

**TAPP**

**(Transact-App)**

**Crypto Payment System**

**BEng(H) in Software & Electronic Engineering**

**Project Engineering**

**Year 4**

**Elise Santella**

G00387170

Atlantic Technological University

2022/2023

**Project Graphic:**

Customer scanning QR Code presenting by Merchant during Transaction

Graphical user interface

Description automatically generated

**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

A picture containing text

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Acknowledgements**

I would like to express my gratitude and appreciation to the ATU Project Engineering lecturers, Brian O’Shea, Michelle Lynch, Niall O’Keeffe and Paul Lennon who have provided numerous amounts of support and guidance throughout the academic year.

I would especially like to thank my supervisor Brian O’Shea, who was incredibly helpful in providing feedback as well as using his expertise to help give direction in certain times during my project when extra support was required.

Lastly, I would like to thank my fellow classmates who were there for support and to give feedback as well as offer advice throughout the year.

**Table of Contents**

[1. Summary 6](#_Toc134013055)

[2. Poster 7](#_Toc134013056)

[3. Introduction 8](#_Toc134013057)

[4. Background 8](#_Toc134013058)

[5. Project Architecture 10](#_Toc134013059)

[6. Project Plan 11](#_Toc134013060)

[7. Software Features 11](#_Toc134013061)

[7.1 Front End Software 11](#_Toc134013062)

[7.1.1 Navigation 11](#_Toc134013063)

[7.1.2 QR Code & QR Scanner 11](#_Toc134013064)

[7.1.3 Post Functions 11](#_Toc134013065)

[7.2 Back End Software 11](#_Toc134013066)

[7.2.1 Database 11](#_Toc134013067)

[7.2.2 Authentication 13](#_Toc134013068)

[7.2.3 API Endpoints 16](#_Toc134013069)

[7.3 Native Crypto Tokens 19](#_Toc134013070)

[7.3.1 Migrations 20](#_Toc134013071)

[7.3.2 Contracts 21](#_Toc134013072)

[7.3.3 Testing 24](#_Toc134013073)

[7.3.4 Sale Deployment 27](#_Toc134013074)

[8. Ethics 31](#_Toc134013075)

[9. Conclusion 32](#_Toc134013076)

[10. References 32](#_Toc134013077)

# Summary

The desire of Transact (TAPP) is to facilitate the transfer of funds between ATU students and various businesses inside the ATU campus, using a native TAPP crypto currency. TAPP is a mobile app that allows for the customer, the student, to pay the merchant, ATU run businesses, for their products or services using native TAPP tokens instead of traditional money.

The project consists of two versions of the mobile app, one merchant version and one customer version. Both mobile apps have a signup and login feature with a secure authentication procedure that encrypts the password into a database. TAPP mobile app allows the merchant to enter in the price owed by the customer, generating a QR code containing the price inside. The customer app can scan the merchants QR code, triggering a transfer of funds in a matter of seconds. The price is deducted from the customers balance and the merchant balance will increase by the price amount, returning both users to their home page to view their current balance and latest transaction in their account.

The approach towards the project was hesitant at first because the outline included various technologies and methods that were unknown. Creating a timeline was crucial to the project and helped in staying on track throughout the college year. There were certain aspects of the project that altered the timeline but keeping an organised log and good project management allowed for adjustments in the timeline and helped prioritise features.

The main technologies used throughout this project were React Native framework to build TAPP mobile apps using JavaScript which was hosted on the Expo platform. A Node.js webserver was used for the backend to communicate between the two apps as well as the MongoDB database which uses a JavaScript library called Mongoose to connect the database with the webserver. These technologies were hosted on VS code along with the Ethereum ERC-20 standard approved language Solidity. Solidity was used to build the TAPP tokens, paired with JavaScript and HTML to create a website where the user could buy TAPP tokens. The Ethereum accounts were obtained using the testing accounts from Ganache and were accessed on the website using Google Chrome extension MetaMask so that the user could log into their account to purchase tokens from the website.

The project accomplished two mobile apps with secure authentication where the customer was able to pay the merchant based on a price specified by the merchant. The customer also has an option for a top up of funds and both apps displayed their current balance and most recent transactions. TAPP tokens was built along with a website using a testing environment that would allow different user accounts to purchase the tokens.

TAPP is a way for students around ATU to pay for ATU products and services using native tokens which take away the third party, banks, out of the transaction process. It also takes away the need for any hardware to be needed, using only two phones, one for the customer and one for the merchant, for the transaction. Thus, making the process more efficient and effortless.

# Poster

Figure 2.1 Project Poster

# Introduction

Transact App (TAPP) is a wireless mobile application that allows ATU students, the customer, to purchase products and services from ATU supported business, the merchant, using a native TAPP crypto currency.

The goal of this project was to create a quick and easy payment system using TAPP tokens, that would transfer funds from the customer to the merchant in a more efficient manner than other traditional methods. Traditional methods of payment consist of time wasted by a third party, banks, to authorise the payment as well as incur fees to one’s account. There are also the expenses for the hardware such as EFTPOS machines that the merchants would need to purchase from the banks so that a sale can be made.

This led to the motivation for TAPP as it hopes to offer a de-centralised platform for users with fast processing times and no transaction fees for either customer or merchant. There will also no longer be a need for any hardware as using TAPP is a completely wireless process where only a mobile phone is needed so that the user can sign into their account to perform a transaction.

This report will give background into the project followed by an Architecture diagram to help visualise how to project should work in terms of software flow. A project plan is listed to give insight on the process of building the project, leading into the software features enlisted within the project. These features are broken down into front-end, back-end and token creation describing the technologies and methods used for each feature. Diagrams of the project and code will assist the viewer in painting an image on how each feature works within the project. The report will then talk about the ethical considerations of the project before concluding the project outcome. References will be listed at the end of the report.

# Background

Blockchain technology offers a de-centralised platform for users with fast processing times, secure crypto wallets, and no transaction fees. Therefore, it is no wonder that on the global crypto exchange Coinbase there are currently over 100 million users [1]. As a result, many are claiming crypto currencies as the future of the finance sector with internet banking possibly being a thing from the past. This creates potential relevance for applications such as TAPP where traditional ways of payment and transactional hardware are no longer needed. Instead, these traditional methods can be replaced by native TAPP tokens and a completely wireless transaction system.

Though in the initial stages of the project, there was some thought put into whether hardware should be included into the project such as a Near Field Communication (NFC) Tag, which is a piece of hardware that allows for wireless transmission of data. After some deliberation and mapping out an architectural diagram, listed below in Figure 5.1, it was clear that a complete transaction system based on software would be much more useful to the user and offering something more unique than other traditional payment methods – no hardware.

Before implementation of the project, weeks of research went into developing an understanding of crypto currencies and the blockchain. It was important to first gain some knowledge on what crypto is and the blockchain it is stored on. Cryptocurrency is digital currency used as an alternative form of payment [2]. It uses encryption algorithms that make crypto transfer incredibly secure and has no third party to regulate them. A blockchain hosts crypto currencies and acts like a database that maintains a secure and decentralised record of each transaction [3]. This information is shared over a computer networks nodes, which are the communication points of a network.

From the research, there were three potential ways to build a crypto currency. The first was to create your own blockchain and native currency, second was to modify the code of an already existing blockchain and lastly create a new cryptocurrency and host it on an already existing blockchain [4]. Hosting the native currency on an already existing blockchain was chosen because it allowed for better time management to focus on the creation of the TAPP app and TAPP native tokens. The Ethereum blockchain was chosen as it is already host to multiple crypto currencies from different developers. It is important to note that any currency that is not native to that specific blockchain are classified as tokens. When using a trusted platform like Ethereum it means there is no need for professional auditors or lawyers before issuing the tokens to the blockchain, making the process that much more efficient. Ethereum also has a set of ERC-20 standards that guides a token developer on the requirements the code will need to follow to host the tokens on the blockchain. Smart contracts are a key element to the Ethereum network and works by executing code that carries out a set of instructions [5]. These contracts are autonomous, de centralised and unmodifiable, making them a secure addition to the project. Ethereum also uses the coding language Solidity which was an unfamiliar language and became something that was a topic of interest to learn and improve knowledge on.

# Diagram Description automatically generatedProject Architecture

Figure 5.1 Architectural Diagram

# Project Plan

Project was managed using Microsoft Project. Each task was entered into the file with an estimated deadline. The greylines symbolise the baseline of when each task needed to be finished where the blue lines indicate the actual date of finish. Red lines are any task that are currently not completed.

Weekly Logs on One Note was also used to keep any working out or thoughts organised each week through the year. Team logs where stored on the One Note as well, where weekly stands were recorded with fellow classmates so that peer feedback could be given each week.

A picture containing application

Description automatically generated

# Software Features

TAPP is a wireless software-based project that simulates a transaction event between the customer, the student, and the merchant, ATU supported business, using native TAPP tokens which are stored locally. The tokens are available to purchase on a test environment website, using local accounts to make the purchase.

## 7.1 Front End Software

React native is a JavaScript framework used to develop applications for Android and IOS. The framework hosts the TAPP mobile application which is coded using JavaScript for functionality and the React Native StyleSheet component to design the layout of the app. There are also variety of other React Native libraries and components have been used to create a transaction process between the two versions of the apps that will be further discussed.

React Native was a preferred framework to use as it offers cross-platform mobile development, an important feature since according to Finances Online there are approximately 3 billion active Android smartphones and 900 million active iPhones worldwide [6]. This indicates the need for a mobile application that works both on Android and IOS. The framework also includes a live reload feature that allows the developer to make changes in real time whilst being able to view the app. This made it much more fluent in fixing errors within the app or adjusting the layout when needed. There was also the support that came from using React Native that made working with the framework more desirable. React Native has lengthy documentation and multiple resources online which made it easier to research how to integrate certain features or fix errors.

Expo is another framework that is used to build React Native apps. It allows the TAPP application to be hosted on the Expo CLI platform and accessed through the expo app so that the app can be run and tested for further development. Expo also provides various tools and libraries that were used to create certain features within the app. To view the TAPP mobile app for development, the terminal command, **npx expo start –tunnel**, which creates a tunnel via ngrok. This allows the expo app to run the code even if the computer and the phone being used have different internet connections. This command, set up by lecturer Brian O’Shea during Mobile App Development, made using expo possible on the college computers whilst running the app on the phones. A QR Code is generated which can be scanned using the downloaded expo app. From there the expo cli displays the TAPP mobile app which is then ready for development.

### 7.1.1 Navigation

Navigation inside the TAPP mobile app allows the user to navigate to different pages of the application when prompt using the native stack navigator library. Using the terminal command, **npm install @react-navigation/native-stack**, the react native stack navigator is downloaded into the project and the stack navigator can be accessed. The stack navigator will navigate the user to the appropriate screen called within the code, and was set up for both the customer and merchant versions of the TAPP mobile app.

The **createNativeStackNavigator()** function returns an object containing two React native components, Screen and Navigator. It allows the app to transition between screens, placing each screen that has been declared on top of a stack so that it can be navigated to.

Text

Description automatically generated

Figure 7.1.1.1 Stack Navigator

Text, chat or text message

Description automatically generatedThis code shows the **NavigationContainer** which is a component that manages the navigation state. The navigation state stores the navigation structure and history of the app [7]. The code can pass routes using **Stack.Screen** tags shown in below in Figure 7.1.1.2, i.e., Home is now a component that can be navigated to.

Figure 7.1.1.2 Stack Screen

Figure 7.1.1.3 shows an example of the Home page a.k.a the Home component, where if the user was to press the button it would call the **onPress** function, navigating to a page called Amount that was established using a Stack.Screen.

Text

Description automatically generated

Figure 7.1.1.3 Home Component Navigation Example

### 7.1.2 QR Code & QR Scanner

Text

Description automatically generatedA QR Code component is imported from the **react-native-qrcode-svg** library. This library generates a QR code which passes the value of price from the merchant to the customer. The price is determined by the merchant using the **textInput** component so that they can enter the price using a numeric keyboard on their phone.

Figure 7.1.2.1 Text Input Component

Graphical user interface, text

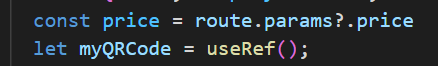
Description automatically generatedThe **dismissKeyboard()** function shown in Figure 7.1.2.2, is called during the **onPress** event shown in Figure 7.1.2.1. This function allows the user to tap near the textInput container dismissing the keyboard from view. Figure 7.1.2.2 Dismiss Keyboard Function

The price is saved by setting the state from an empty array to the value entered by the merchant using the **useState** hook. An useState hook allows such a change in variable state to occur [8], so that when another function or component, like the textInput from the merchant, wants to change the state of price it can do so.

Figure 7.1.2.3 useState Hook

The value of price is then passed from the Amount component page using the navigation so that the value can be accessed by the Transaction component.

Figure 7.1.2.4 Passing variable Price



Qr code

Description automatically generatedFigure 7.1.2.5 Route.params & useRef

The QR Code component is called to display the QR Code setting its value to the price sent through by the merchant on the Amount page and it converted to a String using the **toString()** method. If the price is not set to a value, the QR Code value will be set to NA. There are multiple style variables used to change the appearance of the QR Code. In Figure 7.1.2.6, the size is set to 250 density-independent pixels and the colour is set to use “midnightblue” with a “white” background to create a QR Code that matched the blue tones of the TAPP app. The **getRef** prop is called and receives the QR Code component saving it to the useRef hook in Figure 7.1.2.5 above. This allows the QR Code to be updated if need, meaning that the price can be changed, updating inside the QR Code. This code was implemented through research found on Blog site, Dev by Dallington Asingwire [9]. Figure 7.1.2.6 QR Code Component

Text

Description automatically generated

Figure 7.1.2.7 QR Code Component Code

Once the QR Code value is set, the customer can use their TAPP app to scan the QR Code for the price of the transaction which is sent to the webserver to be processed. For the QR code to be scanned the app uses a barcode scanner library from expo, **expo-barcode-scanner.** It is imported into the Transaction page on the customer version of the TAPP using the code in Figure 7.1.2.7.

Figure 7.1.2.8 Bar Code Scanner

For the barcode scanner to access the mobile devices camera permission needs to be granted. The value of hasPermissions is saved the same as price in Figure 7.1.2.3 using the useState hook. Currently the value is set to null and will change to true once permission has been granted to access the camera.

Figure 7.1.2.9 useState Hook Permission

Graphical user interface, text

Description automatically generatedUsing the **useEffect** hook, the code inside the customer’s TAPP app can gain permission to use the device’s camera so that the QR code can be scanned. The useEffect hook lets the framework know that permission needs to be given to continue with the code [10]. If the permission has been given the state is updated to return true allowing the camera to be accessed. If permission is not given the code will not run as the async function waits until permission is given before continuing. The expo documentation was used to implement the bar code scanner within the code [14].

Figure 7.1.2.10 useEffect Function

The **handleBarCodeScanned()** function in Figure 7.1.2.11, stores the value of the merchant’s QR code into the data. If there is no value detected from the QR code, it logs an error message. If the price has been stored inside data, **setScanned** in Figure 7.1.2.10, which is initially set to false, the state will change to true thus the customer’s bar scanner has detected the price from the merchant’s TAPP app. This will send the price to the webserver to process the transaction.

Figure 7.1.2.11 useState Hook Scanned

A screenshot of a computer

Description automatically generated with medium confidenceFigure 7.1.2.12 Handlebar Code Scanned Function

Text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceThe code below in Figure 7.1.2.12 shows if statements where the screen displays the request for camera permissions until the function above in Figure 7.1.2.11 has set hasPermission to true thus accessing the mobile device’s camera. If hasPermission is set to false, meaning that the permission for the camera has been denied, then the screen will display a message stating there is no access to the camera.

Figure 7.1.2.12 If Statement for Permission

Figure 7.1.2.13 QR Scanner Screen

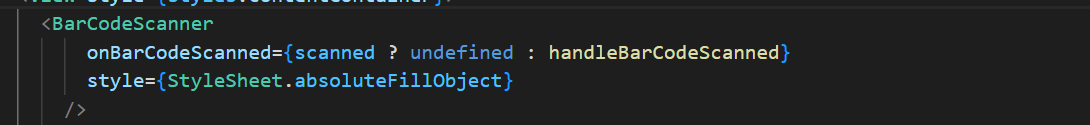
Lastly, the BarCodeScanner component is called to be displayed on the screen, waiting for scan to be set to true and for permission to be granted.

Figure 7.1.2.14 Handlebar Code Scanned Called

### 7.1.3 Post Functions

The **submitData ()** functions are used through the React Native code to send information to the server and retrieve data from the database for the merchant and the customer. For the sign-up component, a submitData () function is used as an async function. **Async** is used because the function is fetching data from a server while allowing the rest of the code on the sign-up page to run. This is done while the function is waiting for a response from the server. In the sign-up functions for both merchant and customer it uses a post method sending the companyName/username and pin/password to the server to be saved to the database as a new account . These submit data functions contain code gathered from code used in the Mobile App Development project and implemented into the TAPP project.

To establish a connection to the server, a URL using the ngrok tunnel is established on the page. The ngrok tunnel allows the server and the phone to connect if they are on two different networks to connect via a tunnel.

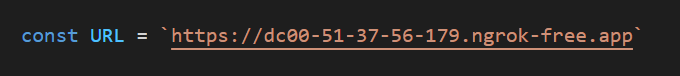
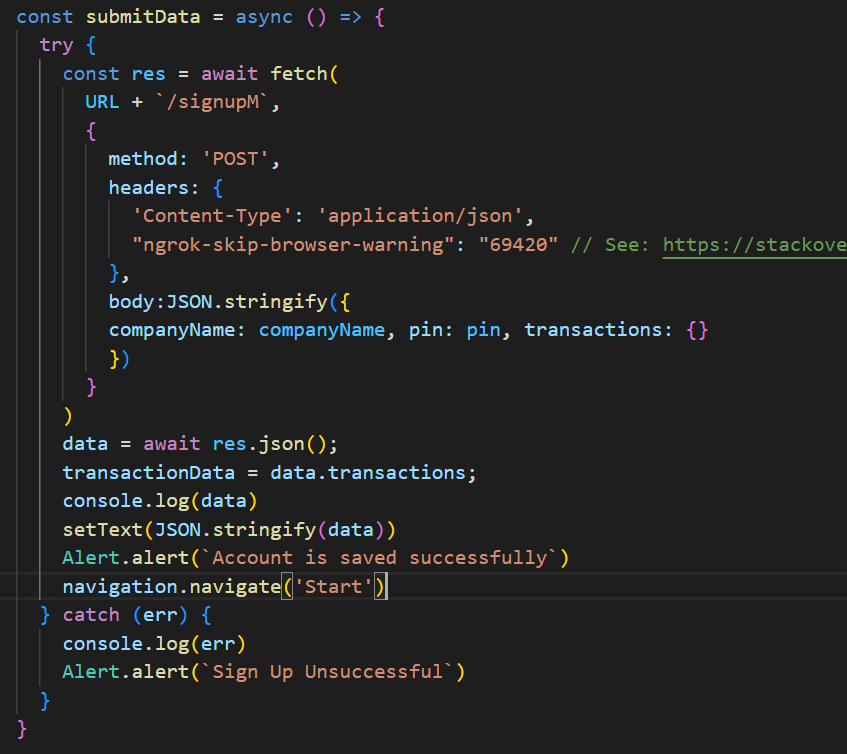


Figure 7.1.3.1 Ngrok URL

The function then waits for a response back from the server to determine if sign up was successful which is indicated by an alert using the alert component from React Native.

Figure 7.1.3.2 Alert Component

If the sign up was successful, the data variable is initialised with the merchant or customer object from the database and the user is navigated back to the start page.



Graphical user interface

Description automatically generatedFigure 7.1.3.3 Signup Submit Data Merchant

The same submitData function is used for the customer TAPP mobile app to send through the properties to the customer schema : username and password.

Text

Description automatically generated

Figure 7.1.3.4 Signup Submit Customer

To be able to enter their desired credentials, the textInput component is used to set the state of the variables so that the user can sign up, sending the properties to the server.

Text

Description automatically generated

Figure 7.1.3.5 Text Input Example Merchant

Once the user has entered their details, they press the confirm button where the onPress() function calls the submitData function shown in figures 7.1.3.3 and 7.1.3.4. The information is sent to the database and an account is made.

Text

Description automatically generated

Figure 7.1.3.6 On Press function

Another submitData() function is used for both the merchant and customer login. The companyName and pin is requested by the merchant so that they can access their account on the TAPP app. Once they have entered the details into the app using the textInput, the onPress function used inside the button component, calls the submitData() function.

A screenshot of a phone

Description automatically generated with medium confidenceText

Description automatically generated This sends the companyName and pin to be authenticated by the server shown below in Figure 7.1.3.6. If the response sent back is ok, the balance, transactions and companyName properties are stored in the app and are passed from the login page to the home page using the navigation component. This is done so that the home page can access and display the merchants current balance and transaction history.

Figure 7.1.3.7 Login Submit Data Function Merchant & Login Customer Screen

Like the merchant, the same procedure occurs for the customer. The username and password properties are sent to the server and if they match the properties in the customer account inside the database, the customer can login.

Text

Description automatically generated

Figure 7.1.3.8 Login Submit Data Function Customer

Since the properties are passed from login to home, the value of the parameters can be accessed on the home page.



Figure 7.1.3.9 Route Params Home Component

This occurs so that the balance and the transactions shown here in Figure 7.1.3.9 and Figure 7.1.3.10 can be displayed on the home page.

Text

Description automatically generated

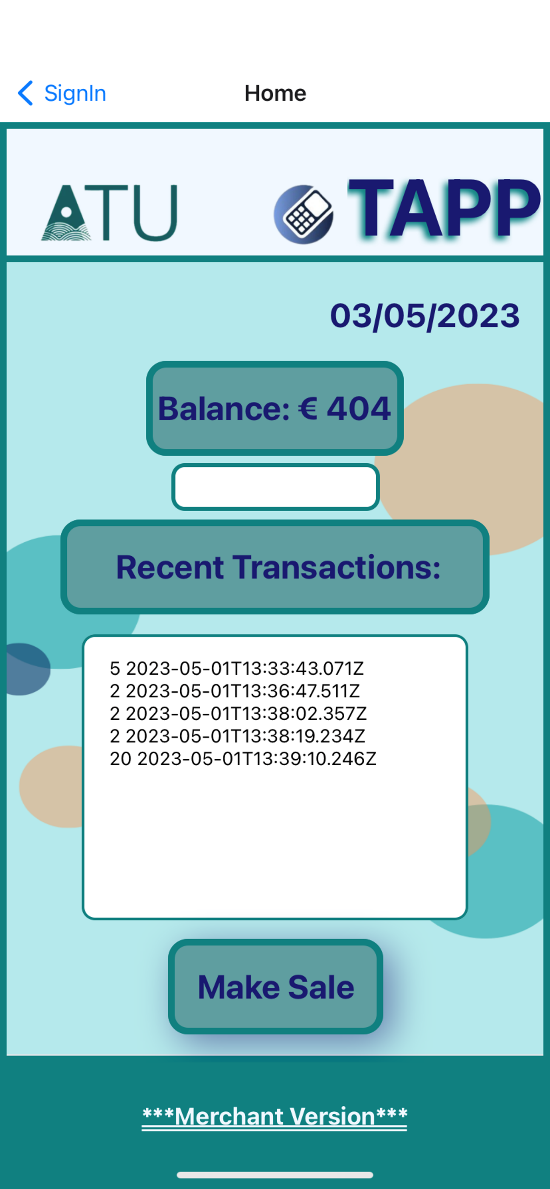
Figure 7.1.3.10 Balance Display

The transaction data is shown using a map so it can create an array storing the data of the transactions inside. The **.length** function is used so that if there is nothing inside the transaction array, for example, if it is a new user signing in the app, “No transactions” will be displayed. The slice function() is used to display only the last five transactions in the users account and the **Math.max()** is used to avoid any negative indexes in case the user has less than 5 transactions in their account.

Text

Description automatically generated

Figure 7.1.3.11 Transaction Display

The exact method and code are used for the Customer TAPP mobile app version.

For the transaction to occur between the merchant TAPP app and the customer TAPP app, submitData() functions are used for both users.

For the merchant, the price is carried through from the amount page to the transaction page Figure 7.1.2.4. Once the QR code is scanned and the merchant has pressed the OK button, the submitData() function is called. This function shown below in Figure 7.1.3.11, sends the companyName to the server so that it can find the merchants account by the companyName. The submitData function also posts the price to the server so that the price of the new transaction can be added by the server to the database. The function waits for a response from the server and if the function is successful a new balance is calculated, adding the price to the current balanceData variable which stores the merchants pre-existing balance from the database. This information is then passed to home via the navigation component so that the new updated balance and transaction list can be displayed.

Figure 7.1.3.12 Home Display

Text

Description automatically generated

Figure 7.1.3.12 Transaction Submit Data Function Merchant

The customer has a similar procedure where inside the handleBarCodeScanned() function shown above in Figure 7.1.2.11. Once the QR code is scanned by the customer, the value of the QR code is stored in the variable price. The username is sent to the server to find the customer by the username so that it can update their account. Once a response is received and the customer account on the database is updated. The new balance is calculated by subtracting the price from the customer’s current balance. This is then passed to home to update the customers balance and transaction list via the navigation component if the top up was successful.

Text

Description automatically generated

Figure 7.1.3.13 Transaction Submit Data Function Customer

In another post function used by the customer, a top up value from the textInput in the React Native app is sent to the server. The customer’s account is located using the username and the number of the top up amount is passed to the server. If a success message returns true, the balance was successfully updated in the database by the server. The balance is calculated in the front end by adding the number to the current balance of the customer. Again, this triggers the navigation component to pass the updated balance to home so that it can be displayed for the customer.

Text

Description automatically generated

Figure 7.1.3.13 Top Up Submit Data Function Customer

## 7.2 Back End Software

Node.js is a back-end JavaScript server environment that executes code. TAPP mobile application uses node.js to pass information between both merchant and customer versions of the TAPP mobile app. This allows for the transaction to occur through a variety of post API methods that send the information to the database or in between the merchant and customer TAPP mobile apps.

### 7.2.1 Database

Mongo DB is a database that stores JSON-like documents [11] using a schema to define its properties that are set up in the TAPP server code.

A connection from Mongo DB to the node.js server is established via the **Mongoose library**. Mongoose is an Object Data Modelling (ODM) library which provides the schema and translates between the objects in the code and those in the database [12].

For the **strictQuery** to pass so that Mongoose does not throw an error, any properties going into the database need to be correctly defined inside the schema, an example is below in Figure 7.2.1.1. Mongoose connects to the database on port 27123 and the port for the localhost is connected on 3000.

Graphical user interface, text

Description automatically generated

Figure 7.2.1.1 Mongoose Set Up

For the TAPP project, three schemas have been defined inside the node.js server. The first schema is the one for the transactions between the merchant and the customer. The properties defined are the price and the date. Using the **Date.now** function the server can store current date and time of the transaction when it entered the database.

Text

Description automatically generated

Figure 7.2.1.2 Transaction Schema

Both the merchant and the customer have their own schemas that will store their perspective properties into the database. For the merchant, that is the companyName, pin and the balance of their account which is stored inside the merchant collection. Whereas for the customer, that is the username, password, and balance of their account which is stored in the customer collection. Both merchant and customer schemas contain a property called transactions which is an array of the transactionSchema shown in Figure 7.2.1.2. This allows the transactions to be saved under the merchant and customer so that is it specific to the users account.

Text

Description automatically generated

Figure 7.2.1.3 Merchant and Customer Schema

Regarding the database for the collection set up, the schemas are established and wrapped in a Mongoose model which allows Mongoose to translate data and objects between the Mongo DB database and the node.js webserver. The implementation for the database setup was taken from code created for the Mobile App Development project.

Text

Description automatically generated

Figure 7.2.1.4 Database Collections

### 7.2.2 Authentication

Due to the TAPP mobile app being a financial payment app, authentication and security are important features to include into the project.

Both the merchant and the customer need to login to gain access to their account. This is achieved by the user logging in using the prompts in the React Native app. The server authenticates if the credentials match what is stored inside the merchant and customer collections inside the database to determine if the user can be authorised to login to the account.

**Bcrypt** is a package that has been included inside the node.js server and is used to hash the pin of the merchant and the password of the customer during sign up before saving it under their account in the database, already encrypted. This encryption process prevents a hacker from accessing user pins/passwords from the database.

The **.env** file contains environment variables that are loaded into the runtime using the **dotenv** package which is configured in Figure 7.2.2.1. **The EXTRA\_BCRYPT\_STRING** contains a string that is used as a salt. A salt is used to add random data to the input of the hash function so that the output will be unique, preventing the password from different attack vectors [13].

Text

Description automatically generated

Figure 7.2.2.1 .Env File

The bcrypt package is listed below in Figure 7.2.2.2 so that the server has access to the hashing function.

Text

Description automatically generated

Figure 7.2.2.1 Bcrypt Package

For the user to login and have their account authenticated they first need to sign up. The sign-up API shown below in Figure 7.2.2.3, registers the merchant and adds their account to the merchant collection in the database. The companyName and pin are sent from the React Native app to the server so that an account can be made. The pin is encrypted using the bcrypt function and the properties inside the merchant schema are stored to the user account in the database. The balance is set to 0 as the user is only creating an account and their transaction history is empty. It is important to note that the pin is stored in the database hashed to avoid the account being hacked.

Text

Description automatically generated

Figure 7.2.2.2 Signup API Merchant

During the login authentication, the companyName and pin are sent up from the app to the server where the code will attempt to find the companyName of the merchant inside the database. If it is not present, the server returns a 401-error stating that the account has not been found. If the companyName is present inside the merchant collection in the database, the checkPass variable checks the pin sent from the merchant during login and compares it with the hashed password. This is achieved by the EXTRA\_BCRYPTY\_STRING generating another salted version of the pin which is then compared to the already hashed pin stored in the database.

Text

Description automatically generated

Figure 7.2.2.3 Sign in API Merchant

A similar process is done for the customer TAPP mobile app shown below in Figure 7.2.2.4. The customer can sign up entering a username and password. The password is encrypted using the Bcrypt hash function and their account is saved to the database.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7.2.2.4 Signup API customer

Much like the merchant, the customer’s credentials are checked during login. The customer’s username is checked to see if that username is present inside the database. If that is the case, the code creates a salt which compares the password the user has sent into the password encrypted in the database. If the checkPass function is true meaning the credentials match, then the customer gains authorisation to login to their account. Bcrypt was taught to the class by lecturer Brian O’Shea during Cloud Development.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7.2.2.5 Sign in API customer

### 7.2.3 API Endpoints

The API endpoint functions are used within the node.js server to submit and receive data from the front-end to the database and vice versa.

The post API in Figure 7.2.3.1 is used to allow the customer to top up their balance if needed. This function receives a number sent by the customer via the React Native app which indicates the amount they want the balance to be topped up with. The mongoose function **findOne()** looks for the user with that specific username and once the username is in the database, a newBalance variable is initialised with the balance already stored in the customer’s account plus the number that has been send through via the top up page on the TAPP app. The **parseInt()** is a JavaScript function that turns the string, number, to an int value so that the balance can be summed. Once the new balance is calculated it is set to the balance property stored in the customer schema and the **save()** function saves the updated customer account to the database. The updated customer properties are sent back to the app so that the customer can view the balance on the home page. If an error occurs such as a network connection error or the username has not been in the database, the console will log the failed attempt and the success is set to false which is sent down to the TAPP app to notify the customer that the top up was unsuccessful.

A screenshot of a computer

Description automatically generated

Figure 7.2.3.1 Top Up API Customer

Both the merchant and customer TAPP mobile apps have an API function that will add a new transaction to their accounts within the database. For the customer the price is sent from the TAPP app to the server and using the mongoose findOne() function, the code looks through the database for that username. If the customer’s username is not found, then it will send back an error to notify the customer. If the customers username is located inside the database a new transaction is made, storing the price of the transaction which was sent up after the customer scanned the QR code. The **push()** function adds the transaction to the transaction array inside the customer’s object inside the database. Because this transaction represents the customer paying for a product or service from ATU, the price of the transaction needs to be taken from the customer’s balance. This is down by updating the balance property to the newBalance which is calculated by subtracting the price from the customers pre-existing balance. Once that is completed the customers updated account properties are saved and a success: true along with the updated properties are sent down to the TAPP app so that the transaction history can be displayed on the customers home page.

Text

Description automatically generated

Figure 7.2.3.2 Add Transaction API Customer

A similar process happens for the merchant when a transaction occurs. Once the QR code is scanned by the customer and the merchant has pressed the OK button in the transaction page. The price is sent up to the server along with the companyName. The code will use the findOne() function to locate the merchant by their companyName and if not detected, an error is logged. If the merchant, is found a new transaction is created for the specific merchant account much like the customer with the same price of the transaction. The balance for the merchant is added onto the merchant’s current balance to simulate the customer giving the merchant the money via the transaction procedure and once again like the customer, the merchants object in the database is updated with the new balance and transaction object. The updated information is sent back down so that the merchant can also view their new balance and transaction history.

Text

Description automatically generated

Figure 7.2.3.3 Add Transaction API Merchant

## 7.3 Native Crypto Tokens

Native TAPP tokens were created as part of the TAPP project to process a transaction between a student, the customer, and an ATU run business, the merchant. The tutorial by Dapp University [14] was a successful learning tool in understanding the process of creating the TAPP tokens. The tokens were made using solidity, a coding language, required to create tokens on the Ethereum blockchain established by the ERC-20 standards.

To be able to test code, accounts were set up using **Ganache**, a local blockchain with test accounts that allows for testing the TAPP token smart contracts. **Truffle**, a framework for Ethereum was also used deploys the TAPP token functions using the ganache accounts to test that the code locally.

Table

Description automatically generated

Figure 7.3.1 Ganache Test Accounts

### 7.3.1 Migrations

Migrations allow smart contracts to be deployed to the Ethereum blockchain by determining order of deployment as well as setting prerequisites that required for the contract to run. Inside the **\_deploy\_contract.js** file, the TappToken and TappTokenSale contracts are deployed. The code initialises the total supply to 1,000,000 tokens, code sets the price of each token to 0.01 ether(1000000000000000 Wei) and updates the blockchain when during deployment.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7.3.1.1 Deploy Contracts File

### 7.3.2 Contracts

Smart contracts are programs stored on a blockchain that enforce the terms declared in the contract. Once deployed onto the blockchain they can not be altered and are decentralised meaning they not controlled by a centralised entity.

In the TappToken.sol contract, the TAPP token has been set up with a name, a symbol, and the standard. The total supply of TAPP tokens has been set to the totalSupply which is passed through by the initial supply declared in the deploy\_contract.js migration file shown above in Figure 7.3.1.1. The total balance is sent to **msg.sender** upon initialisation of the contract. This function contains the address of the account, account 0 in figure 7.3.1, that to tokens will be sent to.

Text

Description automatically generated

Figure 7.3.2.1 Token Set Up

Text

Description automatically generated

Figure 7.3.2.2 Token Supply

The transfer function is used to transfer TAPP tokens from one account to another. The **require** statement is used so that for the transfer to occur, the account must have enough funds to make the transfer otherwise an exception will be triggered. If the requirement is met the value of the sender account will decrease by the value whereas the value of the **\_to** account (the address of the receiving account) will increase. If the transfer was successful a true value is returned.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7.3.2.3 Transfer Function

The TappTokenSale.sol contract allows the TAPP tokens to be sold. The parameters are set for the address of the token contract that is used to purchase the tokens during the sale as well as the token price initialised in ETH using the deploy migration file in Figure 7.3.1.1.

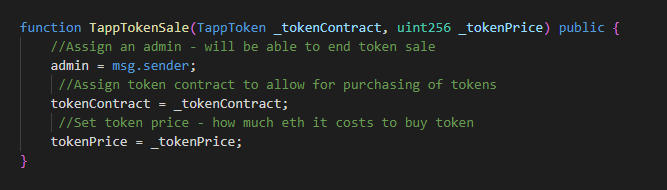


Figure 7.3.2.1 Token Sale Set UP

The multiply function is used to multiply two integers and return a result. A require statement is used to check that the result is the product of the two integers otherwise an exception is thrown. This function is used below in Figure 7.3.2.3 to calculate the total price of the number of tokens that want to be bought.

Text

Description automatically generated

Figure 7.3.2.2 Multiply Function

The buyTokens function allows an account to purchase a certain number of tokens from the sale. The require code states that the value needs to be the tokenPrice multiplied by the number of tokens that are being purchased. The next require statement ensures that there are enough tokens in the contract to meet the sale order. The last require statement ensures that the tokens are transferred to the buyers address otherwise an exception is thrown. The function then keeps track of the number of tokens sold by adding the number of tokens sold to the tokenSold variable every time a sale has gone through. Lastly the Sell event is triggered to notify that a sale has occurred.

Text

Description automatically generated

Figure 7.3.2.3 Buy Tokens Function

To end the sale an **endSale()** function was established so that admin can end the sale if needed. The require statement checks that the msg.sender is the admin account while the next require statement transfers any remaining tokens back to the admin account.

Text

Description automatically generated

Figure 7.3.2.4 End Sale Function

### 7.3.3 Testing

Testing was used to ensure that the functions within the smart contracts worked as they should. The **TappToken.js** file was used to test the initialisation of the contract, the initial supply, and the transfer of token ownership.

The first test is used to check that the TappToken contract is initialised with the correct values. An instance containing the characteristics of the Tapp contracts is created so that the methods inside the contracts can be executed. This test checks that the name, symbol, and the standard match what is established in the contract. This is accomplished by using the **assert.equal()** method which checks that the value returned matches the value expected.

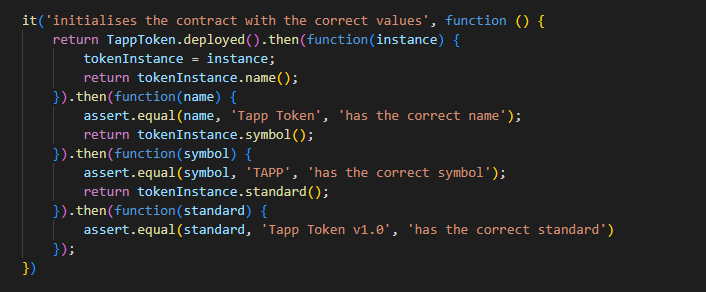


Figure 7.3.3.1 Tapp Token Initialisation Test 1

The test in below Figure 7.3.3.2 checks that the initial supply is correctly set and allocated to the admin account, account [0] in the ganache environment.

Text

Description automatically generated

Figure 7.3.3.2 Tapp Token Initialisation Test 2

The test for the transfer function shown in Figure 7.3.2.3, checks that the function is working as it should. The function should transfer a set number of tokens from one account to another along with updating the balances of the accounts after the transfer is completed. The test checks through various arguments shown below in Figure 7.3.3.3. First, the test checks that if the number of tokens being transferred exceeds the balance of the sender an error will occur. The test then checks that if the number of tokens is under the balance of the sender, they can be sent from sender, account [0] to account [1]. It will then check that the balance of the accounts has been updated accordingly based on the transfer of TAPP tokens.

Text

Description automatically generated

Figure 7.3.3.3 Transfer Function Test

The TappTokenSale.sol contract is also being tested using the TappTokenSale.js testing file. First the contract is checked that the address of the contract as well as the token price are returning the value that is expected, shown below in Figure 7.3.3.4.

Text

Description automatically generated

Figure 7.3.3.4 Tapp Token Sale Initialisation Test

The next test checks the buyTokens function within the TappTokenSale contract. This is done by first checking that account and the number of tokens purchased have been logged as well as the tokenSold variable shown above in Figure 7.3.2.3 has been incremented by the number of tokens sold. The test also attemps to buy tokens with insufficient funds checking as well as trying to buy more tokens than what is available. The tests fail thus throwing an erroring and not allowing a sale to occur.

Text

Description automatically generated

Figure 7.3.3.5 Tapp Token Sale Buy Tokens Test

The last test checks that the sale is only ended by the admin account. It does this by trying to end the sale using another account which throws an error. When ending the sale as the admin, the test then expects the unpurchased tokens from the sale to return to the admin account.

Text

Description automatically generated

Figure 7.3.3.5 Tapp Token Sale End Sale Test

### 7.3.4 Sale Deployment

To simulate a sale so that an account can purchase TAPP tokens, a server code has been written to create a website to buy tokens. The ganache accounts are connected to the website via the google chrome extension MetaMask which allows the ganache accounts to buy tokens and updates their account balance in ETH.

A HTML file taken from the GitHub of TUTORAL for the display of the website which is shown below in Figure 7.3.4.2. Another three files from the GitHub where also added to the project folder: bootstrap, truffle-contract and web3. Bootstrap is a framework that provides a collection of CSS and JavaScript components that were used in the layout of the website. The truffle-contract.min.js is a library that interacts with the smart contracts on the blockchain. Lastly, web3 which was only briefly touched in this project allows interaction with the Ethereum network.

# Ethics

TAPP mobile app allows for a completely wireless and hardware less transaction between the students of ATU and ATU run businesses. Society has already moved into an era where online banking dominates, and people want a quick and easy transaction when they go to purchase goods and services. The banks still control the monopoly of transactions that people make on the daily, even the merchants need to purchase hardware to process the transactions. Due to this being the primary way that businesses and people make transactions, banks charge fees on people’s accounts as well as the merchants on every transaction made.

Crypto and web3 may seem like a scary concept, but it could be the answer to giving people back more control of how their money is processed and where they want to store it. Crypto offers a safe way to make transactions between accounts completely de-centralised from a governing authority.

Right now, crypto is unpredictable. To purchase a crypto currency and determine if the value will increase or decrease is uncertain today. This plays a big risk on people wanting to put their hard-earned money into a crypto currency and use it to make purchases.

Hopefully, as society progresses crypto will be able to stabilise its value and become a safer option for people thus taking away the control that certain governing authorities have over people and their money.

# Conclusion

Transact App (TAPP) project is a react native mobile application that allows students, the customers, to purchase goods and services from ATU vendors, the merchants. This is done with a native TAPP token that can be transferred from one account to the other locally.

The customer and merchants accounts are saved and stored on a mongo DB database which uses Mongoose to connect it with the node.js server. This allows login authentication and well as encryption the password once the user wants to sign up.

TAPP token contracts have been created to allow for transfer between ganache accounts as well as a sale contract which allows a user to purchase a certain of Tapp tokens.

Further developments would be to integrate the token transfer functions from the token contracts into the TAPP mobile app so that tokens are being transferred from student to vendor rather than information.

In conclusion, TAPP is a secure transaction app that allows instantaneous transfer of funds from the student, the customer, to the ATU vendor, the merchant. It is completely wireless without hardware thus making the transaction much efficient and environmentally friendly than traditional banking methods.

# References

[1] A.Lielacher (2022, Sep. 20) *How Many People Own & Use Bitcoin in 2022* [online].

Available: [How Many People Own & Use Bitcoin in 2022? (cryptonews.com)](https://cryptonews.com/exclusives/how-many-people-own-use-bitcoin-2022.htm#:~:text=Interestingly%2C%20the%20global%20crypto%20exchange,holding%20bitcoin%20surpassed%2040%20million.)

[2] *The basics about cryptocurrency* (no date) *The Basics about Cryptocurrency | CTS*. OSWEGO, State University of New York. Available at: https://www.oswego.edu/cts/basics-about-cryptocurrency (Accessed: April 27, 2023).

[3] Hayes, A. (2023) *Blockchain facts: What is it, how it works, and how it can be used*, *Investopedia*. Investopedia. Available at: https://www.investopedia.com/terms/b/blockchain.asp (Accessed: April 27, 2023).

[4] Garnett, A.G. (2023) *How to make a cryptocurrency*, *Investopedia*. Investopedia. Available at: https://www.investopedia.com/how-to-make-a-cryptocurrency-5215343 (Accessed: April 29, 2023).

[5] Cryptopedia Staff (2022) *What is a crypto smart contract? how they work*, *Gemini*. Available at: https://www.gemini.com/cryptopedia/crypto-smart-contracts-explained (Accessed: May 2, 2023).

[6] Jay, A. (2023) *Number of smartphone and mobile phone users worldwide in 2022/2023: Demographics, statistics, predictions*, *Financesonline.com*. FinancesOnline.com. Available at: https://financesonline.com/number-of-smartphone-users-worldwide/#:~:text=Smartphone%20Operating%20System%20Preference%3A%20Android%20vs%20iOS,-Android%20democratized%20smartphone&text=Various%20sources%20put%20the%20total,27.47%25%20of%20the%20market%20pie. (Accessed: April 29, 2023).

[7] *React navigation* (no date) *React Navigation RSS*. Available at: https://reactnavigation.org/docs/navigation-state/ (Accessed: April 29, 2023).

[8] Herrera, E. (2023) *Usestate in react: A complete guide*, *LogRocket Blog*. Available at: https://blog.logrocket.com/guide-usestate-react/#:~:text=start%20with%20use%20.-,What%20is%20the%20useState%20Hook%3F,the%20setter%20function%20is%20called. (Accessed: April 29, 2023).

[9] Asingwire, D. (2022) *How to generate and download QR code in a react native application*, *DEV Community*. DEV Community. Available at: https://dev.to/dallington256/how-to-generate-and-download-qr-code-in-a-react-native-application-1k8e (Accessed: November 2023).

[10] *Using the effect hook* (no date) *React*. Available at: https://legacy.reactjs.org/docs/hooks-effect.html#:~:text=What%20does%20useEffect%20do%3F,after%20performing%20the%20DOM%20updates. (Accessed: 30 April 2023).

[11] *What is mongodb?* (no date) *MongoDB*. Available at: https://www.mongodb.com/what-is-mongodb (Accessed: 30 April 2023).

[12] Karnik, N. (2023) *Introduction to mongoose for mongodb*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ (Accessed: 30 April 2023).

[13] Arias, D. (2021) *Adding salt to hashing: A better way to store passwords*, *Auth0*. Available at: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/ (Accessed: 30 April 2023).

[14] McCubbin, G. (no date) *Code your own cryptocurrency on Ethereum (how to build an ERC-20 token and crowd sale website)*, *Dapp University*. Available at: https://www.dappuniversity.com/articles/code-your-own-cryptocurrency-on-ethereum (Accessed: 01 April 2023).