M-Commerce Offline Payment

Girish Jeswani
181IT116
Dept. of Information Technology
NITK Surathkal

Yash Kumar Gupta
181IT153
Dept. of Information Technology
NITK Surathkal

Olan Pinto
181IT231
Dept. of Information Technology
NITK Surathkal

***Abstract*—During the ongoing Covid-19 Pandemic we have seen a surge in online transactions. In India there are several regions that lack stable internet connection, thereby we are unable to reach an entirely Digitized India. To cope up with this problem we propose our model wherein we bring in the concept of offline Customer-Merchant payments. The current wallet payment architecture requires online connectivity during transactions. However, it is not always possible to have online connectivity. We identify three main scenarios where fully offline transaction capability is considered to be beneficial for both merchants and consumers. Scenarios include making purchases in locations without online connectivity, when a reliable connection is not guaranteed, and when it is cheaper to carry out offline transactions due to higher communication/payment processing costs involved in online approvals. The aim of the project is to address the challenge of providing secure offline transaction capability when there is no online connectivity keeping in mind to not compromise any security features in our M-commerce offline wallet payment application**

***Index Terms*—M-Commerce, Web Sockets, Threats, Security, Document Level Locking, Async Storage, Information Assurance, Hashing, Intrusion Attack.**

I. Introduction

Mobile payment is the root of financial transactions for most services or goods between trading parties. Majority of the payment system currently in use considers online communication with the network and is very infrastructure dependent, which is very different from wireless mobile adhoc systems. Payment failure is not an uncommon phenomenon these days, may it be a credit or debit card, insufficient balance or just lack of internet connectivity. Unable to establish connection with the payments platform is also a major reason for online payment failure. However with the proposed application, end users will be able to make offline payments using their mobile devices(wallet). The RBI proposed a pilot scheme for small value payments in offline mode with built in features for safeguarding the interest of the users, liability protection, etc. Under the pilot scheme, banks and non banks (Payment System Operators) will be able to provide offline payment solutions using cards, wallets or mobile devices for remote or proximity payments. This model will enable users in regions of unstable connectivity, devoid of internet, to successfully onboard in this venture to eliminate hard Cash Transactions.

II. Literature Survey

Paper [1] addresses the challenge of providing secure offline transaction capability when there is no online connectivity on either the mobile or the terminal. It discusses how tokenisation has been adopted by the payment industry as a method to prevent Personal Account Number (PAN) compromise in EMV (Europay MasterCard Visa) transactions, and how the current architecture specified in EMV tokenisation requires online connectivity during transactions.

Paper [2] proposes a new privacy-preserving offline mobile payment protocol based on NFC. This scheme is well designed to accomplish all security and privacy requirements. In this scheme, a group signature is employed to provide unlinkability of transactions.

Paper [3] highlights a novel approach for building a secure, scalable, and flexible e-payment systems in offline mode of communication for enhanced security on transactions. It discusses how the current payment systems are directly dependent on fixed infrastructure of network (cellular network), which fails to facilitate optimal level of security for the payment system. The proposed system uses Simple Public Key Infrastructure for providing the security in payment processes. The performance analysis of the proposed model shows that the system is highly robust and secure ensuring anonymity, privacy, non-repudiation offline payment system over wireless adhoc network.

III. Methodology

This paper employs a novel application that incorporates an M-commerce wallet payment system. This application was developed with a particular goal that is to enhance and improve the current wallet payment system between a customer and a merchant without compromising any security features. The application incorporates online as well as offline payment methods. To carry out this well designed M-commerce payment architecture and to prevent any security threats we used NodeJs as our Back-end and the Web-Socket libraries that it has to offer, we used MongoDB as our Database that took care of all the transactions and logins, and the front end was developed on React Native.

In the case of online payments, the first step is a user login page, that has a safe API protocol whereby the end users credentials are hashed and stored in the database.

Next, there is a homepage that takes care of the basic requirements of the user for example, balance, transaction history and the associated bank details, which are all accessible with a single tap. The transaction between two users is the most important part of the app and security must be a top priority in such payment apps. Given the already existing safety of API calls these days, we wanted to create a more secure channel/tunnel where only the concerned client and server can interact. The client first tries to establish a Web-Socket connection if possible, and if not, the app initiates an offline mode transaction.

And, for concurrency, we incorporated MongoDB(s) multi-granularity locking that allows operations to lock at the global, database or collection level, and allows for individual storage engines to implement their own concurrency control below the collection level.
Once the connection is established, any kind of communication between the client and the server will be made through this socket connection. After the connection is established, the user can make a transaction by providing the details of the merchant and the amount that has to be transferred.

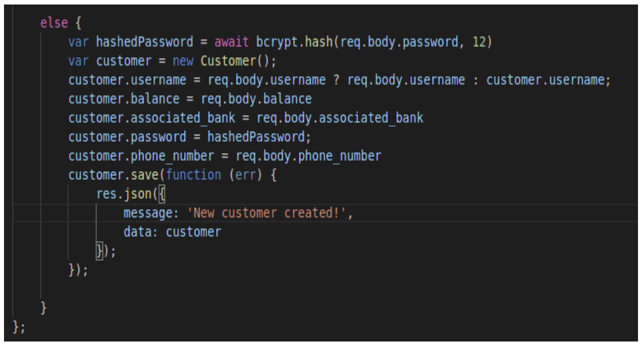
After the transaction is complete, the user is redirected to a transaction confirmation page, post which they are redirected to the homepage from the payment gateway. From the homepage the user can either continue making new transactions or can close the application.

Axios is a promise-based HTTP client that works in the browser and in a node.js environment. It provides a single API for dealing with XMLHttpRequests and node's HTTP interface. Besides that, it wraps the requests using a polyfill for ES6 new promise syntax. We can also intercept and cancel requests, and there's built-in client-side protection against cross-site request forgery.

In the situation where the customer has no internet connection, identified by react natives' NetInfo we make use of the asynchornous storage that react native has to offer to store a logged users information until the customer gets back online. AsyncStorage is React Native's API for storing data persistently over the device. It's a storage system that developers can use and save data in the form of key-value pairs which is very similar to a browsers local storage API. After the customer gets a stable internet connection, the data is picked up from the Async Storage and a secure socket connection is made from the homepage and is connected to the server and the transaction continues as it would similar to the online scenario.

IV. Work Done

The M-Commerce offline payment application opens to a login page that asks the concerned customer or merchant to enter their credentials i.e, username and password. For added security all passwords are hashed and stored.



```
    else {
        var hashedPassword = await bcrypt.hash(req.body.password, 12)
        var customer = new Customer();
        customer.username = req.body.username ? req.body.username : customer.username;
        customer.balance = req.body.balance
        customer.associated_bank = req.body.associated_bank
        customer.password = hashedPassword;
        customer.phone_number = req.body.phone_number
        customer.save(function (err) {
            res.json({
                message: 'New customer created!',
                data: customer
            });
        });

    }
};
```

Figure 1 : Creating a new customer

Figure 1 depicts how a new user is created into our Mongo DB collection. On creating a POST request via the route [/customers] the username is matched with the database and if matched, a response is sent to the client indicating that a customer with those credentials already exists, else a new customer is created and the procedure for it are as follows. First the password entered by the end user is hashed using NodeJS' bcrypt library that takes in two parameters, one, the password to be encrypted and two, the salt to be used to hash the password, and if specified as a number, a salt will be generated with the specified number of rounds. Thereafter the end users balance, associated bank and phone number are recorded into the collection.

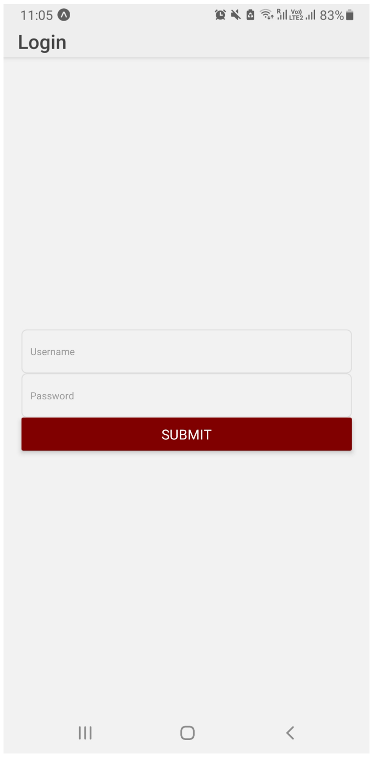


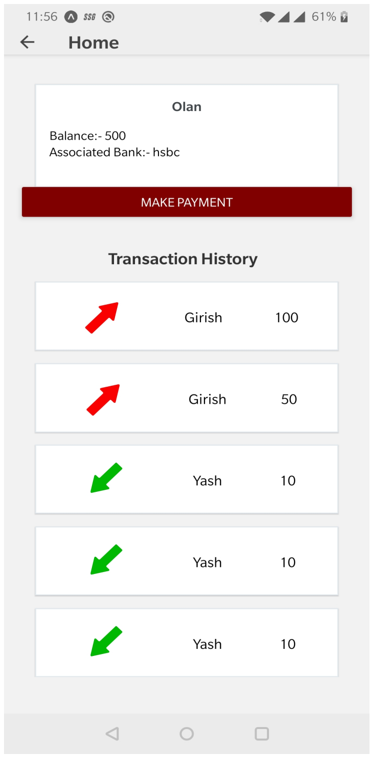
Figure 2 : User Login page

The first page of the app is the login page as seen in figure

2. Here, the user enters their credentials that have been stored in the database. The entered credentials are matched with the database and the user is allowed to login if they are correct. The login works through the routes set up through Express. The POST request [/customers/login] is created, after which the app moves on to the next page. The way the route works is it first checks if the username exists in the database and then hashes the entered password and matches it with the database. Upon successful verification, the user is allowed to use the App. Upon successful authentication the end user logs in and is redirected to the clients homepage where the client would be able to see their remaining balance, their associated bank and a history of the last five transactions made by them via this application. In the case the customer has a stable internet connection, clicking on the make payment button will take the user to the transaction page that is when a socket connection is established with the server, and once this connection is established, any other communication between the client and the server will take place through this socket. We have also coded an additional document level lock for Mongo DB, which ensures that multiple users can make a transaction at a given time without locking the whole collection. It works by locking the rows returned by a selection query, such that other transactions trying to access those rows are forced to wait for the transaction that locked the rows to finish. These other transactions are effectively put into a queue based on when they tried to read the value of the locked rows. The application takes care of concurrency by ensuring that a document/row level lock is created so that the application does not contain any write/write and read/write conflicts. Once the server side processing of the transaction is complete, a status of the transaction is sent back to the client through the same socket.

Figure 3 shows the Homepage of the App. The client details such as Username, associated bank account, and the current balance are displayed on this page. Along with such details, the transaction history of the client is displayed as well, which shows the last five transactions of the user, green depicting incoming and red depicting outgoing.

React lets us define components as classes or features. To define a component class we simply extend React.component. Each component has several “life-cycle methods” that we can override to run code at particular times in the process. We have used the component mounting life cycle method in our project. Mounting is the phase in which our React component will mount on the DOM i.e., is created and inserted into the DOM. This phase comes onto the scene after the initialization phase is completed. In this phase, our component renders for the first time. Once the user has entered valid credentials and login is successful, the App moves onto the homepage. Here, the componentWillMount() method is used which is called just before the component is mounted onto the DOM or the render method is called. Once this method is called, all the client details and the transaction history is queried from the database. React components are stacked on top of the other, for example the homepage above the login and the payment above the homepage, which makes updating these components difficult as they only get updated with ReactDOM.render. Which is why we added an additional listener (div focus) which calls the render method when we come back to our homepage, making sure that only the updated values are displayed. The .focus() method tells the app which screen is being acted on, which is helpful to find out which page the app is currently on. By exploiting that, the app can re-render when a payment is completed. Moving on to the offine mode payments, we have made use of React Async Storage which is an unencrypted, asynchronous, persistent, and key-value storage system that is global to the app which is better than Local Storage. Here, the associated bank details and the balance amount is stored, which will be modified after a transaction. We have made the use of a community library in react native called ’Netinfo’ which allows us to get information about connection type and connection quality. This helps us when the the mobile gets a stable internet connection which the app then checks for using the Netinfo library, after which, the last offline mode transaction can be updated online. This is done from our homepage itself and another socket connection is initiated through this page, which is then used to update the balance info. More about the socket connection is explained in detail in the transaction page.


11:56
61%
Home

Olan

Balance:- 500
Associated Bank:- hsbc

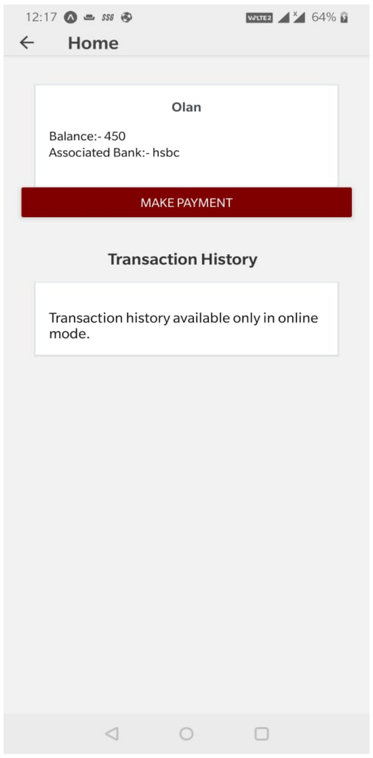
MAKE PAYMENT

Transaction History

↗	Girish	100
↗	Girish	50
↙	Yash	10
↙	Yash	10
↙	Yash	10

Figure 3 : Homepage

Figure 4 is the transaction page of our application and every transaction that takes place is happening through a secure tunnel/channel also known as socket. When a customer wants to pay the merchant, the former will enter the transaction amount and the merchants phone number and thereafter if sufficient balance is available i.e, post transaction the wallet value should be greater that zero, a successful transaction will have taken place and the customer will have redirected to the home page where their remaining balance and transaction history will be displayed. All these transactions are taken care of by NodeJS' socket.io library which enables real-time, bidirectional and event-based communication between the customer and the merchant. Web Socket is a communication protocol which provides a full-duplex and low-latency channel between the client and the server i.e customer and merchant. Using Web Sockets, a developer can come up with a function that works consistently across all the platforms. Web Socket represents a single TCP socket connection, thus eliminating the problem of connection limitation. In the case of REST architecture, the client and server can be implemented independently, without knowing each other. This client/server paradigm has lots of benefit with it i.e, the client-side code can be altered any time, without the server getting affected. The different client having REST interface can hit the endpoints at the same time and receive the same response. Also, one other feature is statelessness. A server does not need to know which state the client is in, and the same holds true for the client. All in all when it comes to security and information assurance it is necessary to ensure that the transactions or any such confidential information is passed through a secure socket only, so that the chance of an intrusion attack or a network vulnerability is minimized.


12:17 64%

Home

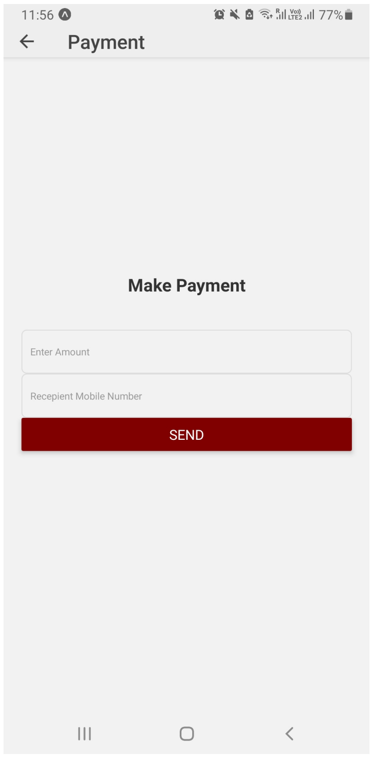
Olan

Balance:- 450
Associated Bank:- hsbc

Transaction History

Transaction history available only in online mode.

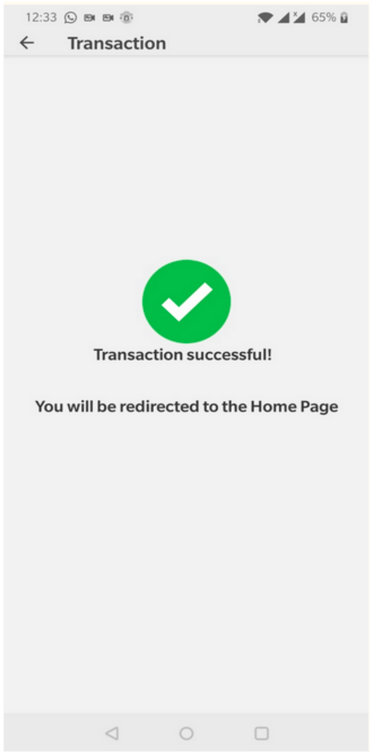
Figure 5 : Offline Homepage


11:56 77%

Payment

Make Payment

Figure 4 : Transaction Page

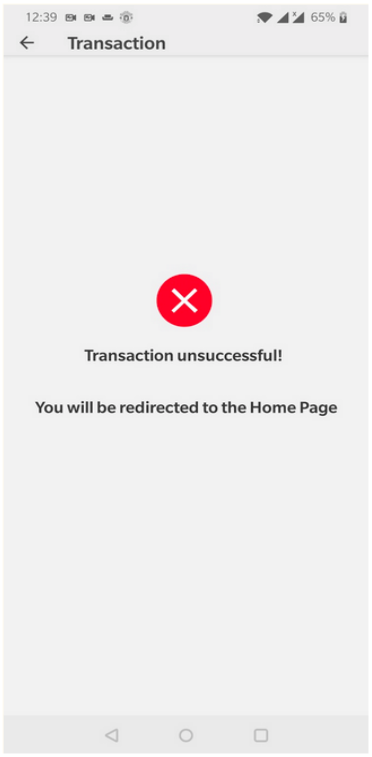

12:33 65%

Transaction

Transaction successful!

You will be redirected to the Home Page

Figure 6 : Transaction Successful



Transaction unsuccessful!

You will be redirected to the Home Page

Figure 7 : Transaction Failed

V. Conclusions

In this article we present a novel prototype of an offline wallet payment using React Native as our Frontend, NodeJS as our Backend and MongoDB as our database. The main idea of this prototype application was to handle and prevent intrusion attacks and vulnerabilities, to ensure this we used NodeJS' socket.io library and made sure all our API calls were safe. For Information Assurance and Security, we have made use of NodeJs' bcrypt module, where we hash and store all passwords to our database, thereby reducing the chance of threats to an end user's wallet. To establish concurrency and to avoid delay we implemented our own document level lock. Furthermore, for the offline mode of our prototype, the application makes use of React Natives' Async Storage which is a global storage system which we used to store the transaction amount and bank details. This work was drafted and implemented under the guidance of Dr. Bhawana Rudra, Department of Information Technology, NITK Surathkal. We thank her for her continuous inputs and valuable conceptual assistance throughout the course of the project. We would also like to express our gratitude towards Mrs. Thanmayee S., Department of Information Technology, NITK Surathkal for her help, support and guidance.

VI. References

I Jayasinghe, D., Markantonakis, K., Gurulian, I., Akram, R. and Mayes, K., 2016. Extending EMV Tokenised Payments to Offline-Environments. 2016 IEEE Trustcom/BigDataSE/ISPA,.

II San, A. and Sathitwiriyawong, C., 2016. Privacy-preserving offline mobile payment protocol based on NFC. 2016 International Computer Science and Engineering Conference (ICSEC),.

III Kiran, N. and Kumar, G., 2011. Building robust m-commerce payment system on offline wireless network. 2011 Fifth IEEE International Conference on Advanced Telecommunication Systems and Networks (ANTS),.

IV H. Zhao and S. Muftic, ”The concept of Secure Mobile Wallet,” 2011 World Congress on Internet Security (WorldCIS-2011), London, UK, 2011, pp. 54-58, doi: 10.1109/WorldCIS17046.2011.5749882.

V M. A. Hassan and Z. Shukur, ”Review of Digital Wallet Requirements,” 2019 International Conference on Cybersecurity (ICoCSec), Negeri Sembilan, Malaysia, 2019, pp. 43-48, doi: 10.1109/ICoCSec47621.2019.8970996.

VI W. Adi, A. Al-Qayedi, A. A. Zarooni and A. Mabrouk, ”Secured multi-identity mobile infrastructure and offline mobile-assisted micro-payment application,” 2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No.04TH8733), Atlanta, GA, USA, 2004, pp. 879-882 Vol.2, doi: 10.1109/WCNC.2004.1311302.

VII R. Tso and C. Lin, ”An Off-Line Mobile Payment Protocol Providing Double-Spending Detection,” 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), Taipei, Taiwan, 2017, pp. 570-575, doi: 10.1109/WAINA.2017.56.