Conclusion we talk about what we may have done differently, what we learnt from our time doing this application and did we meet our objectives that we set out to meet at the start.

2.2 Application Context

This project will need to address a multitude of distinct issues in many different areas. We will develop it based on the context of a imaginary bank or credit union that is looking to move into the mobile banking field. We have identified this niche based on the credit unions available in our area and their lack of services outside of the physical branch. The organisation of the application will be separated into four main areas: front end (GUI) user interface, connectivity, database and security encompassing all of these layers in the best way possible. Our current research also involves analysing currently available applications from banks such as AIB and Bank of Ireland. These are designed in a extremely intricate way, creating applications that are not only secure but also extremely hard to reverse engineer, most of these application do not even allow for screen recording to happen during the use of the app. As a application that deals with highly sensitive user data it will have to cover many security aspects which will not all be possible to implement at our level of knowledge.

2.3 Github URL

- URL for the main application:

`https://github.com/sinderpl/BankingApplication`

- URL for the API part of the application

`https://github.com/sinderpl/BankingApplication/tree/herokuAPI`

2.4 Objectives

Since a application of this size will require a considerate amount of planning, we have decided to create a set of context objectives which we will work on applying in the application. Attached to these will also be rough ”User stories” as we will develop the application with a Agile approach.[3] Each

of these user stories will have Acceptance criteria attached so that we can review the goals achieved by the end of the project.

2.4.1 Scope

- **App portability** - The application has to be available on a wide range of devices: browser, IOS, Android as well as being suitably designed for these devices. It should be available remotely from a mobile device through a internet connection.

    1. As a banking application user I want to be able to access the application on any device I choose so that I am not constrained to a single device.
        - The application view scales well for a web browser.
        - The application view scales well for a Android devices.
        - The application view scales well for a IOS devices.
    2. As a banking application user I want to be able to access my data from anywhere so that I can use it on the go.
        - The application can be used on a mobile device.
        - The application can be used with an internet connection.

- **App security** - The application has to be secure and implement as much security as we are able to implement. This metric is the main objective of our research and the banking theme.

    1. As a bank administrator I want to know that my user information is secure on my database so that I can provide services to many customers without compromising their personal information.
        - The application database is secure from unwanted users.
        - The application is not accessible to third parties.
    2. As a bank administrator I want to know that the user data is safe in transfer so that they can access it from anywhere.
        - User information gets to the user device without the possibility of a man in the middle attack.
        - Sensitive data is not saved to the mobile device.
    3. As a bank administrator I want to be able to add and delete users on my end so that no one can sign up without verification.
        - The bank administrator has a way of creating users.

  - The bank administrator has a way of deleting users.
  - The bank administrator has a way of tracking user information, location,login device etc.

4. As a bank administrator I want to be able to manipulate the database so that I can correct any errors that arise in transactions.
  - The bank administrator has access to the database.
  - The bank administrator can revise account information.
  - The bank administrator can modify the database information.
5. As a banking application user I want to be able to log in to my personalised account.
  - The user can log in to his account through authentication procedures.
  - The user has access to his account only.
  - The application displays information only from the selected user.
  - The application displays the account data for the current user.
6. As a bank administrator I want the user to be able to log in so that I can track the users identity.
  - The administrator has a way of tracking user log in information.
  - The administrator can identify distinctive users.
  - The bank administrator has a way of tracking user information, location, login device etc.

- **App hosting** - The data the application has access to has to be stored someone off the user device and be available at all times. It has to be accessed in a secure way and support data editability.

1. As a banking application user I want to be able to access my data from anywhere so that I can use it on the go.
  - The user can access his information with a Internet connection.
2. As a bank administrator I want the application to be available both to me and the users from anywhere so that I can conduct business on the go.
  - The user has a application with a interface that is available anywhere.

  - The bank administrator has a administrator dashboard or interface he can interact with.

- **App interaction** - The user has to be given a interface that will be easily accessible and fully understood at a relatively quick time, it has to be designed keeping the requirement of intuitive design as one of it's main point.

  1. As a banking application user I want to see a balance for my accounts so that I know the balance after a transaction is applied. [3]
     - The user can view his accounts.
     - The user can view his account balance.
     - The user can choose a account to view.
     - The accounts are updated from the database.
  2. As a banking application user I want to see the transactions for my accounts so that I can review my past transactions.
     - The user can view the balance and account number of the account he chooses.
     - The user can view transaction info for the chosen account.
     - Account details are updated from the database.
  3. As a banking application user I want to be able to add payees to my account so that I will not have to re-enter their details.
     - The user can add a recurring payee to link to his account.
     - The payees will be stored and updated from the database.
  4. As a banking application user I want to be able to make transfers to other accounts so that I can conduct business on the go.
     - The user can make a transfer to another account.
     - The user can choose from his added payees.
     - The user can choose from his current accounts to pay from.
     - The user can specify the amount and any message to attach.
  5. As a banking application user I want to be able to view branches in my area.
     - The user can view a map with the locations of nearby branches of the bank.

Chapter 3

Methodology

3.1 Methodologies

Since this is a research project we have experimented with many different techniques to improve the creation of our application. We have utilised variations of extreme programming, Agile and Scrum methodologies. Agile Methodologies such as Scrum and Agile are tremendously important in the current technological sector which is why we have utilised the Agile Methodology during the creation of this application[4]. Iterative development is a subject of much scrutiny in modern software companies as the sector is looking to move away from the rigid and constrained "Waterfall" development process[5].

3.1.1 Agile Methodology

The Agile development method is much more suitable for developing applications as it focuses on creating a minimum viable product at the end of each development cycle. The minimum viable product is a standalone part of the application that has been designed, coded and tested within a single sprint allowing for much more control over, The development and evolution of the application. The application is evaluated at the end of every sprint and integrated into the full product. Git branches have been very beneficial for the task of integrating our sprint products into the application.

3.1.2 Scrum Methodology

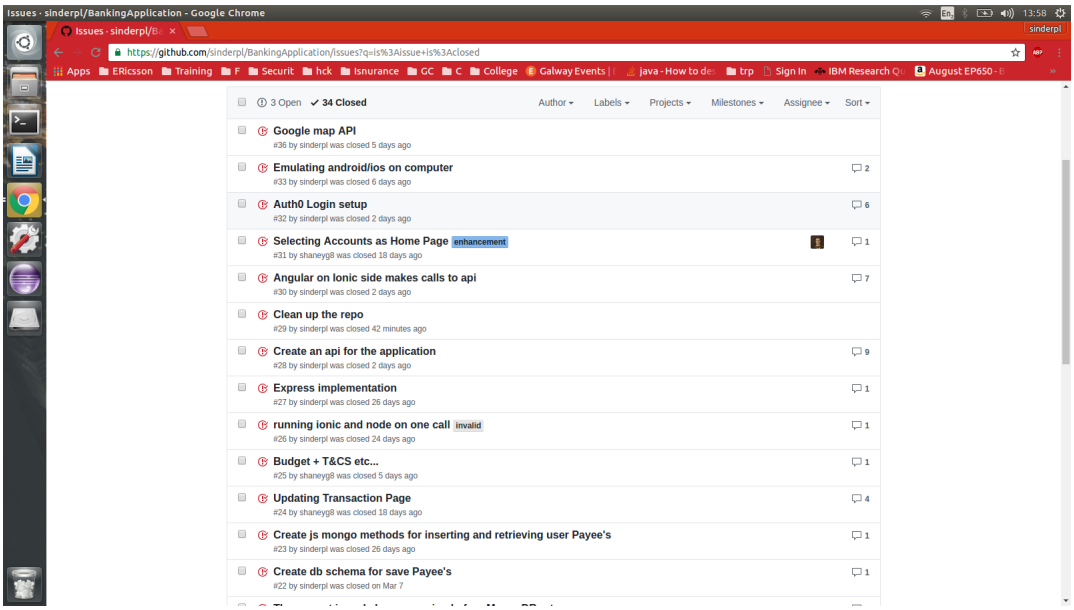
We have started as a two person team and have been joined by another member near the end of our development. The Scrum methodology which focuses on the interactions between individuals rather than code and documentation

has helped us co-operate as a team and integrate the newest member with our team. This focus on interactions also means we have regularly met with our project supervisor who acted as our client for this application, helping us guide the development process based on the products we were able to present to him. The team had many online channels of communication such as Facebook Messenger, GitHub issues and email through which we could co-operate and solve problems together. Outside of the regular meetings with the supervisor we have met regularly as a team, applying the stand up meetings of the mentioned methodologies during which we could discuss any issues that we have encountered as well as present the current progress of our designated tasks. At many of those meetings we would apply techniques similar to extreme programming into our sessions[6]. One person would be on the computer creating the code while the other was guiding him through the process and watch for any mistakes and errors. This is one of the most successful techniques we have used during the creation of this project, it allowed for much faster and smoother code creation while at the same time safeguarding from most errors that would not be spotted by one person.

3.1.3 Github Issues

We have utilised the Github issues option to keep track of current issues and stories to complete. This option allows for any user to see the issue and comment on it, creating a great interaction which allows for problem solving approached by the whole team. As you can see in the following figure 3.1 we have created many of these during the development process. These comments have many updates, links and discussion each of which not only enhances the creation of the app but also gives us a point of reference for the creation of this dissertation.

Each of these issues can also be given a label describing what kind of issue or improvement it is, in some cases this has been changed to a ”wontfix” or ”invalid” tag which means that it has been abandoned for one reason or another.


Issues · sinderpl/BankingApplication - Google Chrome
https://github.com/sinderpl/BankingApplication/issues?q=is%3Aissue+is%3Aclosed
3 Open
34 Closed
Author
Labels
Projects
Milestones
Assignee
Sort
Google map API
#36 by sinderpl was closed 5 days ago
Emulating android/ios on computer
#33 by sinderpl was closed 6 days ago
Auth0 Login setup
#32 by sinderpl was closed 2 days ago
Selecting Accounts as Home Page
enhancement
#31 by shaneyg8 was closed 18 days ago
Angular on Ionic side makes calls to api
#30 by sinderpl was closed 2 days ago
Clean up the repo
#29 by sinderpl was closed 42 minutes ago
Create an api for the application
#28 by sinderpl was closed 2 days ago
Express implementation
#27 by sinderpl was closed 26 days ago
running ionic and node on one call
invalid
#26 by sinderpl was closed 24 days ago
Budget + T&CS etc...
#25 by shaneyg8 was closed 5 days ago
Updating Transaction Page
#24 by shaneyg8 was closed 18 days ago
Create js mongo methods for inserting and retrieving user Payee's
#23 by sinderpl was closed 26 days ago
Create db schema for save Payee's
#22 by sinderpl was closed on Mar 7

(3.1)

Chapter 4

Technology Review

4.1 Front-End

4.1.1 Ionic Framework

The application is packaged as an Ionic 1 project, allowing for it to be cross-browser and device compliant. Ionic is a free and open source software and has many useful options allowing us to create a user friendly GUI and then emulate it on all three of our targeted devices: Internet browser, Android and IOS. Ionic is a complete open-source SDK for hybrid mobile app development.[7] Built on top of AngularJS and Apache Cordova, Ionic provides tools and services for developing hybrid mobile apps using Web technologies like CSS, HTML5, and Sass.[7]

We have originally planned to use Ionic 2 but after some issues decided to use the more tested first version. The other resources described in this section are mainly connected to this central framework.

The combination of this native framework combined with the out of the box functionality allows for the design of superb, user friendly interfaces which provide a smooth experience for any user. The developer does not need to work with CSS to make the application look and feel comfortable.

Additional features such like the Ionic View application help to test it by allowing you to push your application to the cloud then download it straight to the mobile device where it can be tested. Ionic 1 is also closely tied to AngularJS which is one of the other frameworks we have decided to use.

There are many advantages of using the ionic framework, because it is cross platform, when loaded on either IOS or Android it will change the look to match whatever platform. A desktop version can also be created of your app if done in the Ionic Framework.

4.1.2 AngularJS Framework

AngularJS is used to control the behaviour of the web pages from the creation of the app all the way to the controllers and routes. It is a structural framework used to create dynamic web applications. Co-operating closely with Ionic and HTML templates, it manages the behaviour of our application and its web pages utilising data binding techniques and dependency injections, eliminating a large amount of code necessary for the application to be created.

Through the application of the MVC (Model-View-Controller) pattern it decouples the application by generating the application on the client side, Then allowing the controller to manage the interaction between the HTML and the data model. Most frameworks implement MVC by asking you to split your app into MVC components, then require you to write code to string them up together again.[8] That's a lot of work. Angular implements MVC by asking you to split your app into MVC components, then just let Angular do the rest. Angular manages your components for you and also serves as the pipeline that connects them.[8]

With this application we will be focusing on working with controllers. Controllers in Angular are simple functions that have one job only, which is to manipulate the scope.[8] For example, you can use it to prefill data into the scope from the server or implement business logic validations.[8] Unlike other frameworks, controllers are not objects and don't inherit from anything.[8] In AngularJS, a Controller is characterized by a JavaScript constructor work that is utilized to enlarge the AngularJS Scope. At the point when a Controller is joined to the DOM by means of the ng-controller order, AngularJS will instantiate another Controller constructor, utilizing the predetermined Controller's constructor work.

4.1.3 Auth0 Framework

While OAuth is the industry-standard protocol for authorization we have decided to implement the Auth0 framework as our authentication option. The service which is used as a free Heroku add on.

It provides a database for your user login data while also allowing legacy databases to be used. It implements the JSON web token technology that can be used to connect with any API's you are using for added security.

You can connect any application (written in any language or on any stack) to Auth0 and define its Connection, the method used to authenticate the users of that application:[9]

- Custom credentials: username + passwords [9]

Figure 4.1: Auth0 [9]

- Social network logins: Google, Facebook, Twitter, and any OAuth2, OAuth1 or OpenID Connect provider [9]
- Enterprise directories: LDAP, Google Apps, Office 365, ADFS, AD, SAML-P, WS-Federation, etc. [9]
- Passwordless systems: Touch ID, one time codes on SMS, or email [9]

4.1.4 HTML5 language

HyperText Markup Language (HTML) is the code that pages are composed in.[10] It controls how content is shown on the web and is known as the foundation of the web. Each time you shop on the web, visit a blog or do a Google search, HTML code is working out of sight to make what you see on your screen.[10]

HTML is used by the Ionic framework and co-operates closely with Angular. It allows for the creation of native web pages that can be displayed on many devices.

Allowing for many different options such as templating, local data storage and video/audio support it is definitely a language beneficial to the creation of this application.

HTML5 makes creating accessible sites easier for two main reasons: semantics and ARIA. The new (some currently available) HTML headings like <header>, <footer>, <nav>, <section>, <aside>, etc. allow screen readers to easily access content.[11] Before, your screen readers had no way to determine what a given was even if you assigned it an ID or Class.[11] With new semantic tags screen readers can better examine the HTML document and create a better experience for those who use them.[11]

4.2 Middleware

4.2.1 NodeJS Language

From our research Ionic cannot be connected directly to MongoDB and most people use SQL databases as their data layer. We have decided to create a dedicated REST API server hosted on Heroku which can take in requests to the database through HTTP.

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications.[12] Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. [12] NodeJS is an Open Source, Cross Platform runtime enviroment which can develop server side. These applications are written in Javascript and can be run on Linux, Windows and OS X within the NodeJS runtime. Many major companies use NodeJS including the likes of Yahoo, Ebay, PayPal and Microsoft.

There are many features of NodeJS which include:

- **Asynchronous and Event Driven** - All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. Provides a context for your project[13]

- **Very Fast** - Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution [13]

- **Single Threaded but Highly Scalable** - Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server[13]

- **No Buffering** - Node.js applications never buffer any data. These applications simply output the data in chunks [13]

- **License** - Node.js is released under the MIT license [13]

4.2.2 Heroku

Because we are using Node.js Heroku comes in very handy. Heroku lets you deploy, run and manage applications written in Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP. [14] Heroku is used to scale applications and manage these applications and because Heroku is fully managed the developer doesn't have to worry about maintaining servers or hardware. Because we use GitHub to push our project updates, Heroku works in-sync with GitHub and instantly deploys apps as soon as you Git Push.

When you create an application on Heroku, it associates a new git remote, typically named heroku, with the local git repository for your application.[14] There are many other ways of deploying applications too. For example, you can enable GitHub integration so that each new pull request is associated with its own new application, which enables all sorts of continuous integration scenarios. Or you can use Dropbox Sync, which lets you deploy the contents of Dropbox folders to Heroku. Finally, you can also use the Heroku API to build and release apps. [14] Deployment then, is about moving your application from your local system to Heroku - and Heroku provides several ways in which apps can be deployed. [14]

4.3 Back-End

4.3.1 MongoDB

This database has also been provisioned from Heroku, it stores user info and account data. It can be reached through the REST API we created. MongoDB is a NoSQL technology. Before describing what MongoDB is first I will talk briefly about what NoSQL technology is about? Not only Sequential Query Language (NoSQL) is a database that stores and retrieves data in a different way compared to the conventional relational database which stores data in a tabular relation. NoSQL databases follow a schema-less data model, this allows for increased flexibility and scalability compared to relational databases.

- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time [15]
- The document model maps to the objects in your application code, making data easy to work with [15]

- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data [15]
- MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use [15]
- MongoDB is free and open-source, published under the GNU Affero General Public License [15]

Because MongoDB is a document database which means a collection can hold different documents, each document can differ from one another in the case of size and content and fields, resulting in Schema less. A single structure object also in Mongo is clear and has no complex joins. Mongo also uses a document-based query language so when querying it is just as powerful as SQL.

4.3.2 Hosting

Heroku Hosting

Heroku hosts our REST API server and provisions and connects the Auth0 and MongoDB. It is connected to our REST API branch of the project, building and hosting the newest version whenever we push to the GIT repository.

Auth0 Hosting

Auth0 provides its own database for the user login data as well as providing a interface for database modification. This allows for a much smoother control over users, combining the ability of creating/deleting users with the option to monitor where their login requests are coming from. The usage of their database frees up some of the limited space we have on mongoDB.

MongoDB Hosting

MongoDB provides its own limited (50 MB) sandbox server space for us to store our data on. This allows the hosting of a small amount of the data we will need, especially after we outsource the login to Auth0 databases. The sandbox environment we are using is running on a shared VM (Virtual Machine), being perfect for development, prototyping and to get familiar with MongoDB. [16]

4.4 Additional Resources

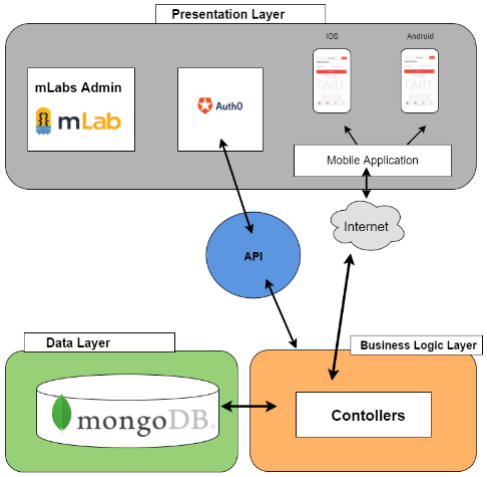
4.4.1 Google Maps API

Our app features a 'Location' page which utilizes the Google Maps API on which we have marked the location of our bank.

Chapter 5

System Design

5.1 Architecture



5.2 Presentation Layer

In the presentation layer we use the mLabs add-on for Heroku to manage our user's information. Here we can do all the CRUD operations like create and manage users and all the content of their accounts and user details. As this is a banking application users do not have permission to create their own accounts. After researching existing banks mobile and web applications and from being ourselves users of said technologies we learned the most secure way is to set up the customer's accounts for them. Also in the presentation layer we have the mobile app itself which can be ported using Ionic View to several different platforms. The platforms we are concentrating on are the

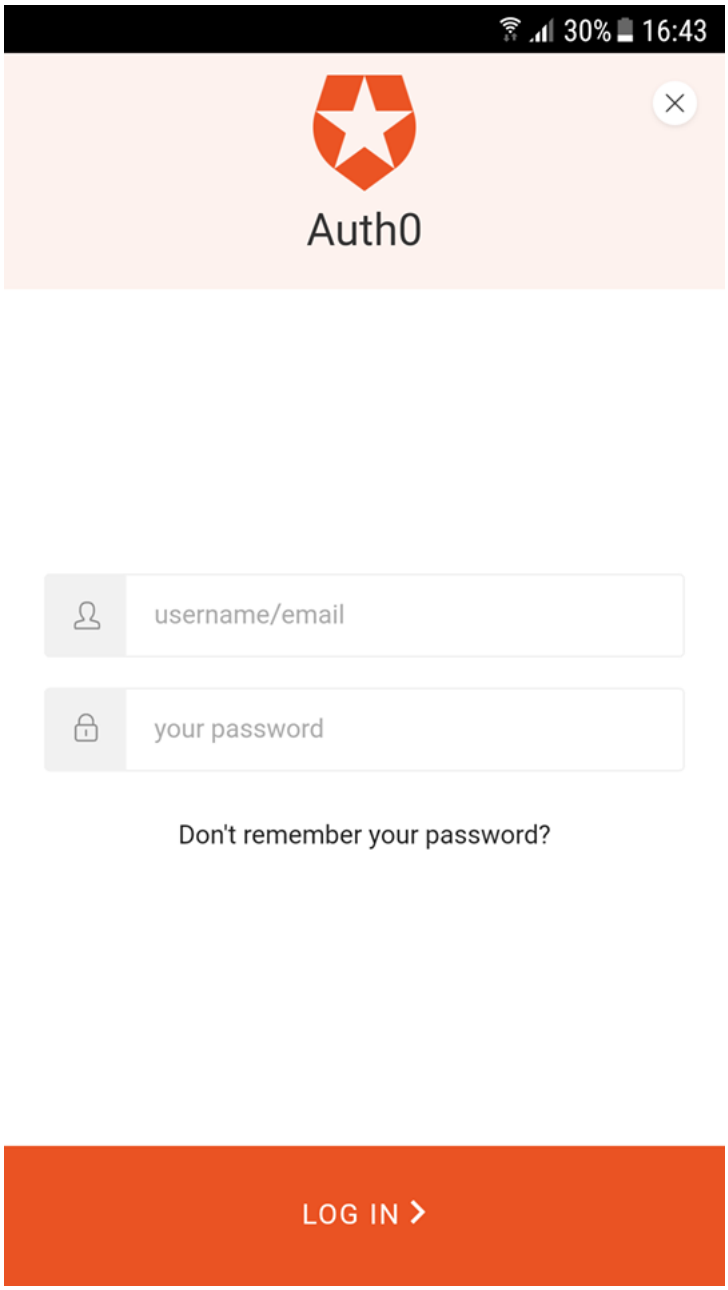
two main market shareholders, Android and IOS (Apple). The mobile apps will be able to use HTTP POST requests to fetch the data dynamically from our custom built API.

5.2.1 mLabs used for Admin

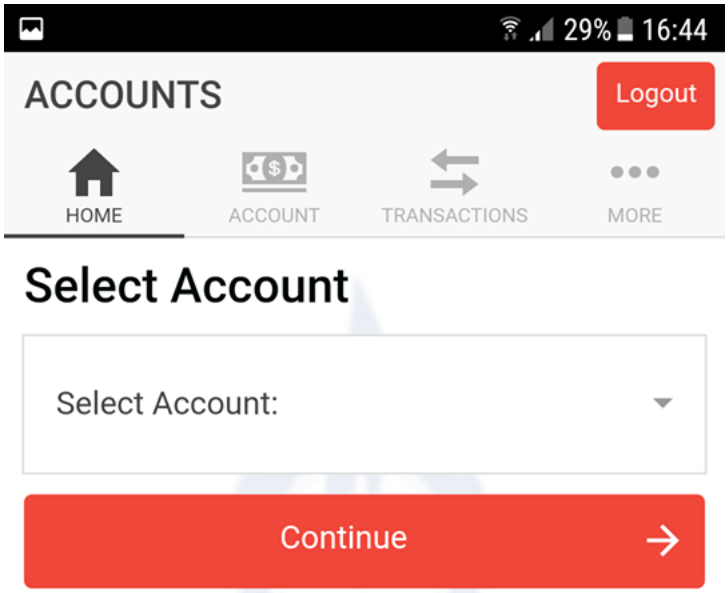
Insert a few pics of mlabs

We decided to use Heroku to host our mobile application. Heroku allows for add-ons so we can manage our MongoDB database we used mLabs. We are using mLabs as our administration page for the CRUD operations on our users and their accounts in the MongoDB database. Alan set up a Heroku account and set Shane and Dara as collaborators allowing them admin access to the hosted application. This also allows collaborators use mLabs to perform CRUD operations on the database records.

5.2.2 Ionic Application

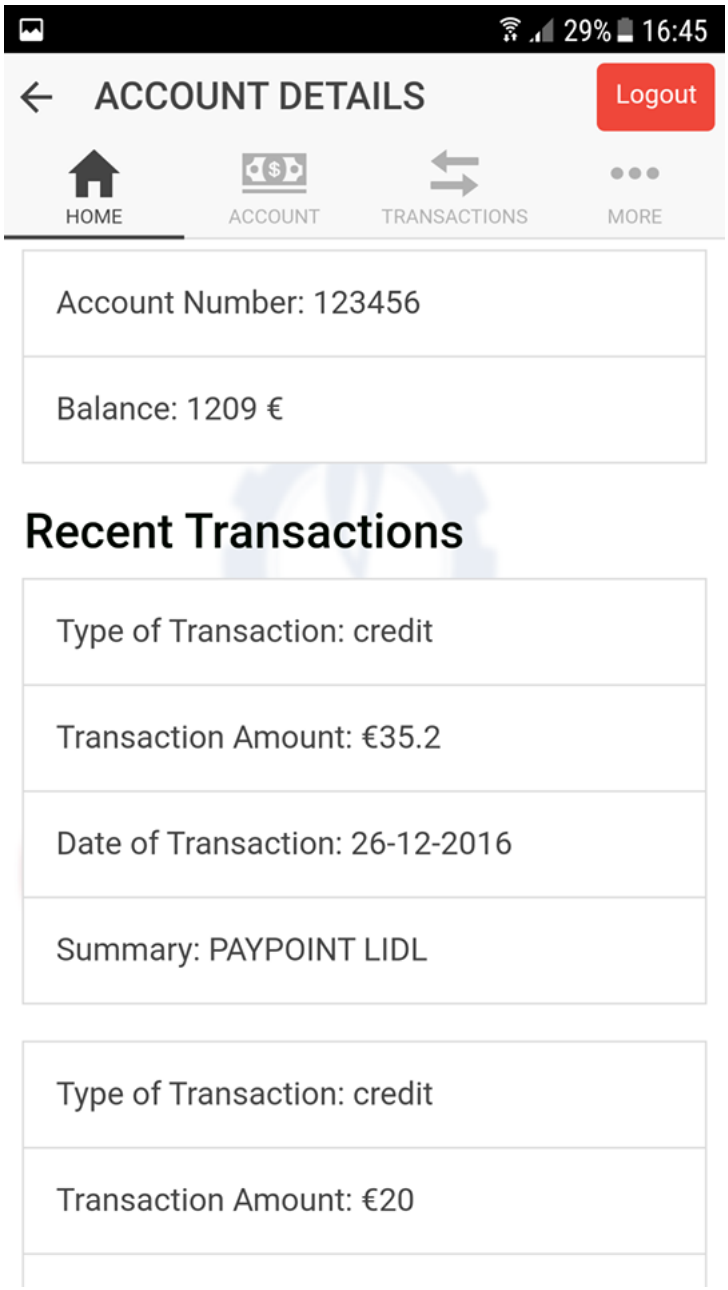


On opening the application the user will always be brought to the Login page. The user will be prompted to login using their email and unique password. This login page uses AuthO authentication. Following a successful login the user is redirected to the account selection page.



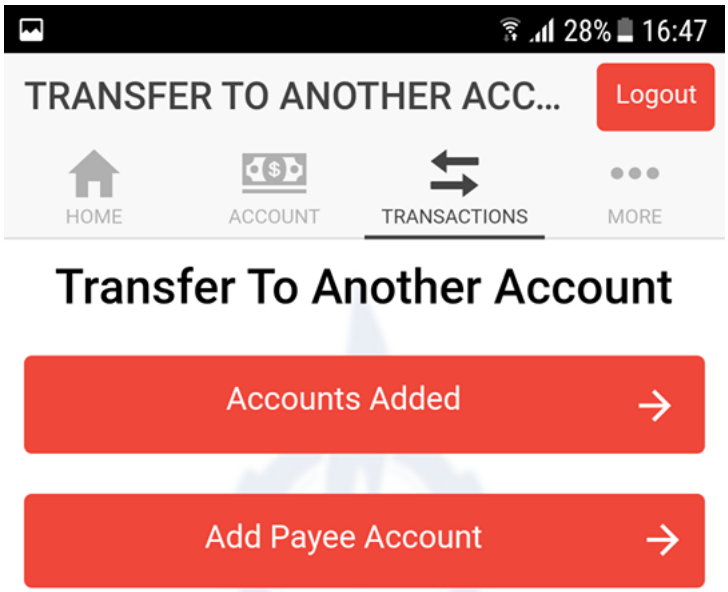
This is the landing page of the application which page allows the user to select which of their branch accounts they want to view given that they might have more than one account i.e Current account, Savings account etc. On selecting which of their accounts they wish to view they can click the Continue button which will redirect you to the Account Details page.

There is also a toolbar on the top or bottom of the page depending on which platform you are viewing the application on. This toolbar contains four sections, HOME, ACCOUNT, TRANSACTIONS and MORE. The HOME tab contains the Select Account page. ACCOUNT contains the Account Details page which gives the logged in user's details and a summary of recent transactions. The TRANSACTIONS tab contains pages to control money transfer to other payees and adding payees. The MORE Tab contains two sub-pages for information of the creators of the application and Location page that shows the branch location of GMIT Bank.

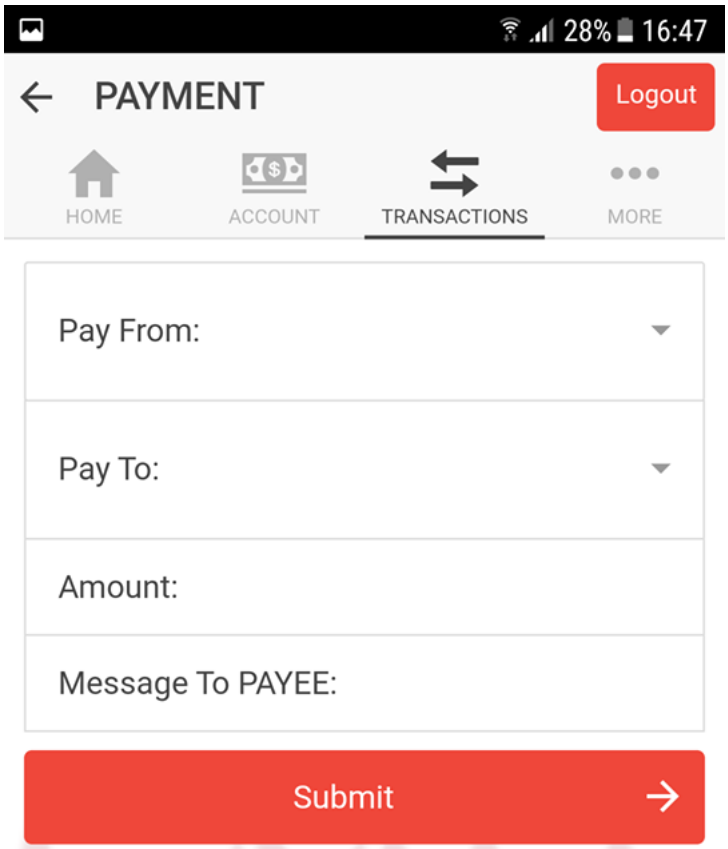


The Account Details Page gives the user details at the top comprised of Name, Account Number of selected account and the current balance. Also there is a section dedicated to the list of transactions that have being made on this account. They let the user know the date the transaction occurred, the type of transaction either debit or credit, the amount of money received or removed from the account and a brief summary of the transaction itself.

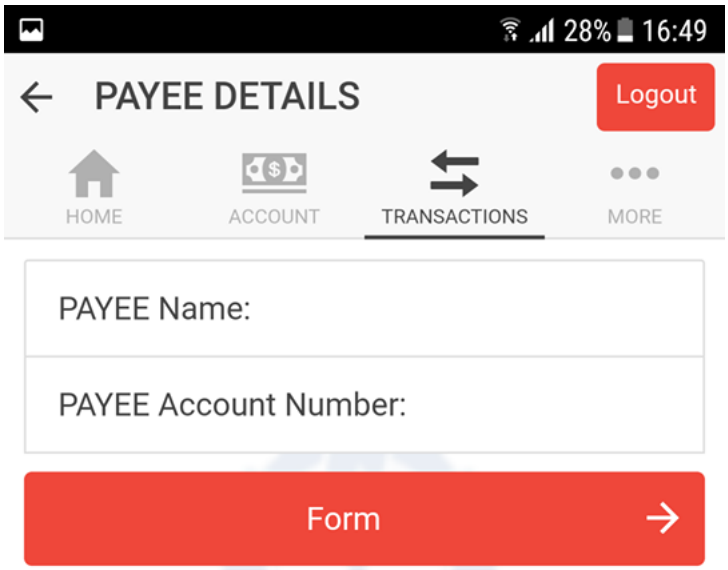
Screenshot of Transfer to Another Account Page Gives the user an option to move to the options page for transferring money to other GMIT Bank user's accounts or creating another payee which you want to transfer money to.



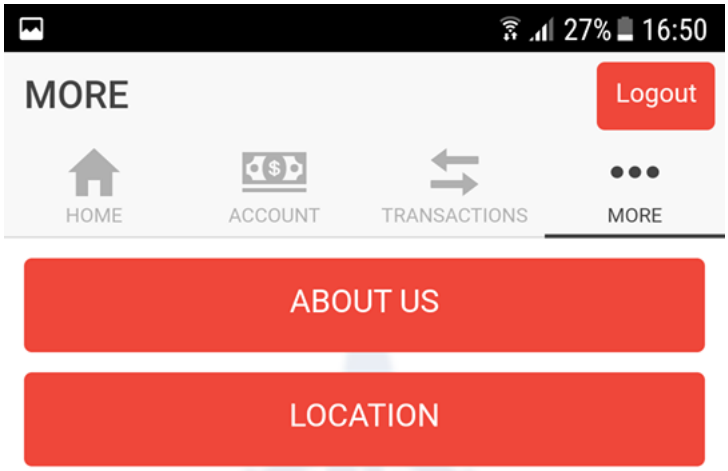
On this page there is two button options. One is the Accounts added page where the user can go to transfer money to existing payees you have stored in your account details. The second button is Add Account where you can add a new payee to your account details.



In the Payment page the user can transfer money from their own accounts to existing payees. The first section allows you to select which of your accounts you want to transfer money from. The second section is a drop-down list that lets you select who you want to pay the money to from a list of existing payees. Third section is a text area where you can enter the amount of money you wish to transfer. The final section allows you to send a summary of what the money transferred was for and will show up in your payee's transaction summary.

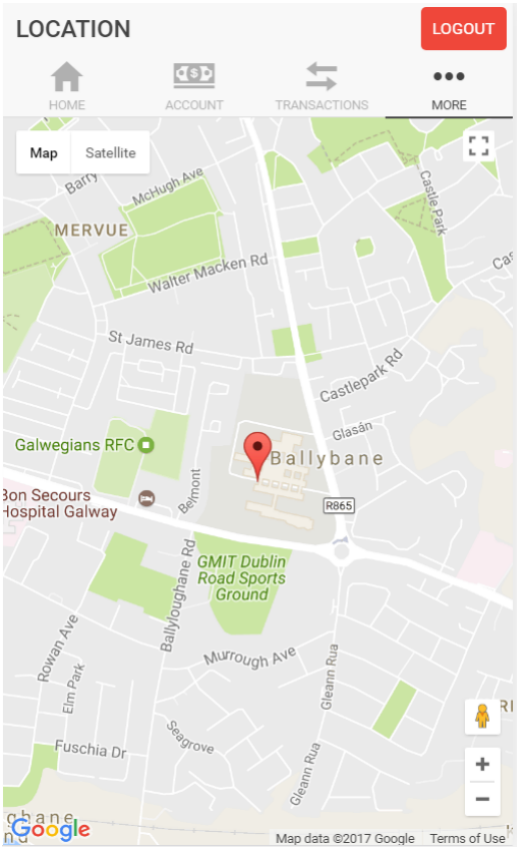


This page allows a user to add a payee to their account details so they will be able use the Payment page to transfer money to the payee's account. In this page there is two textareas where the user can enter the payee's first name and last name. The section textarea lets you enter the payee's Account number and it gets stored by clicking the Continue button. The Cancel button allow you to cancel adding a new payee and redirects the user back to the Transfer to Another Account page.



The more tab contains two buttons. One is for the Developers information and the other is for the location of the GMIT Bank branch.

This page contains information about the three GMIT students which developed this mobile application.

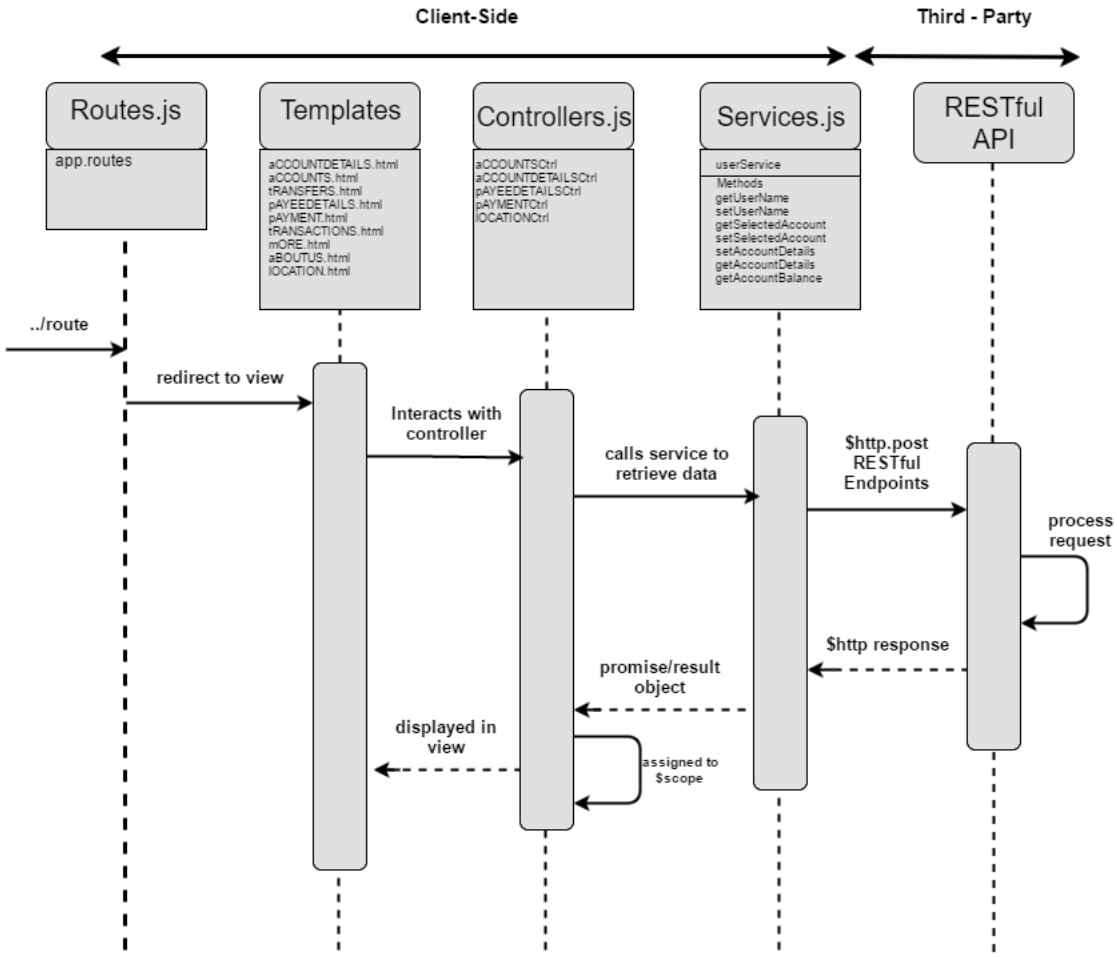


The locations page uses the google maps JavaScript API to show where the GMIT Bank branch location is and places a marker on it. When the marker is pressed information about the branch is brought up on the screen.

5.3 Business Logic Layer

Above we have talked about the Presentation Layer of this application. In this section we will talk about the Business Logic layer and how everything is connected. JavaScript is the main language that the business logic has been written in. Each bit of functionality has its own controller. The controller.js file is used to communicate between the front-end and the back-end of the application. Each controller sends a POST request to the API depending on which action the user wants. For a project we decided to only use the HTTP POST request for fetching data because it is more secure than a traditional GET request. A HTTP GET request is less secure than a POST request because it is sent as part of the URL. This means it can be saved in the browser history and server logs in plaintext.[17]

5.3.1 Class Diagram



5.3.2 Controllers

Now I will talk about the controllers themselves and what each controller does in respects to the application. When the user has successfully logged in they are brought to the Accounts pages where they are presented with options of which of their accounts they wish to select. Like most banks a member can have several different account ranging from Savings account to Current accounts. The selection of accounts is handled by the aCCOUNTSCtrl. A HTTP POST request is sent to the API and the data which is requested is returned from the MongoDB database.

```
.controller('aCCOUNTSCtrl', ['$scope',
'$stateParams', '$http', 'userService','authService' ,
function ($scope, $stateParams, $http, userService, authService) {
userService.setUserName(authService.userProfile.username);
//This function listens to changes in the account selection and maps
//it to a value in the scope for further use
$scope.chooseAccount = function(accountSelection) {
//Set the value in the service to the new account value
userService.setSelectedAccount(accountSelection);
}

//userService.checkProfile(authService);a

//HTTP request for the user account data
$http({
//method to use
method: 'POST',
//where API is hosted
url: 'https://mobilebanking.herokuapp.com/user',

origin: 'http://localhost:8100',
//data type being received
dataType: "JSON",
//getting username to use using userService from the services.js file
data : "username="+userService.getUserName() ,

headers: {'Content-Type':
'application/x-www-form-urlencoded; charset=UTF-8'}
}).then(function successCallback(response) {
//Function activated if data is succesfully returned
//Set the ionic Scope variables for this page based on
// the data to display
// when the response is available
userService.setAccountDetails(response.data.accounts);
$scope.accInfo = response.data.accounts;

}, function errorCallback(response) {
  console.log('failure');
  // called asynchronously if an error occurs
  // or server returns response with an error status.
});
```

```
}])
```

On successfully selecting which account they wish to see the user clicks continue and are brought to the Account Details page. This page gives the user details of which account they have selected followed by all the recent transactions that has being performed on said account. The controller that loads the information for this page is the aCCOUNTSDETAILSCtrl. This action is performed using another POST request and retrieving the data in the accounts section of our MongoDB database.

```
.controller('aCCOUNTDETAILSCtrl', ['$scope',
'$stateParams', '$http', 'userService','authService' ,
function ($scope, $stateParams, $http, userService) {

var accs = userService.getAccountDetails();
for (x in accs){
if(accs[x].accid == userService.getSelectedAccount()){
  $scope.accountNumber = accs[x].accid;
  $scope.accountBalance = accs[x].balance;
}
}


$http({
//method to use
method: 'POST',
//where API is hosted
url: 'https://mobilebanking.herokuapp.com/user',

origin: 'http://localhost:8100',
//data type being received
dataType: "JSON",

data : "accid="+userService.getSelectedAccount() ,

headers: {'Content-Type':
'application/x-www-form-urlencoded; charset=UTF-8'}
}).success(function(response) {
```

```
$scope.transInfo = response.transactions;

// this callback will be called asynchronously
// when the response is available
}, function errorCallback(response) {
  console.log('failure');
  // called asynchronously if an error occurs
  // or server returns response with an error status.
      });
}])
```

The next major controller we have implemented is the Add Account page in the Transactions section of the application. To be able to transfer funds between accounts firstly the user must add a payee's name and account number to their banking records. This is handled by the pAYEEDETAILSCtrl which receives the new name and new account number from a form on the HTML page. We make use of AngularJS's ng-model to bind the input-label back to the controller and ng-submit to execute the submitPayee function in the controller. The API needs three pieces of information to POST the data to MongoDB which are the current user's name, new payee's name and account number. Using the service userService.getUserName() we can get who's details we need to add the new payee to. When we have the user's name then the new payee's name and account number is added to the payee section of that user's collection in the database.

```
.controller('pAYEEDETAILSCtrl', ['$scope', '$stateParams',
'$http','userService','authService' ,
function ($scope, $stateParams, $http, userService) {

$scope.submitPayee = function(PayeeName, PayeeAccountNumber){

//Create a custom HTTP POST request to add a new payee to the logged in user
$http({
//Type of request - used POST since it is more secure than GET
method: 'POST',
//The URL to which call will be made
url: 'https://mobilebanking.herokuapp.com/payee',
//The origin of the requeset (Current host)
origin: 'http://localhost:8100',
```

```
//The type of data being sent
dataType: "JSON",

//The data sent with which to query
data : "username="+userService.getUserName()+
"&name="+PayeeName+"&account="+PayeeAccountNumber,
//The header for the call being made
headers: {'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'}
}).then(function successCallback(response) {
//Function activated if data is succesfully returned
console.log('success');

$scope.payeeConfirmation = "The payee has been added."

}, function errorCallback(response) {
console.log('failure');
$scope.payeeConfirmation = "An error has occured while adding to the database."
// called asynchronously if an error occurs
// or server returns response with an error status.
});
    }
}])
```

Once a user has added payees to their account details they are able to transfer funds to that person. The controller that handles this is the pAYMENTCtrl. This controller requires the user to to select which account they wish to transfer money from. which one of their payees they want to transfer the money to.The amount of money they wish to transfer to the payee and finally a brief summary of why they are making this transfer.

```
.controller('pAYMENTCtrl', ['$scope',
'$stateParams', '$http', 'userService','authService' ,
function ($scope, $stateParams, $http, userService) {
 $scope.sendTransaction = function(payfrom, payto, amount, message){
//getting the current user using userService
var currentUser = userService.getUserName();
var accbalance ;
//getting which account to use
var accs = userService.getAccountDetails();
```

```
for (x in accs){
if(accs[x].accid == payfrom){
accBalance = accs[x].balance;
}
}
//setting up wat is required to complete the transfer
var dataString = "username="+currentUser+
    "&currentbalance="+accBalance+
    "&accountid="+payfrom+
    "&amount="+amount+
    "&summary="+message+
    "&type="+"credit"+
    "&date="+"27/04/2017"

console.log(dataString);
$http({
//Type of request - used POST since it is more secure than GET
method: 'POST',
//The URL to which call will be made
url: 'https://mobilebanking.herokuapp.com/transaction',
//The origin of the requeset (Current host)
origin: 'http://localhost:8100',
//The type of data being sent
dataType: "JSON",

//The data sent with which to query
data : dataString,
//The header for the call being made
headers: {'Content-Type':
'application/x-www-form-urlencoded; charset=UTF-8'}
}).then(function successCallback(response) {
//Function activated if data is succesfully returned
console.log('success');

}, function errorCallback(response) {
console.log('failure');
// called asynchronously if an error occurs
// or server returns response with an error status.
});
$scope.paymentStatus = "Transfer complete.";
}
```

```
$http({
//Type of request - used POST since it is more secure than GET
method: 'POST',
//The URL to which call will be made
url: 'https://mobilebanking.herokuapp.com/user',
//The origin of the requeset (Current host)
origin: 'http://localhost:8100',
//The type of data being sent
dataType: "JSON",

//The data sent with which to query
data : "username="+userService.getUserName(),
//The header for the call being made
headers: {'Content-Type':
'application/x-www-form-urlencoded; charset=UTF-8'}
}).then(function successCallback(response) {
//Function activated if data is succesfully returned
console.log('success');

//select which account you want to transfer money from
//select which existing payee you want to transfer money too
$scope.selectPayee = response.data.payees;

}, function errorCallback(response) {
console.log('failure');
// called asynchronously if an error occurs
// or server returns response with an error status.
});


}])
```

A nice feature we added to the application using the Google maps JavaScript API we were able to incorporate the location of the GMIT Bank branch. For the project we set the campus of the Dublin road as the branch location. For this to work we had to create a lOCATIONCtrl. Also we had to include the ng-Cordova libraries into the lib folder in the project. Using google.maps.LatLng() and the latitude and longitude coordinates we were able to locate where the branch should be placed on the map and also show where it exactly is using the google.map.marker() to have a pointer fall on the

location too. We were also able to give information about the branch such as a description and opening hours using google.maps.InfoWindow().[18]

```
.controller('lOCATIONCtrl', ['$scope',
'$stateParams','$cordovaGeolocation','authService',
function ($scope, $stateParams, $cordovaGeolocation) {
var options = {timeout: 10000, enableHighAccuracy: true};
//getting current position
$cordovaGeolocation.getCurrentPosition(options).then(function(position){
//setting latLng variable to GMIT's location
var latLng = new google.maps.LatLng(53.27842930000001, -9.011151100000006);
//setting centre of map to be GMIT
//how far out the zoom is and usin the road map style Google Map
var mapOptions = {
  center: latLng,
  zoom: 15,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};
//creating a new map
$scope.map = new google.maps.Map(document.getElementById("map"), mapOptions);

google.maps.event.addListenerOnce($scope.map, 'idle', function(){
  //adding a marking which will land on the branch location/GMIT
var marker = new google.maps.Marker({
  map: $scope.map,
  animation: google.maps.Animation.DROP,
  position: latLng
});
//infoWindow describing the banks opening hours
var infoWindow = new google.maps.InfoWindow({
  content: "<h1>GMIT Bank, Dublin Road</h2>"+
            "<p>Opening Times: </p>"+
            "<p>Monday:     9:00 - 17:00</p>"+
            "<p>Tuesday:    9:00 - 17:00</p>"+
            "<p>Wednesday:  9:00 - 17:00</p>"+
            "<p>Thursday:   9:00 - 17:00</p>"+
            "<p>Friday:     9:00 - 17:00</p>"+
            "<p>Saturday:     Closed</p>"+
            "<p>Sunday:       Closed</p>"
});
```

```
google.maps.event.addListener(marker, 'click', function () {
  infoWindow.open($scope.map, marker);
});
});

}, function(error){
console.log("Could not get location");

  });

}])
```

5.4 Data Layer

For this project we decided to use **MongoDB** as our database and use the **Heroku** add-on **mLAbs** to handle the CRUD operations that we need to perform as administration. The schema which we as a group decided on that would work best was to have two collections of data. The collections are as follows **Users** and **Accounts**.

5.4.1 Users Schema

Users which will hold information required for login along with additional information like **name**, **accounts** and **payees**. Accounts and payees themselves have sub-documents. Accounts is broken into three further sections **accounttype** which can be whatever type of account you hold with GMIT Bank. Also the **accid** which is the account number for that type of account and the final piece of information in the accounts sub-document is the **balance** showing how much is in this account. The payees as we have called them are the people that you wish to transfer money too. The sub-document only contains two fields **name** and **account**. Name is the name of your payees and account is just their personal account number which you will be transferring funds to.

The database schema for the user looks like the following, this is based on our example user:

```
"username": "alanniemiec",
```

```
"name": "Alan Niemiec",
"pin": "2345",
"deviceid": "A234wqe21e",
"accounts": [
    {
        "accounttype": "Current",
        "accid": "123456",
        "balance": 1209
    },
    {
        "accounttype": "Savings",
        "accid": "654321",
        "balance": 5855.56
    }
],
"payees": [
    {
        "name": "Shane Gleeson",
        "account": "765432"
    },
    {
        "name": "Chris Weir",
        "account": "999888"
    },
    {
        "name": "David Hickey",
        "account": "147741"
    }
]
```

Legend for the user database schema:

- Username - The login username for the authorization also used to find the user document.
- Name - The name and surname of the user.
- Pin - Used to validate the user login after he authorises with the login framework. This was going to be used to encrypt the user data but we didn't have time to implement it.
- DeviceID - This was to be the IMEI ID of the device the user is using,

only the devices on with the ID's included would be allowed to access the user information.

- Accounts - A array of sub documents describing the details of user accounts.
  - AccountType - The type of account, either current or savings.
  - Accid - The unique id of the account by which it can be found in the Accounts document.
  - Balance - A quick reference balance for the main page of the application.
- Payees - The list of payees the user has saved.
  - Name - The name of the saved payee.
  - Account - The account number of the saved payee.

5.4.2 Accounts Schema

Accounts is closely tied with the Users collection because the accounts sub-document which is saved in Users is broken into separate account objects here. Accounts are broken up into five main records, **ownername** the name of the account holder, **accounttype** which is what type of account it is e.g. Savings, **accid** is the account number, **accbalance** is the cash balance in the account and **transactions** which has a sub-document off it. The transactions are made up of four separate records, **date**, **type**, **amount** and **summary**. These four tell the user when the transaction took place, was it debit or credit, how much the transaction came to and a brief summary of where the transaction took place.

The database for the Account document looks like the following based on a fictional account by one of our mock users:

```
"ownername": "Alan Niemiec",
"accounttype": "Savings",
"accid": "654321",
"accbalance": 5855.56,
"transactions": [
    {
        "date": "20-4-2014",
        "type": "debit",
```

```
            "amount": "35.20",
            "summary": "PAYPOINT CENTRA"
        },
        {
            "date": "5-1-2017",
            "type": "credit",
            "amount": "20.00",
            "summary": "TRANSFER TO 897723662"
        }
]
```


Legend for the account database schema:

- Ownername - The name of the owner of the account.
- Accounttype - The type of the account.
- Accid - Unique id of the current account, used as a query parameter.
- Accbalance - The balance of the account.
- Transactions - A array of transaction sub documents.
  - Date - The date of the transaction.
  - Type - The type of transaction.
  - Amount - A numerical representation of the transaction amount.
  - Summary - A short word summary for the transaction

5.5 Deployment - Heroku

For our project we decided through research and consultation that Heroku would suit our needs for hosting our application. Ionic does not directly connect with MongoDB so for this to work we had to create an REST API. The great thing about using Heroku is it could host our REST API server and it could also provision and connect AuthO which we use for login authentication and MongoDB. Heroku has a great feature where you can install add-ons as they are called once you have your project set up. Both AuthO and mLabs - MongoDB were easy to install and work with. We have it set up to the REST API branch of the project in Github. When there was ever a modification done and the work was pushed to this branch of the repository Heroku would build and host on this latest version.

Chapter 6

System Evaluation

6.0.1 Testing

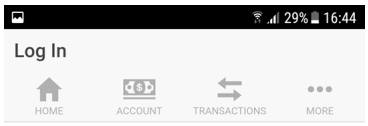
Most of the testing on this application was done manually. We have however made heavy use of Google Chrome debug menu which allowed us to view logs from our application. Since our application was cross platform we had to check the logs on a phone as well. We have emulated and connected Android phones to the browser and debugged them that way. Ionic view was also utilised to check the applications on the phones, this did not however provide us with any other output than the visual feedback.

6.0.2 Limitations

The functionality of the application is solely dependent on the user having connection to mobile data or to an Internet connection. The application needs Internet connection to be able to contact the online Web Server (API) to be able to retrieve data from the MongoDB database. Also our application uses Google Map's JavaScript API to show the branch location of our bank. For security purposes we felt it best that this application should only be used when the user has full connection to Internet. Storing data locally on the user's device could make it easy for sensitive information to be obtained if the device was to be lost and or stolen. Another limitation is the amount of storage we have been supplied with using MongoDB's sandbox server. The total space allotted for our database is 50mb which in a real world banking application would not last very long.

- The application encounters a problem on during usage on Android and IOS. After a successful login, you should be redirected to the /Accounts page but it seems the call-back URL is not implemented properly in

Auth0 interface. This causes the page to redirect to the /Login HTML page. This is not a breaking issue as the menu bar is still present and the user can simply redirect back the proper application by clicking on the Home tab. See the following figure to see the failed login tab.



- The date for the transactions is currently hardcoded, this has been simply because I have not had time to implement the correct code for it. While AngularJS provides a Date object, it creates it in format that includes time and time zone. I have not had time to figure out a way to filter this variable.
- If the users sends a transaction the database is updated. The current session in the app is not however. The user will only see the updated balance once he logs back into the application. The transaction will be added and updated in the application straight away. This is due to some code getting mixed up and not having time to fix it back.
- (Resolved) At the time of writing this there is a certain issue with the Auth0 framework. The search query user name for our database has till now been received from Auth0 once the ”lock” widget was called and successful authorization has occurred. User metadata could then be called back with the following code:

```
userService.setCurrentUser(authService.userProfile.username);
```

Currently this does not return anything even though the user has logged in successfully according to the Auth0 Interface. This is a big issue and we are working on trying to fix it. The code used to work but seems to have changed all of a sudden, this could be a issue of the Auth0 team implementing some sort of a update. The username is currently hardcoded in the application.

This issue has been partially resolved, the username is no longer hardcoded. We are not however fully sure of how JWT web tokens behave so the application does not log in properly at times. This new development is out of our control as it depends on whether we get the auth0 profile back or not. Would require further investigation but there is no time.

- Android users need to be wary when first using the /Location tab on the phone. If the user does not allow the application to access location settings, he will not be able to view the Google Map. A possible workaround for this would be to identify the location permissions on the mobile device and whitelist the Ionic View application.

- The project was mainly developed on Linux, therefore certain errors can arise when running npm install on Windows. If so the user needs to follow the onscren instructions and run :

```
npm rebuild node-sass
```

6.0.3 Problems Encountered

One of the hardest things we had to deal with when we started this project was trying to use Ionic 2. We spent a long time trying to get all three of our machines configured to work with Ionic 2. We extensively researched ways to install and configure ways to get it working but nothing we tried seemed to work. Our thoughts were that it was clashing with the original version of Ionic that was installed already on our machine but nothing on the Ionic website or forums could help our cause. After conferring with several lecturers in our course and that one in particular teaches the Ionic module we were prompted to go use the more stable version one. The time we lost on trying to do what we thought would be the easiest part of the project installing the environment to work in would have been valuable to have towards the latter stages of the projects.

6.0.4 System Interaction

Front End

Each HTML page in the application is managed by its AngularJS object equivalent called a controller. The Ionic application co-operates closely with AngluarJS by the use of ”scope” variables and ”ng” operator tags. These operators are used in many different AngularJS options:

- Scope are variables through which the HTML view is updated. these can be assigned to anything from strings to object arrays.

```
scope.accInfo = response.data.accounts;
```

- Angular services - A service is a way of modelling a object and methods that will be kept in memory for the duration of the application session. This means the object, its data and methods can be shared across controllers unlike the scope variables which are not available to the scope of a different controller.

```
        .service('userService', [function() {
        //Current user userName
        var userName";
        var selectedAccountValue ;
        var accountDetails;

        //Return the userName
        this.getUserName = function() {
          return userName;
        }
    }
```

- NG - A shorthand for Angular, this option allows for manipulation of HTML pages. It can for example be used to call a method in the controller when a form is submitted, parsing the specified data fields in the process:

```
<form class="list"
ng-submit="sendTransaction(payfrom, payto, amount, message)">
```

  Another use in our project was the repeat option which allows for us to pass in a object array in our scope variable and NG then runs it through a for loop printing out the details:

```
<select
    ng-model="accountSelection"
    ng-change="chooseAccount(accountSelection)" >
<option ng-repeat="info in accInfo"
    value={{info.accid}}>{{info.accounttype}} : {{info.balance}}  €
</option>
</select>
```

- HTTP requests - The front end communicates with the API by sending HTTP POST request to the database, this was determined to be safer than GET request which we used initially.

```
http({
//The type of request
method: 'POST',
//URL of the API this is aimed at
url: 'https://mobilebanking.herokuapp.com/user',
//Origin of the call
origin: 'http://localhost:8100',
//Payload data type
dataType: "JSON",
//The data sent, this might look like URL
//but that is just the type
//of encoding
data : "username="+userService.getUserName()" ,
//The headers
headers: {'Content-Type': 'application/x-www-form-urlencoded;'}
})
```

Middleware

In the case of the Middleware we have created a Hosted API which interacts with both the user interface through previously described HTTP requests and the database through Mongoose, an object modelling wrapper for the NodeJS native driver. [19]

- The API connects to the database and maintains a connection:

```
MongoClient.connect(
"mongodb://Test:Test@ds139187.mlab.com:39187/heroku_vh3f7203",
(err, database) => {
```

```
  db = database;
  console.log("db connection ready");
 })
```

- We then extract the data from the POST request sent by the user application, in this case it extracts the username from the request body so that we can use it in the MongoDB search query:

```
    var query = { "username" : req.body.username};
```

- The query is then sent to my custom universal method for finding singular document objects along with the name of the collection it needs to traverse:

```
    function findOne(collectionName, query, callback){

  db.collection(collectionName).findOne(
  query, function (err, item){
    if (err){
      callback(err);
    }
    else{
      //Return the data with no errors
      callback(null, item);
    }
  });
}
```

  This way I can re-use these lines of code for nearly every other method. The function also returns a ”callback” or a promise which tells the original method to await for the result at this address. This is necessary as NodeJS is asynchronous, it will move on with the code before the database has a chance to return the result. The result is then returned to the caller method as the ”item”.

  The original caller method then returns a JSON response to the user application which handles the rest on that end:

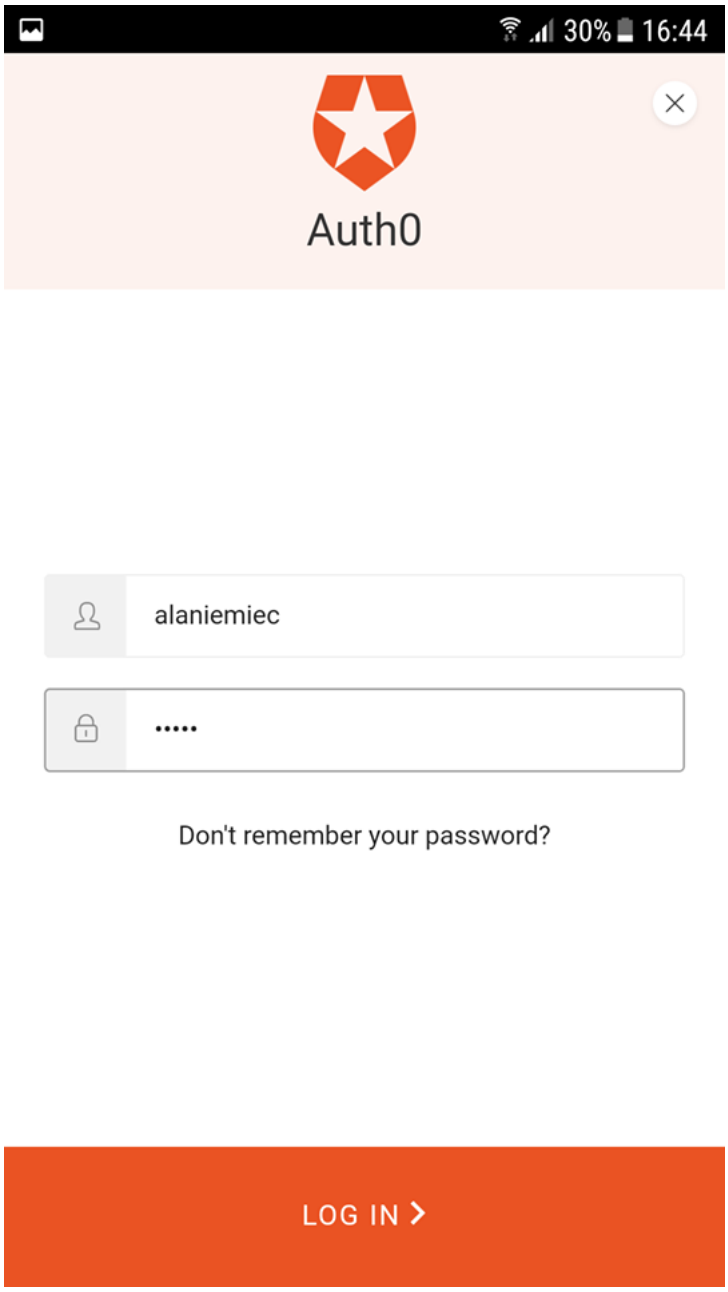
```
    res.type('json');
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept");
    res.json(item);
```

- In the case of more sophisticated operations such as adding a sub document to a array in a document we utilise a push method, it works similarly to push in normal programming languages, appending the new object to the end of the array:

```
    db.collection("Accounts").update(
      {"accid" : req.body.accountid},
      {push: {"transactions":  {"date" : req.body.date,
      "type" : req.body.type, "amount" : req.body.amount,"summary" : req.boo
    }}}
  )
```

Back End

This section encompasses both the MongoDB database and Auth0 database. The case of MongoDB has already been described previously, the Auth0 login database communicates directly with the user application. The user application has a library of scripts, one of which handles the authorisation interaction by initializing the Lock widget:




```
lockProvider.init({
        clientID: AUTH0_CLIENT_ID,
        domain: AUTH0_DOMAIN,
        options: {
          auth: {
            redirect: true,
```

```
            //http://localhost:8100/#/page1/accounts
            redirectUrl : location.href + '#/page1/accounts',
            sso: false,
            params: {
              scope: 'openid',
              device: 'Mobile device'
            }
          }
        }
      });

Logging in and out is also easy :
  authService.login();
  authService.logout();
```

This widget allows for the input of user login information which is then sent to the Auth0 servers for validation. The script then saves a JWebToken in local storage along with certain user data.

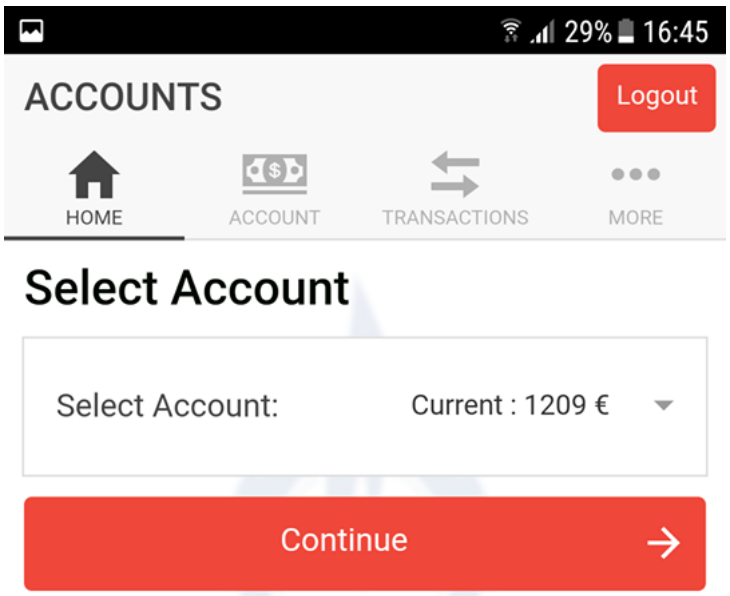
6.1 Evaluation of Acceptance Criteria

- **App portability** - The application has to be available on a wide range of devices: browser, IOS, Android as well as being suitably designed for these devices. It should be available remotely from a mobile device through an Internet connection.

    1. As a banking application user I want to be able to access the application on any device I choose so that I am not constrained to a single device.
        - The application view scales well for a web browser. - This has been achieved.
        - The application view scales well for an Android devices. - This has been achieved.
        - The application view scales well for a IOS devices. - This has been achieved.

        This user story can be considered fully completed. The application interface is fully scalable for any display.
    2. As a banking application user I want to be able to access my data from anywhere so that I can use it on the go.

– The application can be used on a mobile device. - This has been achieved with minor limitations as set out in the previous section.
– The application can be used with an Internet connection. - This has been achieved, an Internet connection is necessary for the application to run.



Here is an example of what the app looks like on an Android phone screen. This user story can be considered fully completed, the application is usable on all the specified devices with just an Internet connection.

- **App security** - The application has to be secure and implement as much security as we are able to implement. This metric is the main objective of our research and the banking theme.

  1. As a bank administrator I want to know that my user information is secure on my database so that I can provide services to many customers without compromising their personal information.
     - The application database is secure from unwanted users. - This can be considered completed as far as mLab's MongoDB servers are secure.
     - The application is not accessible to third parties. - This can be considered completed, only the administrator has access to the database administration.

     The user story can be considered completed, the database is secure from tampering thanks to MLab's security.
  2. As a bank administrator I want to know that the user data is safe in transfer so that they can access it from anywhere.
     - User information gets to the user device without the possibility of a man in the middle attack. - This has been achieved to a degree, there is always a possibility of hacking the packets but we have evolved our security from HTTP GET requests which parsed the info in the URL to HTTPS POST requests which transfer the payload inside the packet, encrypted.
     - Sensitive data is not saved to the mobile device. - The mobile device does not store any user data. It is all dynamically loaded from the database .

     This user story can be considered mostly completed. Security implementations need to be revised regularly but I feel like it has been completed to the best extent of our knowledge. Here is an example of our HTTP request:

```
http({
//The type of request
method: 'POST',
//URL of the API this is aimed at
```

```
url: 'https://mobilebanking.herokuapp.com/user',
//Origin of the call
origin: 'http://localhost:8100',
//Payload data type
dataType: "JSON",
//The data sent, this might look like URL
//but that is just the type
//of encoding
data : "username="+userService.getUserName()" ,
//The headers
headers: {'Content-Type': 'application/x-www-form-urlencoded;'}
})
```

3. As a bank administrator I want to be able to add and delete users on my end so that no one can sign up without verification.
   - The bank administrator has a way of creating users.
   - The bank administrator has a way of deleting users.
   - The bank administrator has a way of tracking user information, location, login device etc.

   This user story is all completed by the usage of Auth0 framework and the database schema. The administrator can parse the User schema into the database, fill it with necessary details then create an Auth0 user and map this username to the database. This gives the administrator the control over who signs up for a account, similarly to the real world banks.

4. As a bank administrator I want to be able to manipulate the database so that I can correct any errors that arise in transactions.
   - The bank administrator has access to the database.
   - The bank administrator can revise account information.
   - The bank administrator can modify the database information.

   The bank administrator has access to the database, he can edit the details as needed. This is completed.

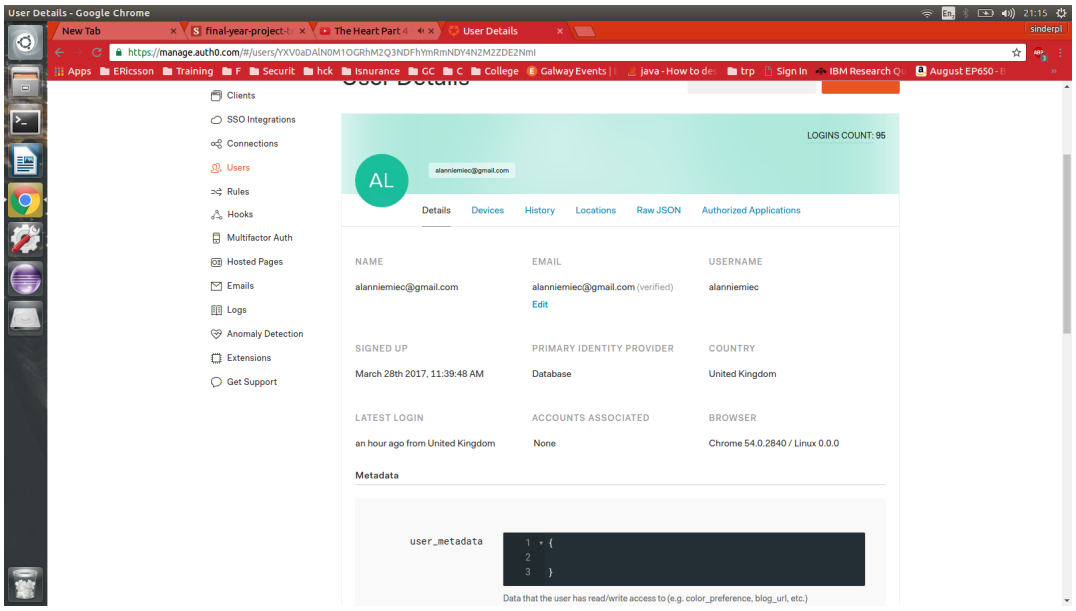
5. As a banking application user I want to be able to log in to my personalised account.
   - The user can log in to his account through authentication procedures.
   - The user has access to his account only.

- The application displays information only from the selected user.
- The application displays the account data for the current user.

As lined out in the Limitations, there is currently a bug that does not allow for the user to log in, the username is hardcoded. As it stands this is not fully complete.

6. As a bank administrator I want the user to be able to log in so that I can track the users identity.
   - The administrator has a way of tracking user log in information.
   - The administrator can identify distinctive users.
   - The bank administrator has a way of tracking user information, location, login device etc.

   This has been successfully implemented apart from the user login limitation previously described. The user can still log in so the Administrator can track his details and location but I would not consider this fully complete.



- **App hosting** - The data the application has access to has to be stored someone off the user device and be available at all times. It has to be accessed in a secure way and support the ability to edit data.

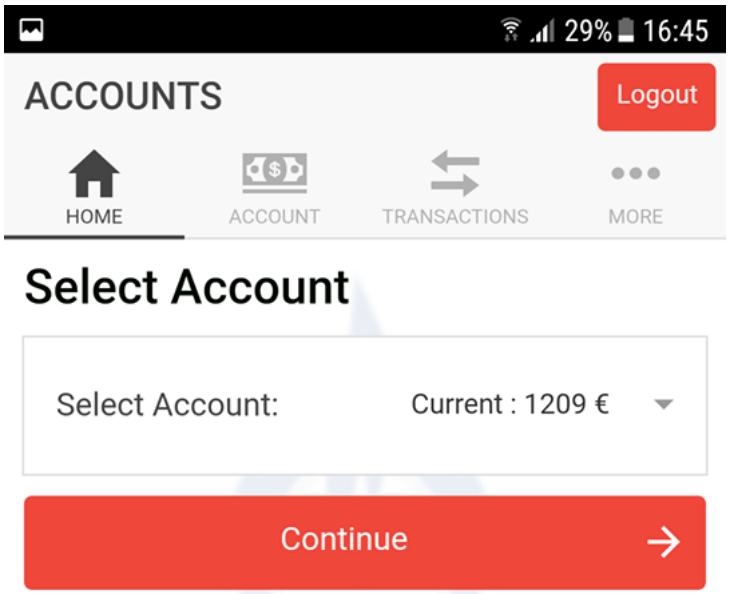
1. As a banking application user I want to be able to access my data from anywhere so that I can use it on the go.
   - The user can access his information with an Internet connection.

   This user story is complete, the user can access the information with a just an Internet connection.
2. As a bank administrator I want the application to be available both to me and the users from anywhere so that I can conduct business on the go.
   - The user has an application with an interface that is available anywhere.
   - The bank administrator has an administrator dashboard or interface he can interact with.

   This user story is complete. The user has the application and the administrator has the Auth0 interface.

- **App interaction** - The user has to be given an interface that will be easily accessible and fully understood at a relatively quick time, it has to be designed keeping the requirement of intuitive design as one of its main point.

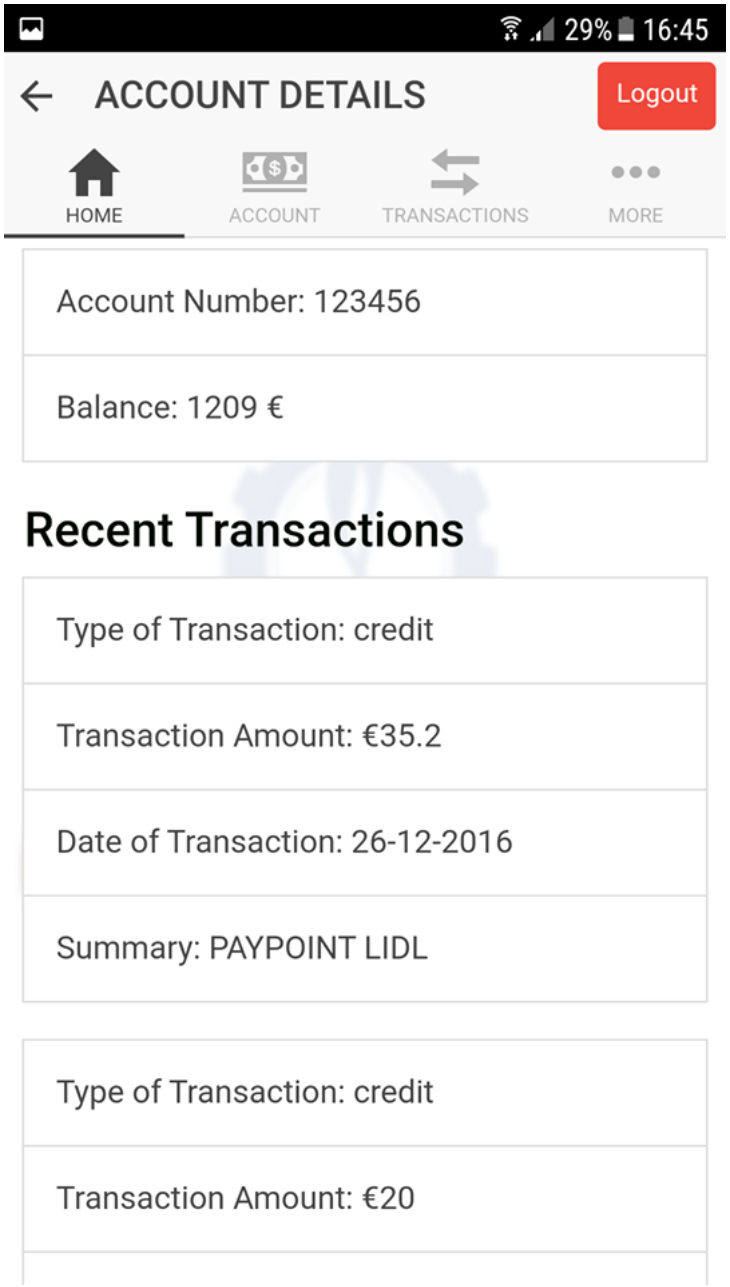
  1. As a banking application user I want to see a balance for my accounts so that I know the balance after a transaction is applied. [3]
     - The user can view his accounts.
     - The user can view his account balance.
     - The user can choose an account to view.
     - The accounts are updated from the database.



This user story is complete, user accounts and balance are displayed successfully.

2. As a banking application user I want to see the transactions for my accounts so that I can review my past transactions.
   - The user can view the balance and account number of the account he chooses.

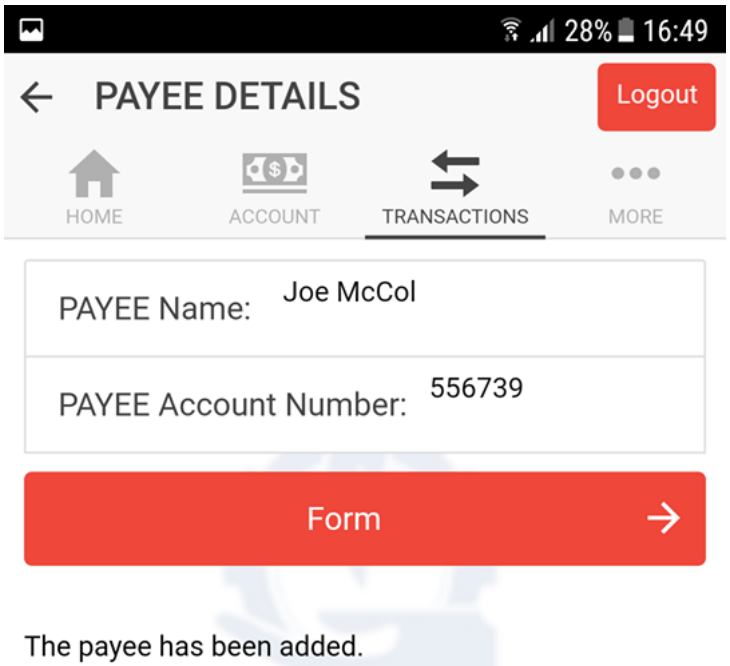
- The user can view transaction info for the chosen account.
- Account details are updated from the database.



This user story is complete, the user can view his transaction history successfully.

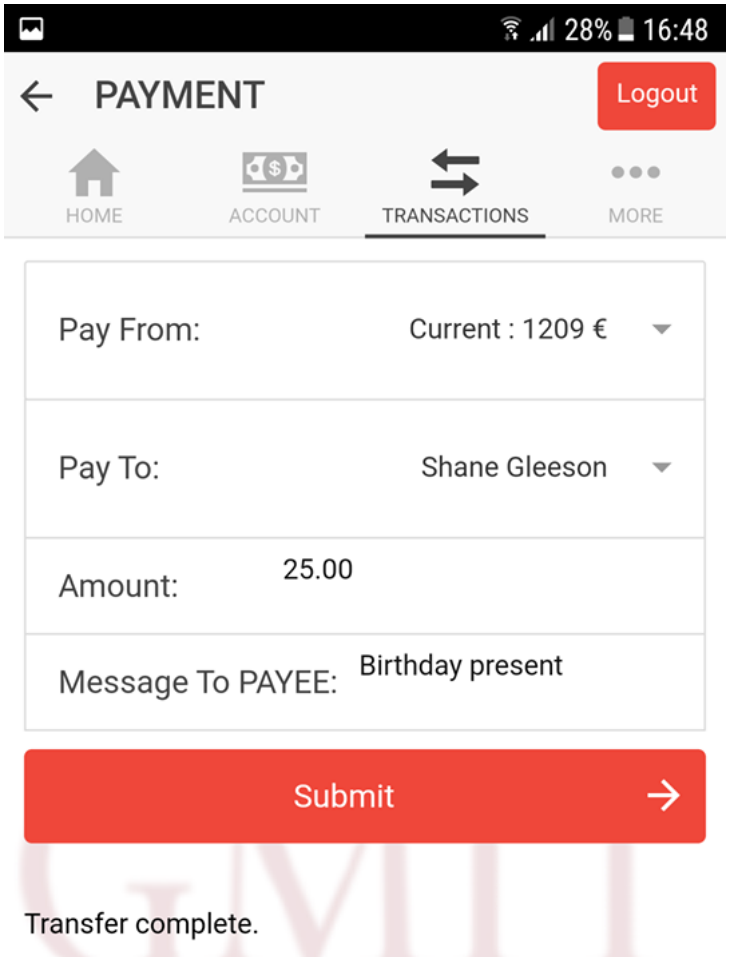
3. As a banking application user I want to be able to add payees to my account so that I will not have to re-enter their details.

– The user can add a recurring payee to link to his account.
– The payees will be stored and updated from the database.



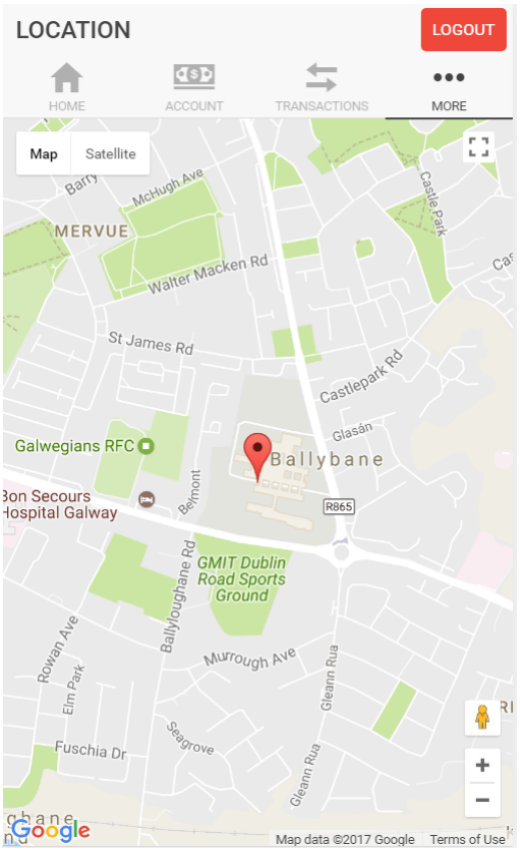
This user story is complete, the user can add payees successfully. There is some small feedback text for the user on the left of the screen.

4. As a banking application user I want to be able to make transfers to other accounts so that I can conduct business on the go.
   - The user can make a transfer to another account.
   - The user can choose from his added payees.
   - The user can choose from his current accounts to pay from.
   - The user can specify the amount and any message to attach.



The user can successfully transfer to another account.

5. As a banking application user I want to be able to view branches in my area.
   - The user can view a map with the locations of nearby branches of the bank.



Chapter 7

Conclusion

When we got together as a group and sat down to line out exactly what was needed it seemed that it was going to be more straight forward than how it actually turned out. This project really challenged us in ways that will be of great benefit to us going on in our professional careers outside of college. Constant planning and task management was definitely one of the main learning outcomes that will be taken from this project. Even tasks as simple as picking which framework we would like to use to develop our initial idea of a banking application was full of trials and tribulations. We collectively in one of our first meetings decided that we would like to use Ionic version 2 to create of application. This led us to our first of many hurdles. After spending a long portion of time researching how to install Ionic version 2 we keep hitting walls with the general installation of the software. None of our research was giving definite answers why it would not install properly and countless hours of trawling through the Ionic Website was still turning up no answers. Our final port of call in solving our dilemma was to approach one of our lecturers who is very familiar with the Ionic framework and with his advice we decided the most intelligent thing we could do was to go with the more stable and reliable version 1. This really gave us an insight as programmers and more so as a team that what lay ahead might not be as straight forward as it first seemed.

This trend continued when we finally had our technologies chosen. Ionic was going to be our front end (user interface), MongoDB was going to be our back-end (database) and to communicate between the two JavaScript and Angular would fill this role. And here is where another major obstacle was discovered. After more research on using these technologies together we realised that Ionic cannot communicate directly with MongoDB so we now knew we need to create an API to handle this. Alan took it upon himself to create the RESTful API while Shane and Dara came up with and designed

the front-end. As Alan created the API we got together and decided it would best and more beneficial if we used Heroku to host it for us and also the idea for using mLabs for handling the CRUD actions and AuthO for handling logging in or user and authentication. Using delegation of the work made life easier and at all time everybody in the team knew which aspect they should be researching and then developing. Also by spreading the work load each member of the team got to learn more about new technologies or reinforced what they already knew about others.

The Mobile Application has been developed with the main goal of managing our user's bank account. Any changes made to the data or requests sent to retrieve data are instantly reflected due to synchronisation. The mobile app gives the user a concise overview of their account activities and keeps track of their spending habits by logging every transaction they make into the database and displaying it when they have successfully logged in and selected their account. It also incorporates a Google Map's JavaScript API to show the location of the bank's branch.

This project has been a great insight to what lies ahead in the near future for us when we venture out into industry. Throughout this whole project we have been learning skills that will be very beneficial in the coming few years. Learning how to plan out a project from start to finish and what are the best methods of achieving your goals. Having regular meetings to see at what stage each member of the team is at and laying out a plan for the following few days keeps the group focused on the job ahead. The use of Github for keeping track of each other's code and also using the Issues section to flag any problems you are having which can be discussed at the next meeting. We also made use of pair programming where it was applicable to take on a challenge and solve it together. Regular meetings with your group supervisor was very helpful and also kept the team focus driven on what needed to be achieved. This really simulated what it will be like when working for a company and gives a real world point of view of expectations put on you and deadlines that need to be met. All in all it has been a great experience and we as a group went through every scenario possible on the way to achieving our final goal. Looking back is there things we would have done differently? Of course there are but this was all part of the learning outcomes that this module was about. We will all come out of this with a greater appreciation and understanding for what our jobs will be once we enter industry. With the way the IT sector is ever evolving one's ability to adapt and learn will be the greatest factor on whether or not you succeed in it as a career.

7.1 Further Development

Like every project there is always extra functionality and features which you would have like to add if time had allowed. Although we are very happy with the application there were things for the sake of time and deadlines that we just did not get time to implement. One such thing is when a user adds a transaction to their account the date field in the transaction array does not update by itself but is hard coded. Unforeseen difficulties were reached towards the end of the development stage which were in our eyes more important to get working so this is one decision that needed to be made for the good of the project on a whole. We feel it would not take much time to rectify but in the fear other aspects of the project would suffer over this small issue we decided best to leave it for now.

Another way Dara has talked about improving the user's experience would be to utilise more to the functionality of the Google Maps API. Making navigation to the branches for the users could be made easier by implementing directions. Added information like estimated time of arrival could also have been included. Also for more branches we could have used every GMIT campus in the Connaught region and pinpointed them as possible location to take care of your banking needs.

Another idea we had was to implement a budget page. This page would give the user a graphical representation of what way their money is being managed. If we were to continue on developing this application this would be one feature we would definitely include due to its visual nature. The use of colours could represent when your finances were mismanaged or managed correctly. Each week would be broke down and displayed in a graph to show the user what days you are over spending and this might encourage the students which use our application to be less frugal with their money. Sometimes looking at figures on a statement or your account balance on an ATM does not give you a real insight into your spending habits. This proposed feature would allow the user to check their account after any spending event and receive a detailed breakdown of where, when and how much was spent.

Bibliography

[1] Wikipedia, "Mobile banking." [Online]. Available: https://en.wikipedia.org/wiki/Mobile_banking

[2] P. Schueffel, "Taming the beast: A scientific definition of fintech." [Online]. Available: http://www.open-jim.org/article/view/322/221

[3] S. Mamoli, "On acceptance criteria for user stories." [Online]. Available: https://nomad8.com/acceptance_criteria/

[4] Linchpin, "Agile methodology." [Online]. Available: https://linchpinseo.com/the-agile-method/

[5] Marklogic, "Waterfall to agile." [Online]. Available: http://www.marklogic.com/blog/3-reasons-to-transition-from-waterfall-to-agile/

[6] E. Programming, "Extreme programming: A gentle introduction." [Online]. Available: http://www.extremeprogramming.org/

[7] Wikipedia, "Ionic (mobile app framework)." [Online]. Available: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)

[8] Sitepoint, "10 reasons to use angularjs." [Online]. Available: https://www.sitepoint.com/10-reasons-use-angularjs/

[9] Auth0, "Auth0 overview." [Online]. Available: https://auth0.com/docs/overview

[10] E. B. G. Learning, "Why is html5 important." [Online]. Available: https://www.gomolearning.com/blog/what-is-html5-why-is-it-important/

[11] P. C. Codrops, "10 reasons to use html5." [Online]. Available: https://tympanus.net/codrops/2011/11/24/top-10-reasons-to-use-html5-right-now/

[12] NodeJS, "Nodejs." [Online]. Available: https://nodejs.org/en/

[13] T. Point, "Node.js quick guide." [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm

[14] Heroku, "How heroku works." [Online]. Available: https://devcenter.heroku.com/articles/how-heroku-works

[15] MongoDB, "What is mongo." [Online]. Available: https://www.mongodb.com/what-is-mongodb/

[16] ——, "Mongodb plans." [Online]. Available: http://docs.mlab.com/plans/

[17] N. J. Kate T, Pooja Sehgal, "Get vs. post." [Online]. Available: www.diffen.com

[18] I. Framework, "Google maps." [Online]. Available: https://ionicframework.com/docs/native/google-maps/

[19] MongoDB, "Mongoose." [Online]. Available: https://devcenter.heroku.com/articles/nodejs-mongoose

.1 Github URL's:

- URL for the main application:

`https://github.com/sinderpl/BankingApplication`

- URL for the API part of the application

`https://github.com/sinderpl/BankingApplication/tree/herokuAPI`