A Decentralized Application for

Location Tracking: dApp GPS Tracker

By

Sainath Padala

Bachelor of Technology, Electronics and Communication Engineering,

Gandhi Institute of Technology and Management, 2017

A Report Submitted in Partial Fulfilment of the Requirements for the Degree of

Master of Engineering

In the Department of Electrical and Computer Engineering

University of Victoria

Supervisory Committee

Dr. Riham AlTawy, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Navneet Kaur Popli, Departmental Member

(Department of Electrical and Computer Engineering)

Abstract

Ensuring the safety of the family members is always a Parent's priority. Nowadays, a mobile phone is a necessary gadget present with every person. It will be a lot easier if a Parent can ensure the family's security right from the phone. Thus, in this project, a decentralized mobile application (dApp) is developed, which helps the Parent to track their Child's location.

This dApp provides various features such as getting the child's current location and tracking location when they are on the move. Moreover, the Parent can use this location data and open their preferred map to navigate to their Child's location. This dApp guarantees maximum accuracy, precision and privacy of data. Data privacy is ensured by encrypting the coordinates with a secret password known only to the Parent and the Child. The encryption algorithm used in this dApp is AES (Advanced Encryption Standard), which is the most popular and is an industry-standard for security.

Although many similar applications offer these features, they are controlled by a central authority, where total control over data lies. If the central authority is compromised or goes down, then user's data is no longer safe, and there is a chance data gets deleted. To address this issue, this dApp uses blockchain in the backend. Blockchain is a distributed ledger, and there are many advantages in using it compared to a centralized entity. Blockchain ensures data is readily available across multiple nodes, data is tamper-proof, and gives the user control of the application.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank my project supervisor, Dr. Riham AlTawy, whose valuable insight, guidance, and suggestions have helped me immensely during my Master of Engineering research and coursework. This undoubtedly helped me in the completion of this project.

Further, I would like to thank Nokia, Ottawa, for providing me with the opportunity to work as an intern. This experience was instrumental in shaping my professional and technical skills as well as gaining invaluable practical knowledge.

Finally, I would like to thank my family for their motivation and support during my project work and throughout my MEng studies.

Chapter 1: Introduction

A Decentralized Application (dApp) is an application or program stored and executed on the blockchain's network.

Decentralized Application for Location Tracking – dApp GPS Tracker is a type of application used to track the user's location. The main motive of this application is to establish a secure location tracking system between a Parent and a Child (or a family member).

By using this dApp, the Parent can track the current location of their Child and also listen to the location updates if the Child makes a move. At the Child's end, they have an option to "Activate Tracking" and "Stop Tracking." As soon as the Child turns on the "Activate Tracking" option, the current GPS coordinates of the mobile are retrieved, and then the data is sent to the blockchain. The Parent then retrieves the GPS coordinates present in the blockchain and can know the Child's location. As long as the tracking is turned on at the Child's end, the GPS coordinates will be sent continuously if the Child's location crosses a certain distance (the distance can be defined by the user, by default, set to 100 meters).

The data flow between the dApp and the blockchain is encrypted to ensure confidentiality. Confidentiality is guaranteed by using a secret key to encrypt and decrypt the data. The encryption algorithm used in this dApp is AES (Advanced Encryption Standard) [5].

1.1 Problem Definition and Motivation

With the growth of science and technology, a person can ensure their family's safety in multiple ways. One way to achieve this is by tracking the location of family members using

mobile phones. In the real world, multiple applications provide location tracking functionality. For example, most common applications such as Google Maps [7] and Apple Maps [8] provide location-sharing functionality. These maps are controlled by a central entity or authority such as Google and Apple, and they ensure that the data is secure, reliable and confidential.

However, as the user trusts this central authority, if it is compromised, the whole data stored in its servers and the data that flows through them will also be compromised (deleted or leaked or manipulated), thereby endangering the safety of the user. Thus, blockchain is used as the backend in this application to address these issues. Blockchain is an immutable distributed ledger. The data stored in the blockchain is duplicated across multiple nodes present in its peer-to-peer network. Hence, the data cannot be deleted or manipulated. And also, in combination with different encryption algorithms, the data stored in the blockchain can be saved as ciphertext and can only be decrypted using a secret key.

1.2 Outline

The structure of this report is as follows:

Chapter 1: This chapter introduces the developed decentralized application and an overview of how it works. It also shows the main motivation behind developing such an application.

Chapter 2: This chapter briefly introduces blockchain and the basic concepts.

Chapter 3: This chapter explains the proposed architecture for the dApp and how it is implemented by using different tools such as Remix, Infura, MetaMask, Flutter, Etherscan etc.,

Chapter 4: This chapter discusses the costs incurred by the user by using this application.

Chapter 5: This chapter summarizes all the steps performed in developing the application and how it can be further developed in future.

Chapter 2: Blockchain Technology

2.1 Blockchain

A blockchain can be defined as a series of blocks that contain information. Each blockchain block consists of the cryptographic hash of the previous block, timestamp, and transaction data. These blocks are connected as a "chain" in the network through peer-to-peer nodes.

Blockchain is also referred to as "Distributed Ledger Technology." Blockchain enables the irrefutable documentation that a certain transaction took place and ensures such transaction is tamper-proof because of hash values and proof of work. Blockchain is also used to timestamp documents so that no one can tamper with the documents in the future or backdate them.

2.2 Blockchain Concepts

2.2.1 Transaction Process

The steps followed to ensure a successful transaction are:

1. A new transaction is created or entered.
2. Then this transaction is broadcasted through the whole blockchain network.
3. Then all the nodes present in the peer-to-peer network validate the transaction.
4. If the nodes identify the transaction as legitimate, they add the transaction into a new block in the blockchain.
5. Then, the nodes will receive a reward or fee for their Proof of Work.
6. The new block is then added to the blockchain and marks the completion of the transaction.

7. Now, all the remaining network nodes get a copy of this newly created block and keep the data to validate further transactions.

2.2.2 Architecture

A Block in a Blockchain generally contains:

- Data (example: transactions, money, documents, messages etc.,)
- Hash – the hash value of the complete block.
- Previous hash – the hash value of the previous block.

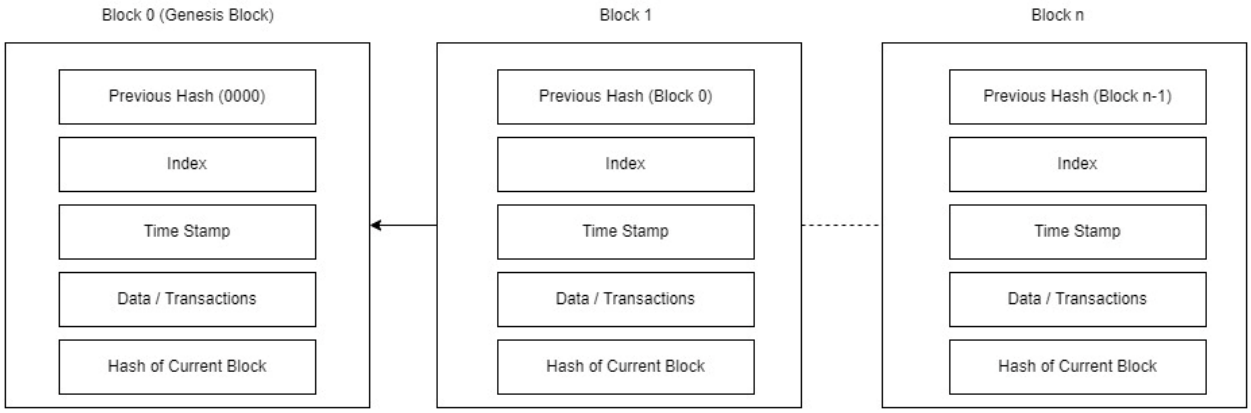


Figure 2.1: Blockchain Architecture

When a block is ready to be added to the blockchain, it is broadcast across the network. Then mining is performed, and miners add the block into the blockchain by chaining it with the previous block, creating a ledger. In the same way, multiple blocks can be created and added to the blockchain.

For example, as each block in the chain contains the previous block's hash value, if a hacker tampers the data of a previous block, then the hash value of the tampered block changes and

will also change the hash values of the subsequent blocks at his side. But the hacker's copy of the blockchain does not match with blockchain copies present at the other nodes, and thus hacker's data will be cast away.

Genesis block is the first block of the entire blockchain. That is why it doesn't have any previous hash value, and the default value is '0000'.

2.2.3 Peer to Peer Network (P2P)

Peer to Peer network model is the foundation of the blockchain. In this network, a group of nodes or devices collectively stores data, files, hashes etc., and each node acts as an individual peer. In P2P, there is no central entity or server, which controls nodes, and all the nodes have equal power. In this model, all the peers or nodes find each other and maintain the synchronization of their blockchain state.

For example, if a node is added to the blockchain network, that node gets the full blockchain copy. Then, if this node or any other node adds a block into the blockchain, all the other nodes must verify and validate this block and reach ***Consensus***. Then, they agree on whether a block must be added or dropped from the chain.

2.2.4 Proof of Work

To add a block, the transactions must be validated by running complex cryptographic operations. Miners perform this task. Miners run blockchain software and, after validating the transaction by reaching Consensus, adds the block to the blockchain. This process is called "Mining." After successfully adding the block to the blockchain, a fee is paid to the miner for

their effort (for example, BTC for Bitcoin mining and Ether for Ethereum mining). The cryptographic calculations performed by these miners is called ***Proof of Work (PoW)***.

2.3 Decentralized Application (dApp)

There are three types of applications over the internet. They are:

- **Centralized Applications:** Most of the internet applications in the world are Centralized. They are controlled by a central authority or company which manages the application's servers, functionalities etc.., This central entity also maintains a database to hold the data of all the users. So, users must solely trust this central entity to protect their data.
- **Distributed Applications:** In distributed applications, the computation is distributed across various nodes. Different nodes work on different functionalities, but their main goal is the same. All the nodes work together by interacting with each other to achieve that goal.
- **Decentralized Application (dApp):** Unlike distributed applications, where computation is divided across different nodes to strive for a single goal, in decentralized applications, all the nodes work individually and work towards their own goal. Thus, no node is instructing any other node to do a task. The code or dApp runs on a node on a peer-to-peer network, and no other node has control over this dApp.

2.4 Ethereum Blockchain

Ethereum is an implementation of blockchain created by Vitalik Buterin. Ethereum can run smart contracts or program codes of decentralized applications (dApps) in addition to tracking cryptocurrencies. Smart contracts can also be used to store data in a distributed ledger.

Ethereum has two types of currencies: **Gas** and **Ether**. Ether is the cryptocurrency used by Ethereum and Gas is the amount of fees required to conduct a transaction by a node on Ethereum. Gas is usually calculated in Ether. Units for Gas is 'gwei,' and one gwei is equal to "10^{-9} ETH". To add a transaction or execute a code, Gas must be specified in the transaction.

2.4.1 Ethereum Virtual Machine (EVM)

Ethereum Virtual Machine (EVM) is a virtual environment that runs on all Ethereum network nodes. EVM acts as a decentralized computer that executes smart contracts and deploys them. As every node runs on the EVM, they maintain consensus across the blockchain.

Smart contracts are first written in Solidity, and at the time of deployment, they are converted to "Bytecode." Bytecode is then converted into operation codes or opcodes. EVM then interprets opcodes to complete the required tasks.

2.4.2 Wallets, Accounts and Addresses

A Wallet is a tool used to manage Ethereum Account. Using it, one can view their balance, initiate transactions and has many other features.

An Address can be defined as the name of each account (like an email address). Using addresses, multiple accounts can send or receive Ethers or information in Ethereum.

Ethereum has two types of accounts:

- Externally Owned Accounts (EOA): These accounts are used by nodes to send or receive Ether. These accounts are associated with a private key.

- Contract Accounts: These accounts hold and execute smart contract code. Unlike EOAs, contract accounts do not have a private key.

2.5 Smart Contract

A smart contract is a software program that resides in EVM. It is used to trade cryptocurrency between users, transfer assets or information, data, and many types of transactions. Functions are defined inside the smart contract, and whenever a function is called or executed, it runs as a transaction. Functions are used to add or retrieve data, send or receive Ether etc.,

2.5.1 Solidity

Solidity is an objective-oriented programming language created to write smart contracts in Ethereum. It is a high-level language similar to C++, Python, and JavaScript.

Solidity Compiler (solc) compiles smart contracts into EVM byte code and ABI. EVM byte code contains machine-code-like instructions, and ABI contains metadata of all the variables, methods present in the smart contract.

2.5.2 Remix

Remix is an open-source web and desktop tool used to write smart contracts. There are many tools apart from Remix for writing smart contracts, but Remix provides more compatibility, plugins etc.., Remix IDE also has modules for testing, debugging and deploying smart contracts.

2.6 MetaMask

MetaMask is a popular Ethereum wallet. Ether (ETH) or any ERC-20 token can be stored in this wallet. Using it, a user can send or receive Ether. A user can also link this wallet to their smart

contract to pay fees associated while deploying smart contract and pay fees when adding new data into the blockchain.

2.7 Infura

Infura provides tools and infrastructure for developers to integrate their applications with the Ethereum network. Infura solves few major problems faced by developers:

1. If there is a need to save files on the blockchain, it is very expensive. But Infura, in combination with IPFS, allows users to store data of such large files in IPFS, thereby storing the resultant hash value obtained from IPFS in the Ethereum blockchain. Thus, it solves the problem of space and cost.
2. A lot of infrastructure is needed to run a node. But with Infura, it takes care of all the infrastructure and improves synchronization with the geth client (which may take hours without infura). And also, Infura provides TLS enabled endpoints.

2.8 Encryption and Decryption

Encryption is the process of converting plaintext or normal data into encoded (or ciphertext) data. Similarly, decryption is the process of converting encoded (or ciphertext) data back to plaintext or normal data.

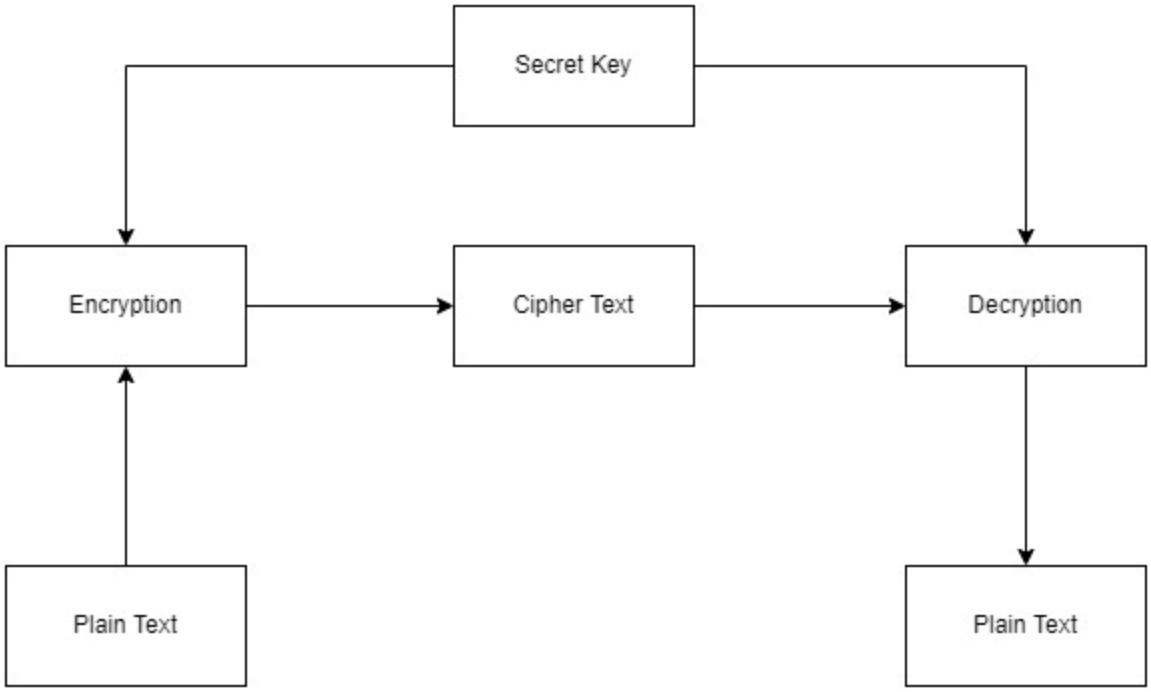


Figure 2.2: Encryption and Decryption (Symmetric)

In this project, the algorithm used for encryption and decryption is AES [5]. AES is a symmetric encryption algorithm that uses a single secret key for encryption and decryption.

Advanced Encryption Standard (AES), also known as the Rijndael algorithm, is a symmetrical block cipher that takes plain text as 128-bits and encrypts it with 128/192/256-bit keys to obtain a ciphertext. It is stronger and faster compared to it its predecessor DES.

If k is the secret key, and plaintext is the input, then:

$$Cipher = E(k, plaintext)$$

Similarly,

$$plaintext = D(k, ciphertext)$$

Chapter 3: Proposed Decentralized Application

In this chapter, a decentralized application (dApp) is developed using different tools such as Remix, MetaMask, Flutter and Etherscan.

3.1 System Architecture

The below figure shows the basic architecture of the application.

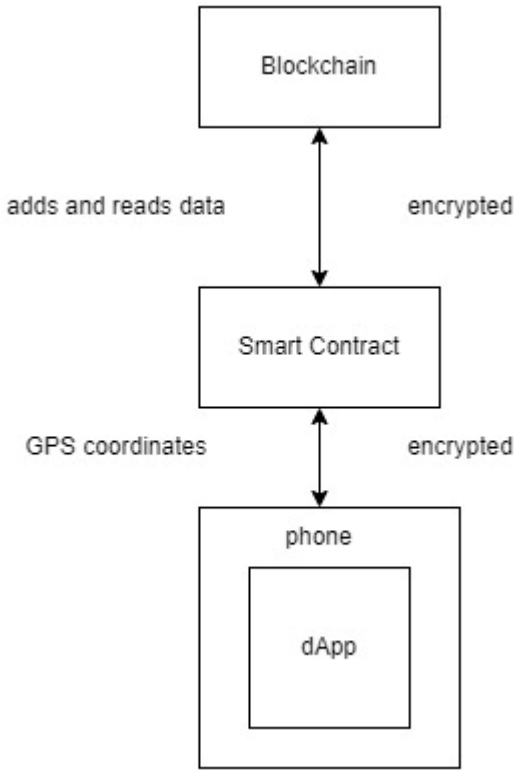


Figure 3.1: Proposed Decentralized Application (dApp) Architecture

First, the decentralized application is installed on the phones of Parent and Child. Then, in the Child's phone, the application retrieves the GPS coordinates, encrypts the coordinates using a password and sends the encrypted data to the smart contract. The smart contract then stores the data onto the Ethereum blockchain. Whenever a child changes location, the new coordinates (encrypted) are sent to the smart contract again. At the Parent's end, the Parent

reads the latest data stored in the block chain and decrypts the data into coordinates using the password.

3.2 Setting up MetaMask Wallet

MetaMask is a cryptocurrency wallet. For deploying smart contracts on the Ethereum network and doing transactions, a certain amount of fee must be paid in the form of “Gas” to the miners.

To use MetaMask wallet:

- Install MetaMask [11] extension on the desired web browser.
- Setup Account and Test network (in this project, Rinkeby test network is used)
- Load funds into Rinkeby account address by using crypto faucets.

After completing the above steps, the wallet is loaded as below:

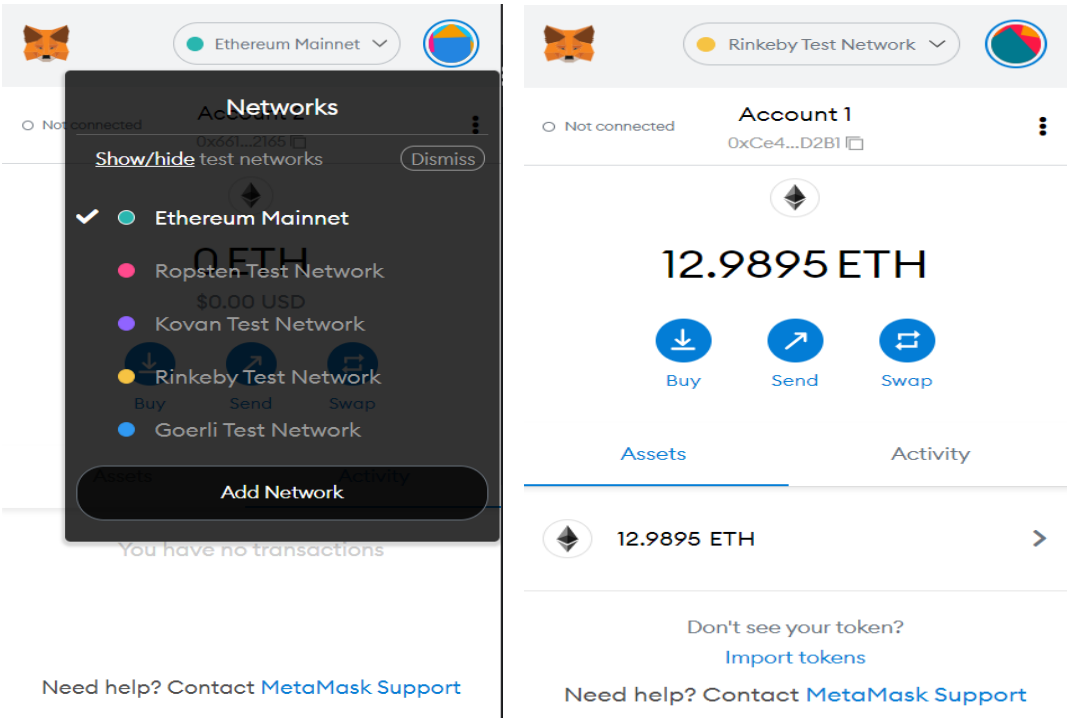


Figure 3.2: Different networks in MetaMask wallet and Balance in Test account

3.3 Deploying Smart Contract via Remix

Smart contracts can be written in any IDE, but the most standard IDE is the online IDE – Remix [12]. Remix is an exclusive Ethereum IDE that has a wide variety of plugins, solidity compiler etc.., As discussed in earlier chapters, Solidity is the programming language used to write smart contracts.

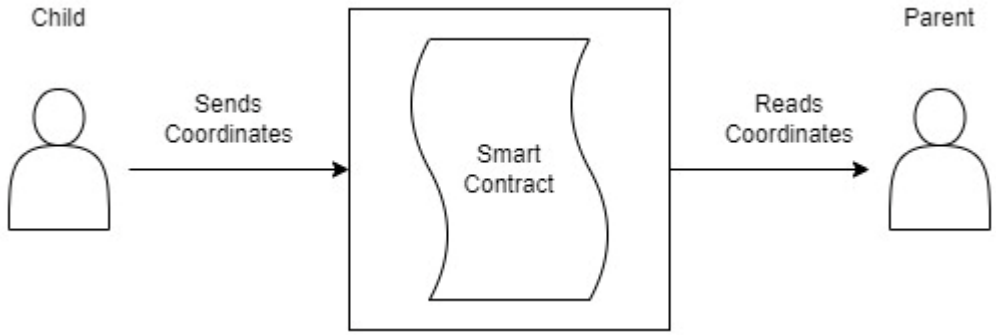


Figure 3.3: Interaction with Smart Contract

Below image shows the smart contract used in this application:

Home | Project.sol

```
1  pragma solidity ^0.5.16;
2
3  contract Project {
4
5      string[2] coordinates = ["0.00", "0.00"];
6
7      function sendCoordinates(string memory _latitude, string memory _longitude) public{
8          coordinates[0] = _latitude;
9          coordinates[1] = _longitude;
10     }
11
12     function readCoordinates() view public returns(string memory, string memory) {
13         return (coordinates[0], coordinates[1]);
14     }
15 }
```

Figure 3.4: Smart Contract in Remix IDE

- The smart contract of name “Project is created.”
- The first line of code: “pragma solidity ^0.5.16,” is called the Pragma Directive. Pragma directive shows which version of solidity compiler to be used.
- Coordinates, an array of size two is created. This array holds the latitude and longitude values.
- Two functions: sendCoordinates() and readCoordinates() are defined.
- sendCoordinates(), adds the array which consists of latitude and longitude onto the blockchain. It consists of two input parameters: “_latitude” and “_longitude.” These are the values received from the mobile application. By default, the read access of a function is restricted. Thus, “public” needs to be defined to access the state variable.
- readCoordinates(), reads the latest coordinates stored in the blockchain. As it only uses “view,” there is no need to make a new transaction to read data. Thus, it doesn’t cost any Ether.

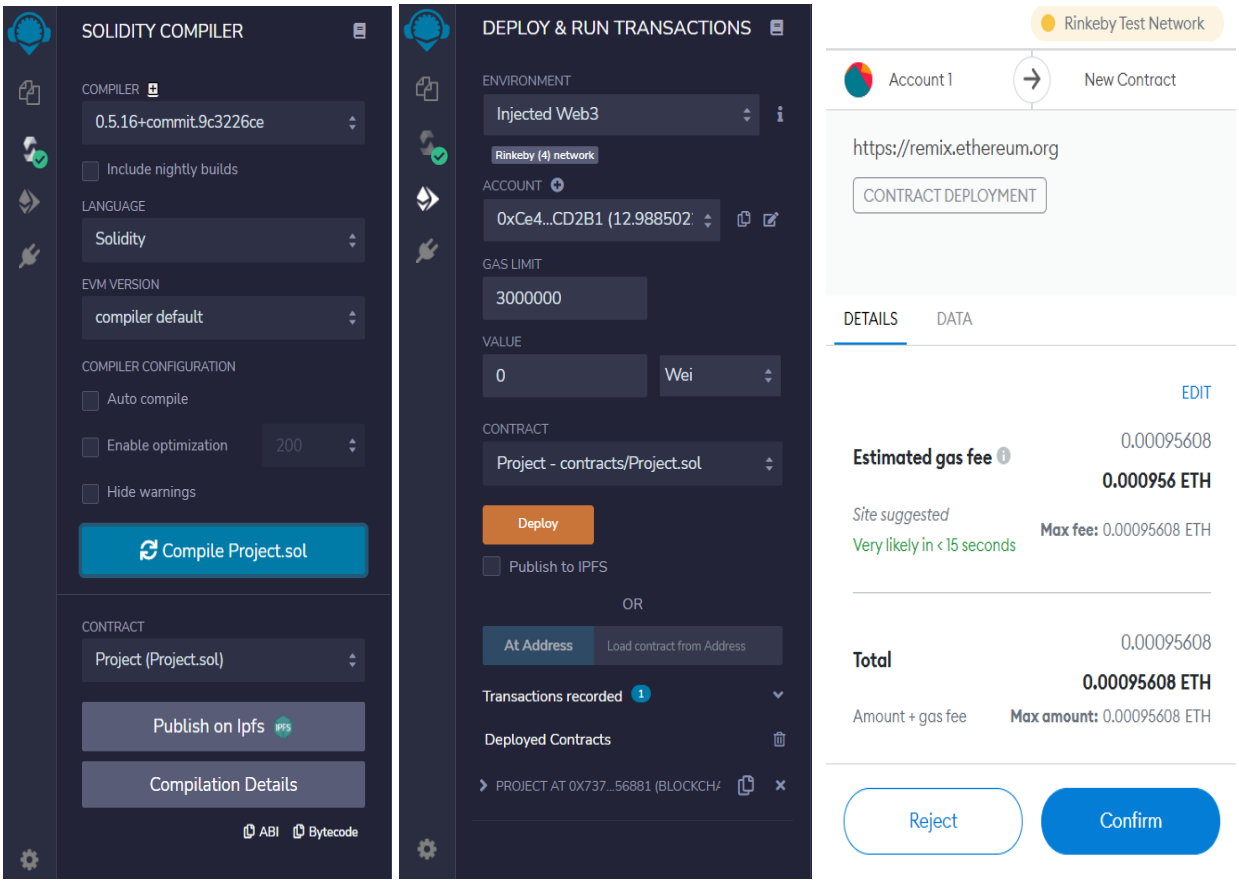


Figure 3.5: Compiling and Deploying Smart Contract by approving transaction in MetaMask wallet

From the above images:

- The contract is first compiled by selecting the required solidity compiler. And the “ABI,” which is obtained after compiling the contract, must be copied to the flutter application’s development directory (“assets/abi.json”) in a later part. *Contract Application Binary Interface (ABI)* interprets the standard way smart contracts can interact with other applications from outside the blockchain.
- Then, the contract is deployed by selecting the Environment. As MetaMask is used here, the Environment must be selected as “Injected Web3”. MetaMask will give a pop-up when deploying, indicating that a smart contract is being deployed and the transaction must be confirmed.

- In the terminal, it shows that the contract is deployed successfully and various other useful values like Gas, addresses etc.,

Figure 3.6: Details of the deployed Smart Contract in terminal

- After deploying the contract, the contract address can be obtained by copying the value present at the bottom of figure 3.5.

The total cost incurred for deploying the smart contract onto the blockchain will be discussed in chapter 4.

3.4 Infura

Infura is an API used to connect the mobile application (or dApp) to the smart contract. It handles all the requests to the smart contract and acts as a Gateway.

To use Infura API, first, an account must be set up. Then, a project must be created for this application, and all the stats of requests made to a smart contract can be checked here.

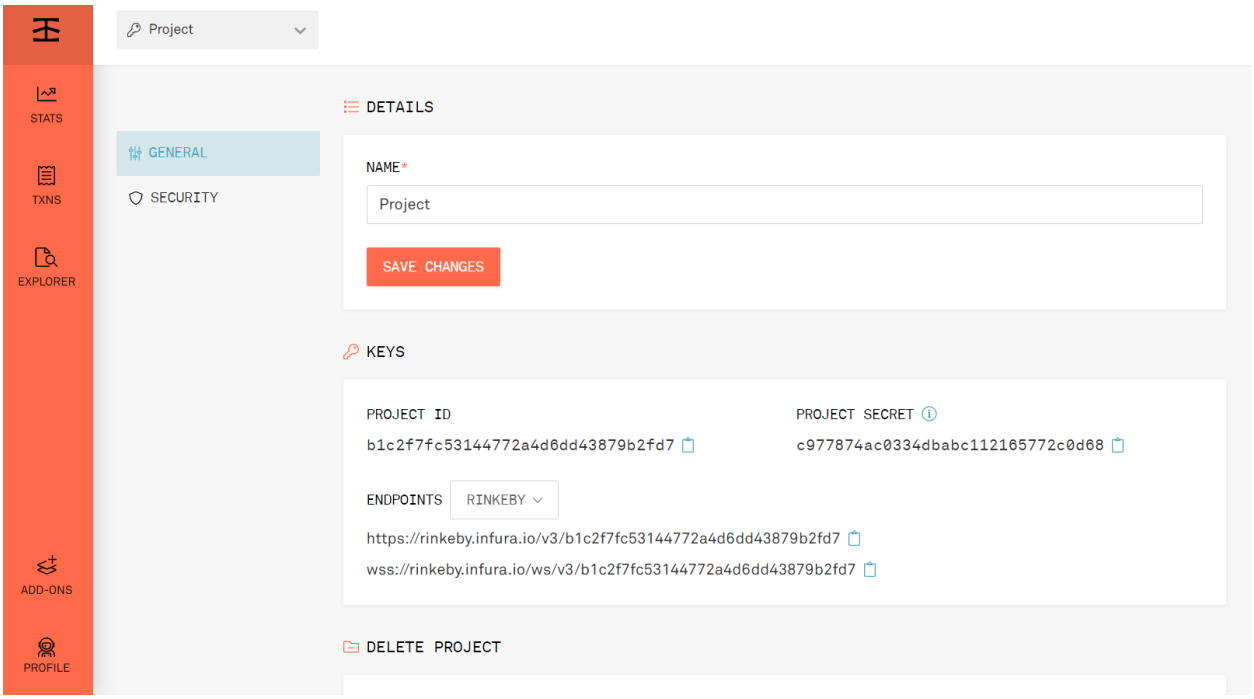


Figure 3.7: Rinkeby Endpoint in Infura

The API or Endpoint can be obtained by selecting the appropriate network in the drop-down.

3.5 Mobile Application Design

Below figure shows the overview of the mobile application:

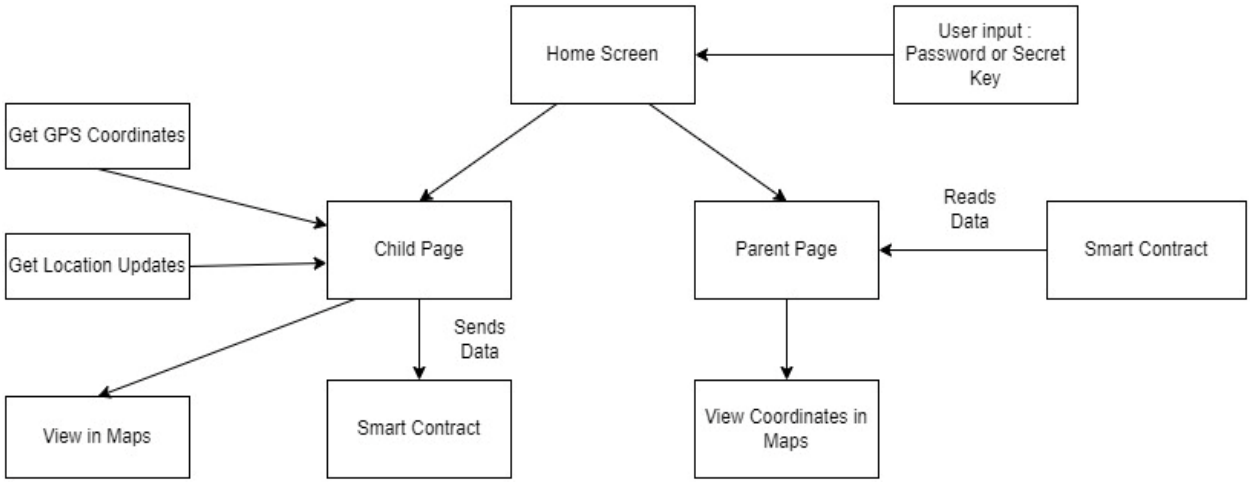


Figure 3.8: Mobile Application Architecture

3.5.1 Flutter

Flutter is used to develop the application in this project. Flutter is an open-source and cross-platform framework used to create mobile applications for Android and iOS. Google created flutter SDK (Software Development Kit) using Dart programming language. It is simple, powerful, and faster than most traditional development methods present for Android and iOS. Using this framework, a developer can develop an application for Android and iOS by making only a few minor changes to their code.

Everything is a Widget – is the core concept of Flutter and the whole application is itself a widget. Widgets are the building blocks to develop the user interface in Flutter. User components like buttons, text fields, input/output fields, sliding bars etc.., are all widgets, and these all are placed on top of the main widget.

3.5.1.1 Prerequisites

Before setting up a flutter project below prerequisites must be met:

1. Setup an Emulator (Android or iOS)
2. Install a code editor (Visual Studio Code or Android Studio)
3. Install Flutter and the required extensions.
4. Understand a few basics of Dart Programming Language.
5. Create and initiate flutter project in a directory.

3.5.1.2 Required Dependencies and Libraries

To develop the dApp and use some functionalities, a few dependencies must be added to the pubspec.yaml file. They are:

- web3dart: It is a dart library used to connect with the Ethereum node to send transactions and interact with smart contracts.
- http: It is a dart library used to request http connections and its resources.
- geolocator: It is a flutter plugin used to retrieve data regarding GPS coordinates. Some of the important features include current GPS location, last known location, location updates etc.,
- url_launcher: It is a flutter plugin used to launch specific URLs. In this dApp, this plugin launches Google Maps and Apple Maps by using GPS coordinates.
- location: This plugin works similar to the geolocator plugin. It is used along with the geolocator to get precise coordinates.
- provider: It is a flutter plugin that is used for a few essential functionalities like future values etc.,
- encrypt: It is a flutter plugin that is used to encrypt and decrypt data.

Some of the features vary depending upon the version of the plugin. The versions of plugins used in this project are:

Flutter Plugin	Version
web3dart	2.3.3
http	0.13.4
geolocator	8.0.1
url_launcher	6.0.17
location	4.3.0
provider	6.0.2
encrypt	5.0.1

Table 3.1: Flutter Plugins and versions

3.5.2 Home Page

After installing the application in the mobile of Parent or Child, the user interface will look as shown below:

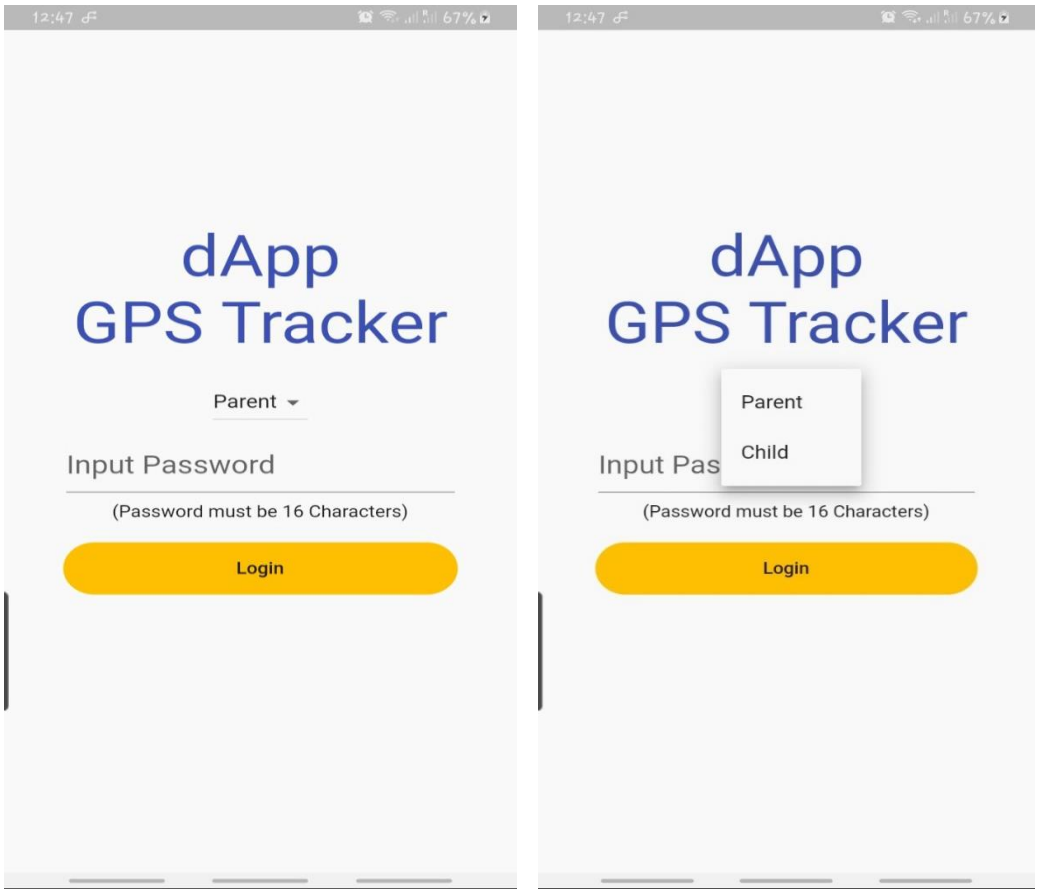


Figure 3.9: Home Page of the dApp

As discussed earlier, in Flutter, Everything is a Widget. Thus, the home page has four widgets. They are:

1. Text field: Name of the application – “dApp GPS Tracker.” It is a text widget that displays the name of the application.
2. Drop-down: “Parent” and “Child.” The drop-down widget identifies whether the user is a parent or a child.
3. Text Input field: “Input Password.” Here, the user must input the 16-characters secret password used to either encrypt or decrypt the data that flows between the application and the smart contract. If the 16-characters password is not entered, it throws an error saying, “Invalid length: Please input 16 characters secret password” and will not navigate to the next page.
4. Button: “Login.” This button is used to navigate and load the next page. There are different pages for Parent and Child. The appropriate page is loaded depending upon the input from the drop-down.

3.5.2.1 Flow Chart

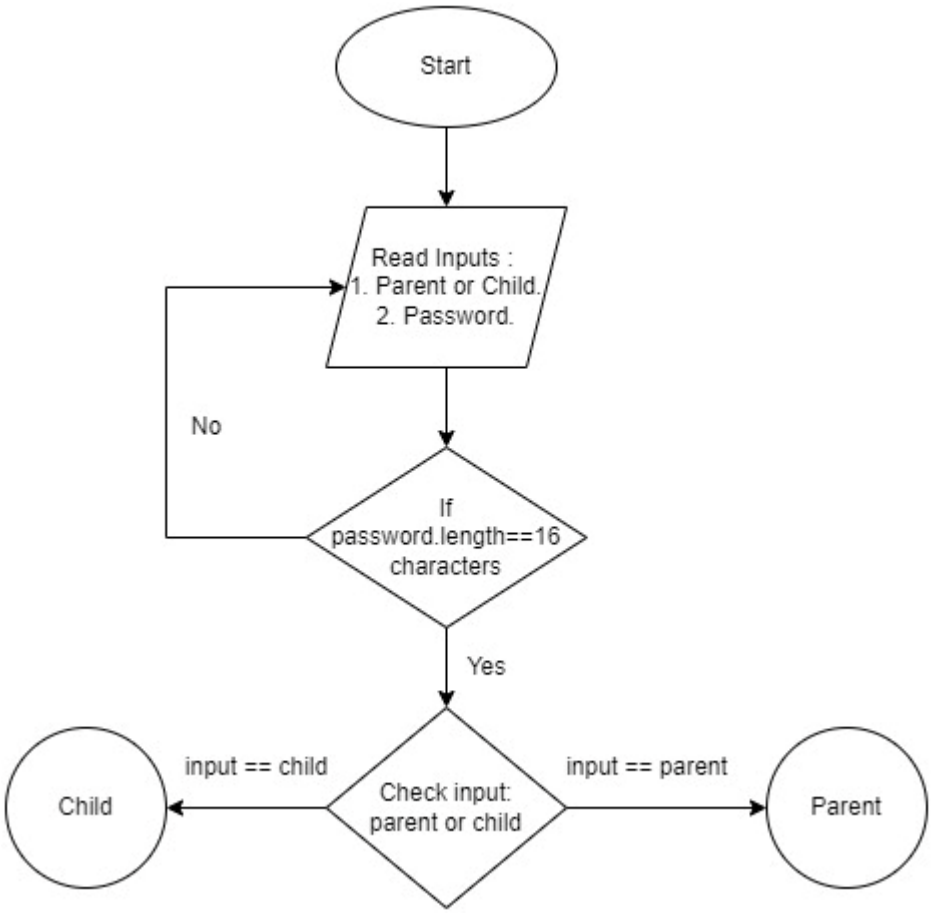


Figure 3.10: Flowchart for the Home Page functionality

3.5.2.2 Pseudocode

The Pseudocode for implementing the functionality of the home page is:

Algorithm 1: Implementing Home Page

Inputs:

using Dropdown: *value(1: Parent, 2: Child);*

using TextFormField: *password;*

```
if password.length == 16
        if value == 1
                parentPage() ;
        else if value == 2
                childPage() ;
        else
                Text("Select Parent or Child");
else
        Text("Invalid length: Please input 16 characters secret password")
```

3.5.3 Child Page

If at the Home Page, “Child” is selected, then the below page is loaded:

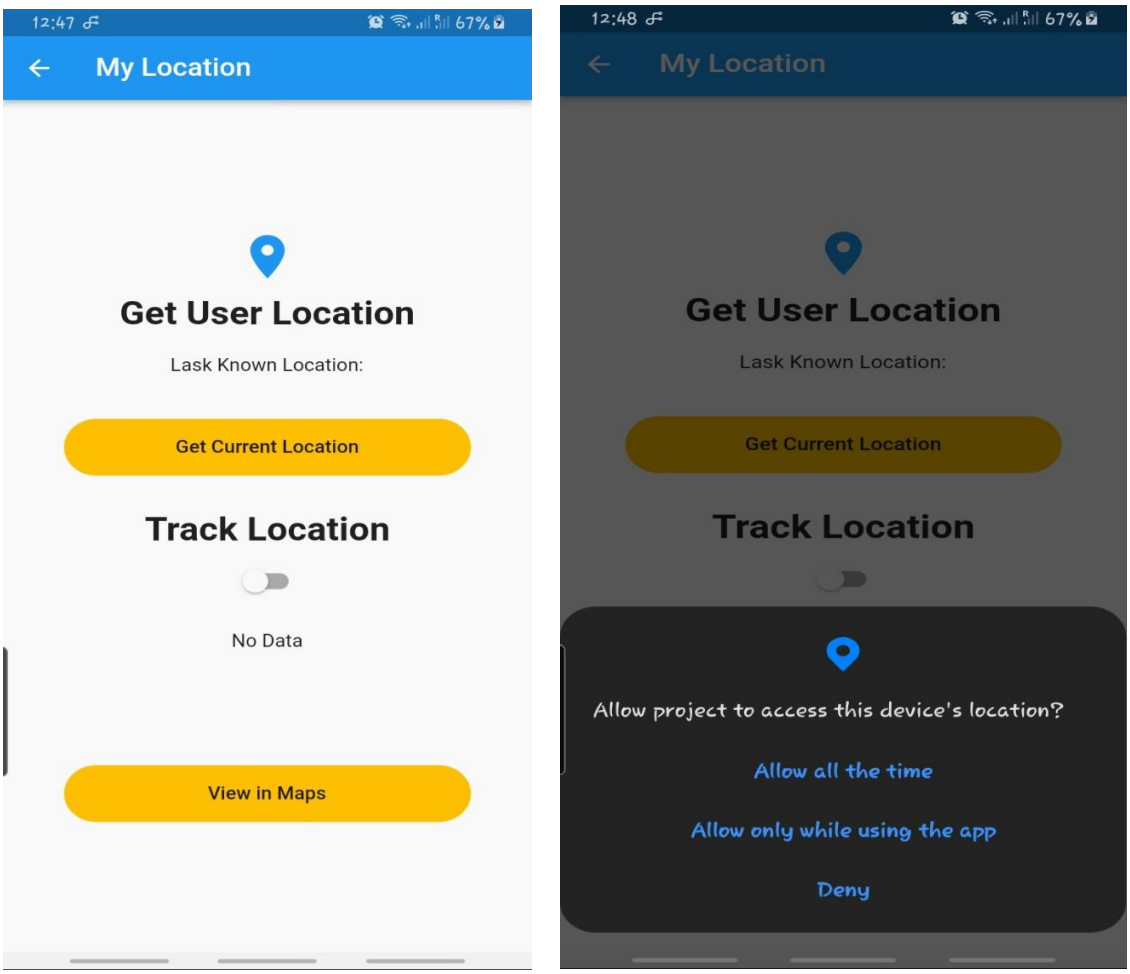


Figure 3.11: Child Page and a pop-up asking for location permission

The Child Page consists of various text fields and buttons. As shown in the figure, initially, there is no data present on the page. The different functionalities present on this page are:

- Button: “Get Current Location.” When this button is pressed, the application will first request location permission from the user (in this case: Child). Then, the application will retrieve the user's current location and display it in the “Last Known Location field,” as shown in the below figure.

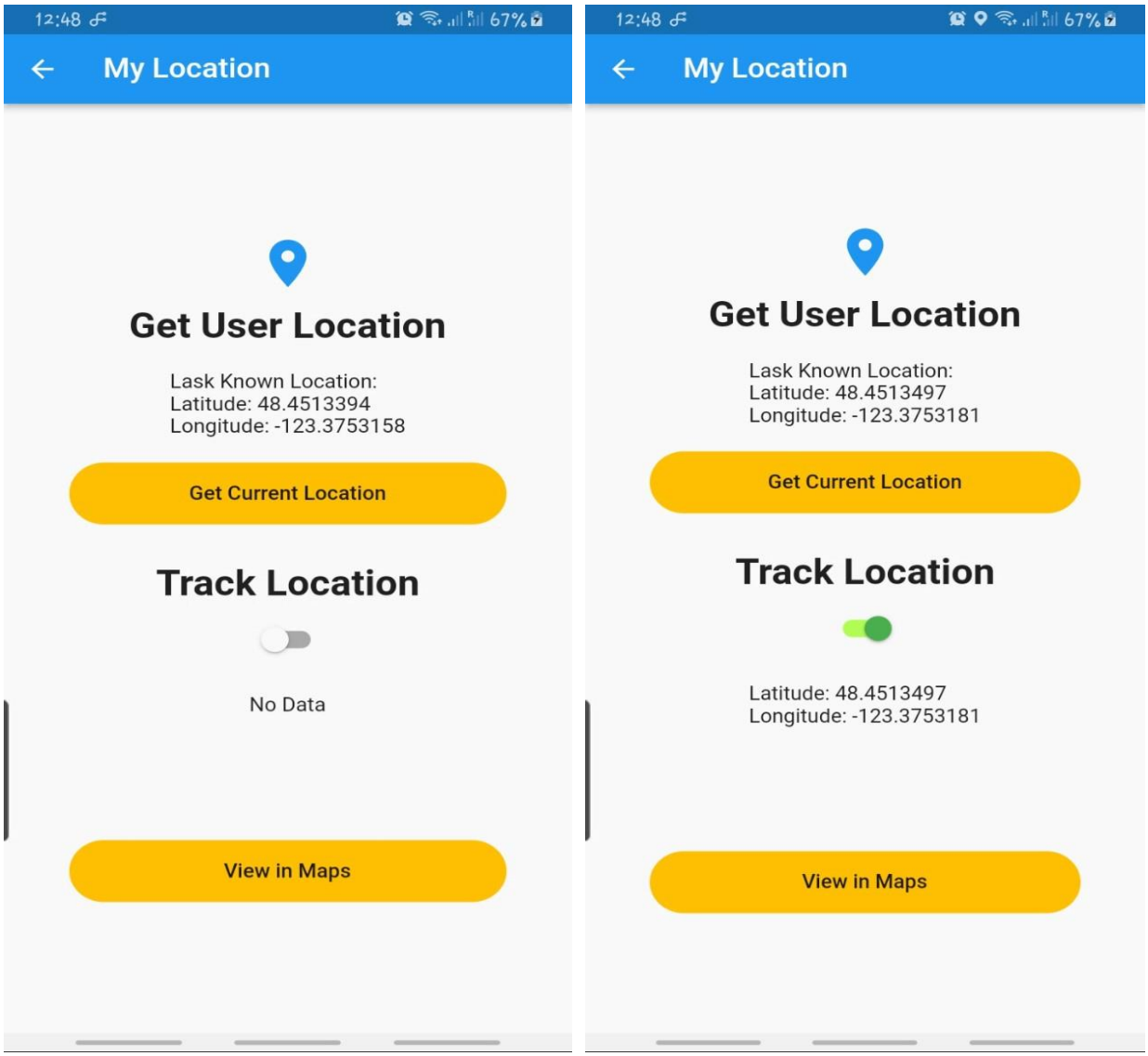


Figure 3.12: Current Location and Track Location (on/off)

- Switch: “Track Location.” The application will start location tracking when this switch is turned on. And when it is turned off, the application will stop sending the coordinates to the blockchain. Operations that occur when this switch is turned on are:
    1. Retrieves the Child's current position and gets the coordinates: Latitude and Longitude.
    2. The coordinates are then encrypted using the password (obtained as input from Home Page).

3. The encrypted data is then sent to the smart contract.
4. The smart contract adds this data as a transaction onto the Ethereum blockchain.
5. When the position of the Child changes, suppose the Child crosses 100 meters of the current GPS location radius, then the new location is retrieved again, and then the above steps are repeated.
6. The application is tested in real-time, and the results are discussed in section 3.6.

- Button: “View in Maps.” When this button is pressed, the dApp will retrieve the last known coordinates from the above processes and pass the coordinates to Google Maps or Apple Maps. Then, depending upon the phone, the appropriate Maps application will be opened to view the location. This button is optional and can be used as a verification step to check their location.

3.5.3.1 Flow Chart

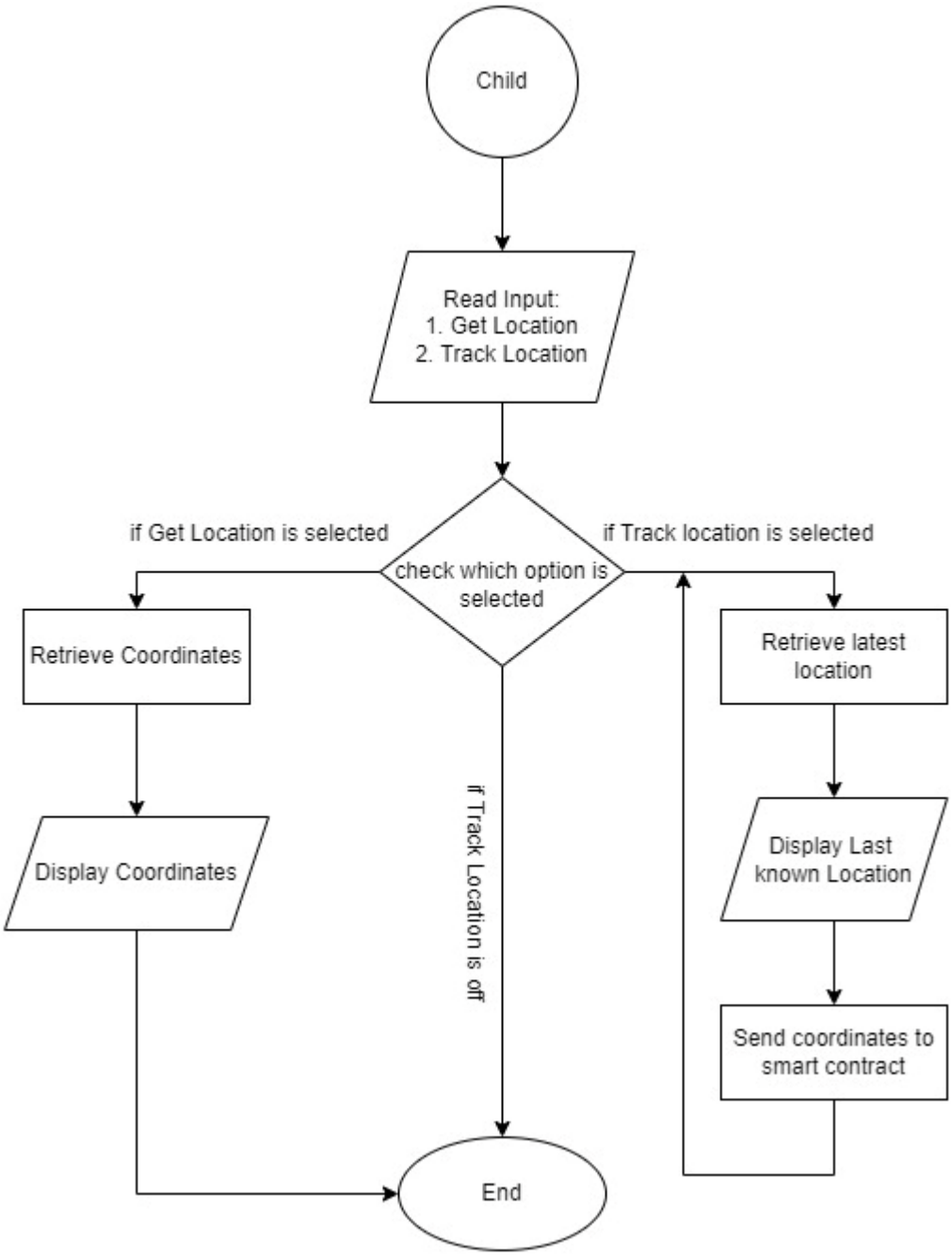


Figure 3.13: Flowchart for the Child Page functionality

3.5.3.2 Pseudocode

The Pseudocode for implementing the functionality of the Child Page is:

Algorithm 2: Implementing Child Page

Inputs:

using Button: *Get Current Location*

using Toggle Switch: *Track Location (on/off)*

using Button: *View in Maps*

password;

Get Current Location:

```
position = Geolocator().getCurrentPosition();
latitude = position.latitude
longitude = position.longitude
Text(latitude + "&" + longitude);
```

Track Location:

```
while tracking == true
    postitionStream = geolocator.getPositionStream().listen((Position position)  {
                                latitude = position.latitude;
                                longitude = position.longitude;
                                Text(latitude + "&" + longitude);
                                lat = EncryptionDecryption.encryptAES(
                                        latitude,
                                        password
```

```
                                        );
                                lon = EncryptionDecryption.encryptAES(
                                        longitude,
                                        password
                                        );
                                smartcontract.addCoordinates (
                                        lat,
                                        lon
                                        );
                                }
```

View in Maps:

```
googleMapsUrl =
        "https://www.google.com/maps/search/?api=1&query=$latitude,$longitude";
appleMapsUrl =
        "https://maps.apple.com/?q=$latitude,$longitude";
if  canLaunch(googleMapsUrl)
        launch(googleMapsUrl) ;
else if  canLaunch(appleMapsUrl)
        launch(appleMapsUrl) ;
else
        Text("Couldn't launch Apple Maps");
```

3.5.4 Parent Page

If at the Home Page, “Parent” is selected, then the below page is loaded:

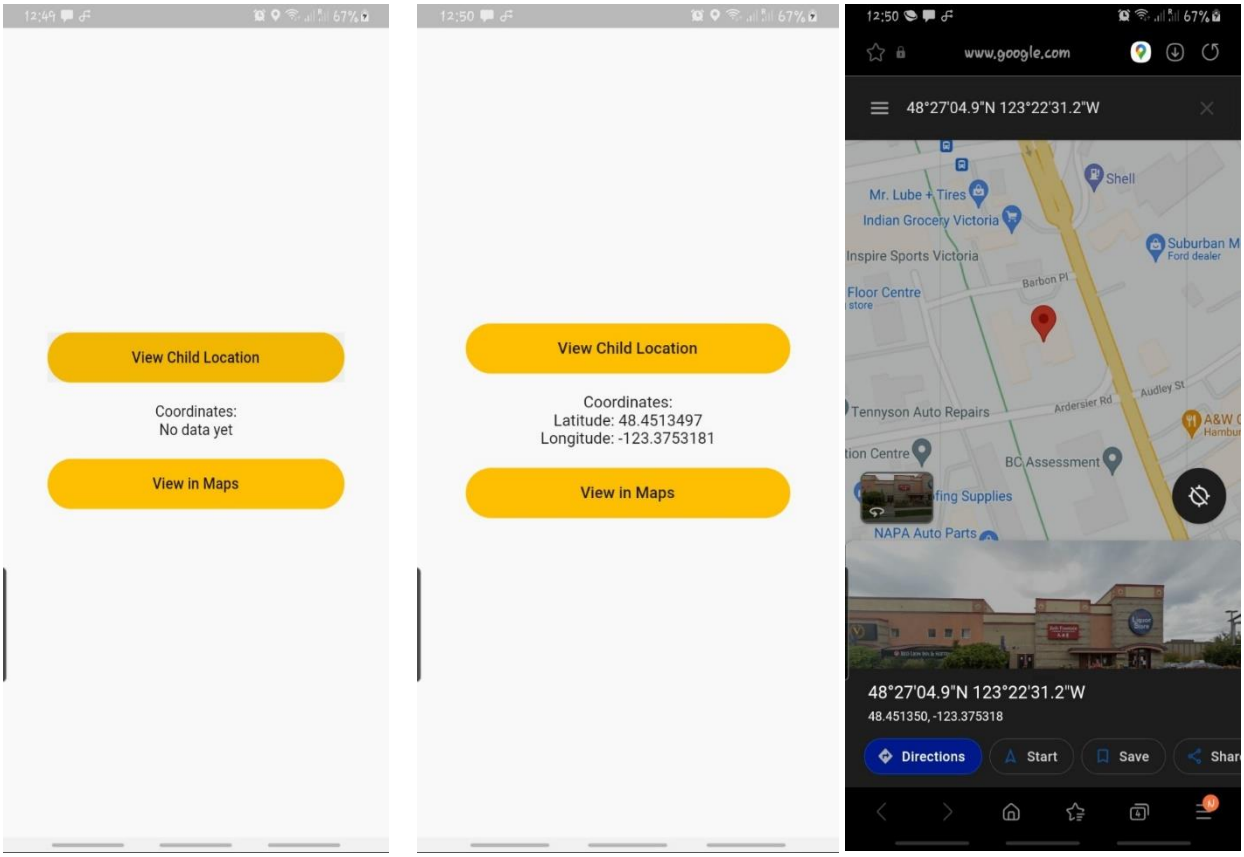


Figure 3.14: Parent Page and its functionality

The Parent Page also consists of various text fields and buttons. As shown in the figure, initially, there is no data present on the page. The different functionalities present on this page are:

- Button: “View Child Location.” Before pressing this button, the text field displays “No data yet.” Operations that occur when this button is pressed are:
    1. The dApp will contact the smart contract and retrieve the data present in the blockchain.
    2. The retrieved encrypted data is then decrypted using the 16-character password given by the Parent from the HomePage.

3. After decrypting the data, the resultant coordinates are displayed, as shown in the above image.

- Button: “View in Maps.” When this button is pressed, the dApp will retrieve the coordinates from the above process and pass the coordinates to Google Maps or Apple Maps. Then, depending upon the phone, the appropriate Maps application will be opened to view the location. This feature is used because Google Maps and Apple Maps are the most common applications in today’s phones. Thus, the Parent can easily navigate to the Child’s location with their preferred app.

3.5.4.1 Flow Chart

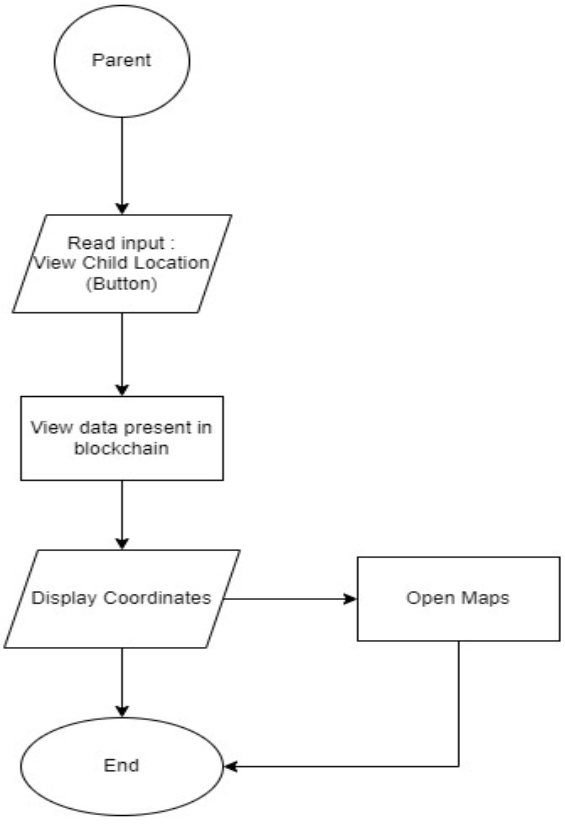


Figure 3.15: Flowchart for the Parent Page functionality

3.5.4.2 Pseudocode

The Pseudocode for implementing the functionality of the Parent Page is:

Algorithm 3: Implementing Parent Page

Inputs:

using Button: *View Child Location*

using Button: *View in Maps*

password;

View Child Location:

```
s martcontract.getCoordinates() ;
latitude = smartcontract.latitude;
longitude = smartcontract.longitude;
lat = EncryptionDecryption.decryptAES (latitude, password);
lon = EncryptionDecryption.decryptAES (longitude, password);
Text(lat + "&" + lon);
```

View in Maps:

```
googleMapsUrl =
        "https://www.google.com/maps/search/?api=1&query=$latitude,$longitude";
appleMapsUrl =
        "https://maps.apple.com/?q=$latitude,$longitude";
if  canLaunch(googleMapsUrl)
        launch(googleMapsUrl) ;
else if  canLaunch(appleMapsUrl)
```

launch(*appleMapsUrl*) ;

else

Text("Couldn't launch Apple Maps");

3.5.5 Developing Model

A model is developed to link the mobile application and smart contract. As discussed in section 3.5.1.2, the libraries used to connect with the smart contract are http and web3dart.

3.5.5.1 Pseudocode

The Pseudocode for establishing a connection with the smart contract is:

Algorithm 3: Linking Flutter with Smart Contract

Inputs: *latitude, longitude;*

Initialization:

httpClient = Client();

client = Web3Client("**#InfuraEndPoint or Infura API Key**", *httpClient*);

abi = rootBundle.loadString("**#abi-path**");

contractAddress = "**#contractaddress**";

credentials = EthPrivateKey.fromHex ("**#Private Key of MetaMask Account**");

contract = DeployedContract (ContractAbi.fromJson(*abi*,**#contractname**),

EthereumAddress.fromHex(*contractAddress*)

);

Functions to interact with smart contract:

addCoordinates():

```
client.sendTransaction (
                    credentials,
                    Transaction.callContract (
                                        contract: contract,
                                        function: sendCoordinates,
                                        parameters: [latitude, longitude],
                                        ),
                    chainId: 4,
                    );
```

readCoordinates():

```
readCoordinates = client.call (
                                contract: contract,
                                function: readCoordinates,
                                params: []
                                );
latitude = readCoordinates[0] ;
longitude = readCoordinates[1];
```

- *httpClient* and *Web3Client* are used to make a connection with the smart contract.
- *abi* is the ABI of the smart contract, which is saved in the "*assets/abi.json*" directory. It is used to access functions present in the smart contract.

- *contractAddress* is the address of the deployed smart contract.
- *credentials* is the private key of the MetaMask Account linked with the smart contract. It is used for authentication.
- *DeployedContract* loads the contract which was deployed in the Ethereum blockchain. It is assigned to a temporary variable "*contract*." The temporary variable is used to make calls or transactions with the smart contract.
- *client.sendTransaction*, sends the data (in this case, latitude and longitude) to the smart contract's addCoordinates function, which in turn adds the data to the blockchain. "*chainId: 4*" denotes the serial id of the Rinkeby network in the MetaMask wallet.
- client.call, makes a call to the smart contract, which in turn uses the readCoordinates function to view the data present in the blockchain.

3.5.6 Output in Terminal

The output of the application can also be viewed in the terminal of the IDE. In the terminal, the flow of the data can be seen in detail as shown below:

Child's Side:

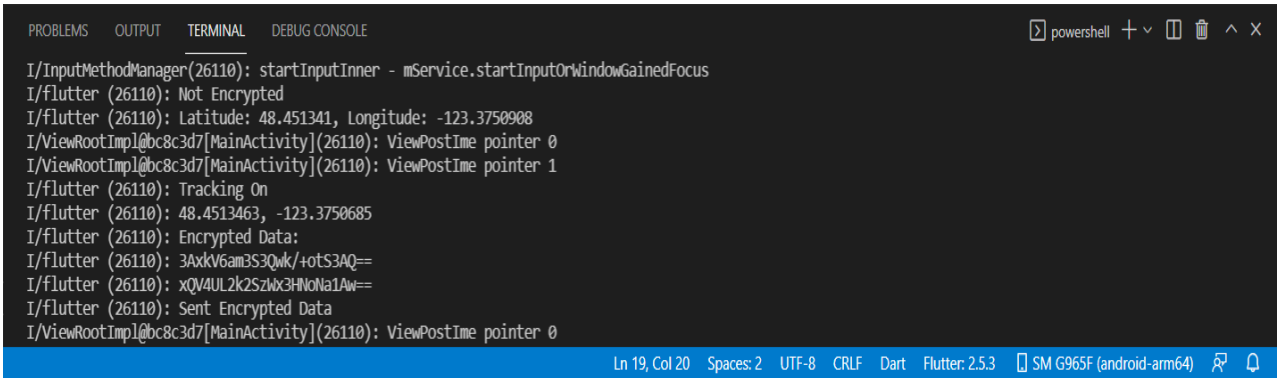


Figure 3.16: Output in the terminal when Child turns on the location tracking

From the above image, first, it shows the coordinates. Then, before sending the coordinates to the blockchain, the data is encrypted. Thus, the encrypted data for latitude and longitude as per the images is: *'3AxkV6am3S3Qwk/+otS3AQ==*' and *'xQV4UL2k2SzWx3HNoNa1Aw=='*.

Parent's Side:

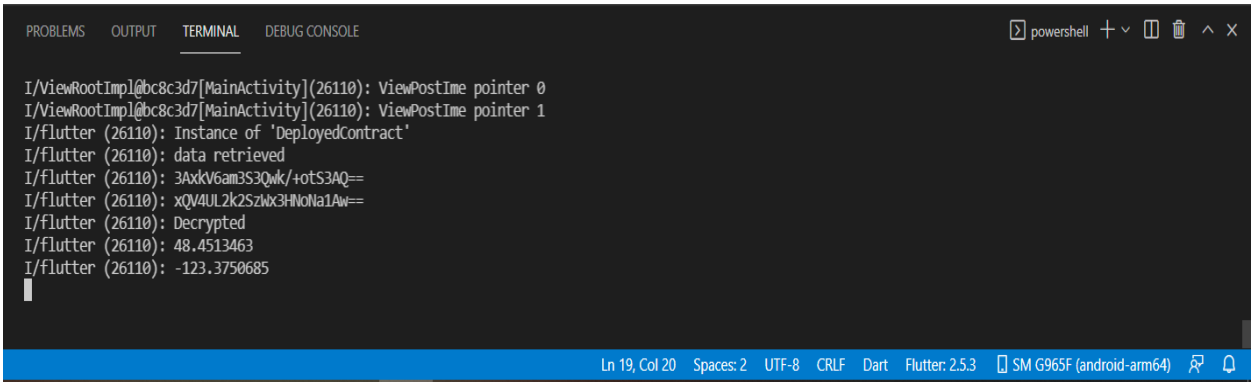


Figure 3.17: Output in the terminal when Parent reads data from the blockchain

From the above image, the data retrieved from the blockchain is encrypted. - then, with the use of the secret key, the data is decrypted into coordinates.

3.6 Etherscan

Etherscan is a blockchain explorer for Ethereum network. It allows the user to check, track, view transactions, blocks, smart contracts and other on-chain data.

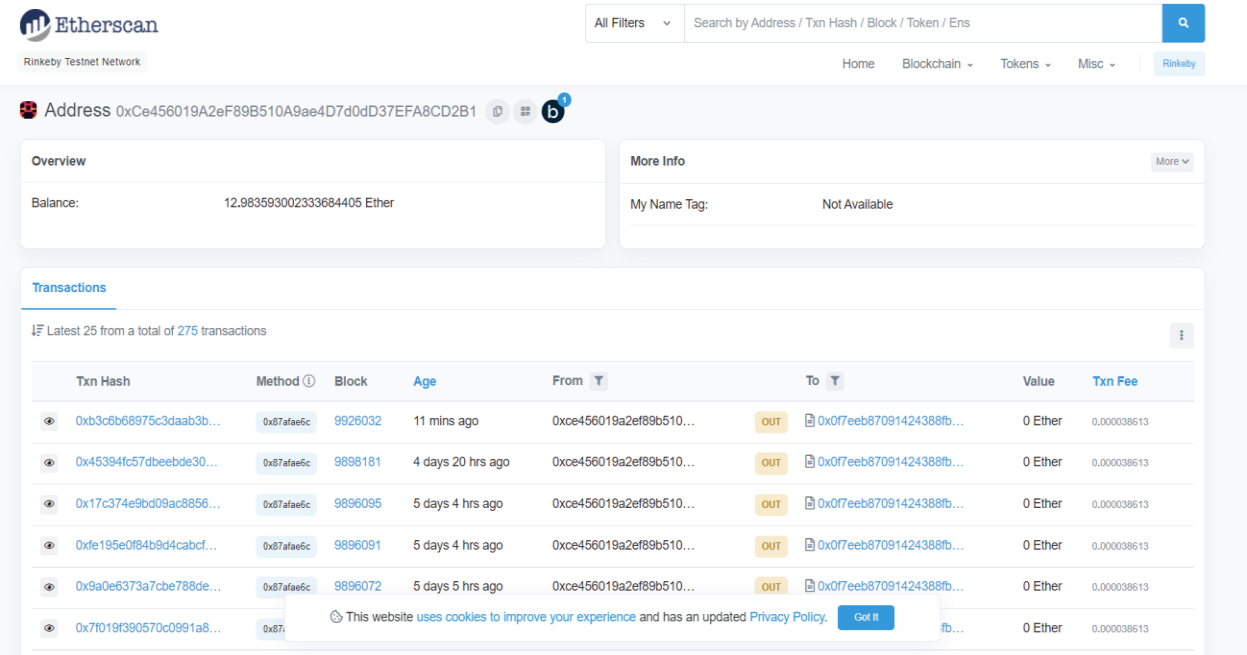


Figure 3.18: Etherscan Homepage

The above image displays the transactions going through the MetaMask wallet's account address. It shows the latest wallet balance, latest transaction etc.,

Further information can be viewed if the latest transaction is selected from the above list.

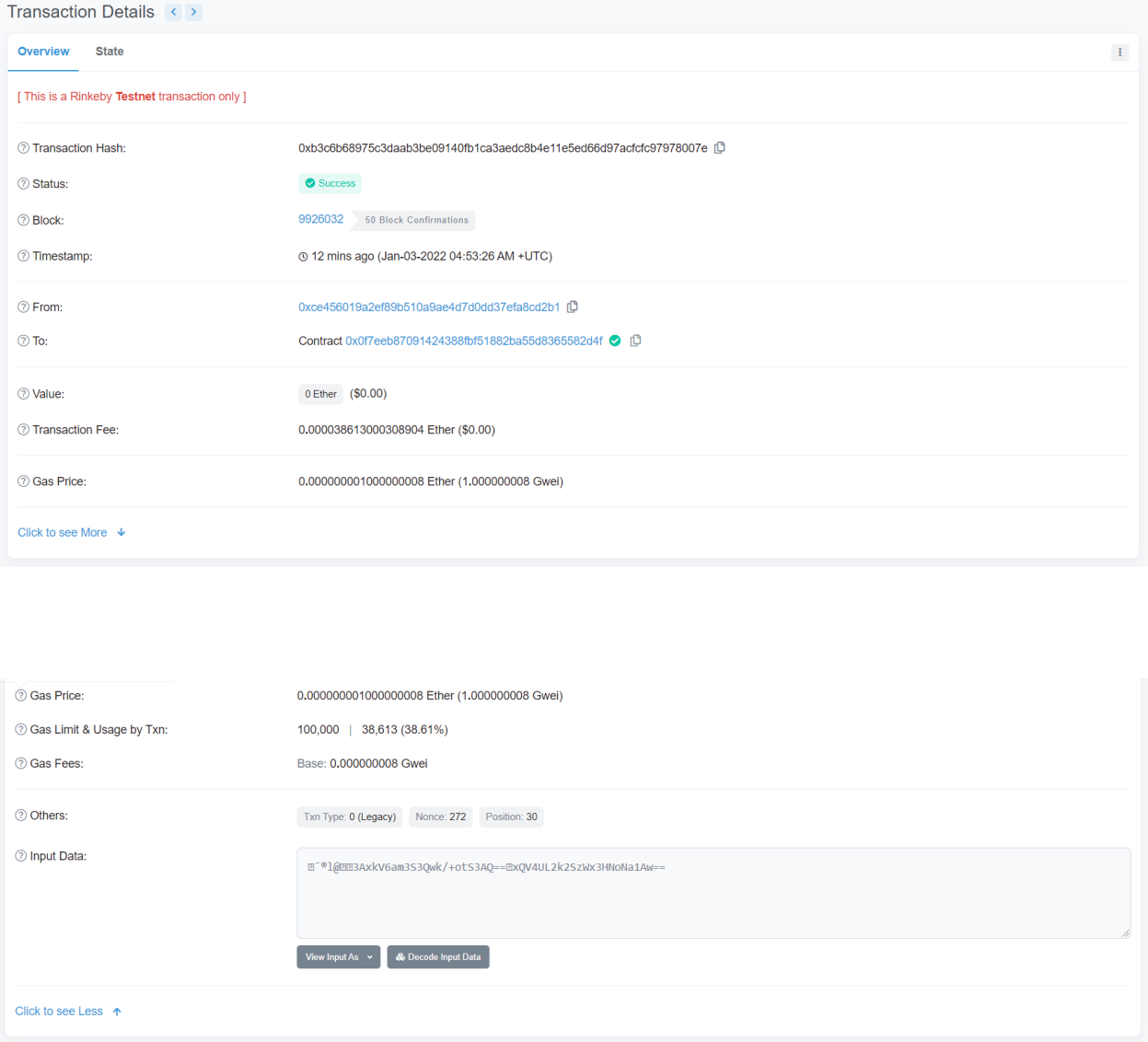


Figure 3.19: Transaction details in Etherscan

The above images show that the transaction occurred on the Rinkeby test network. It also shows a few important parameters like transaction hash, block number, timestamp, Gas, To and From addresses. At the bottom, it also displays the data that is present on the blockchain. This proves that the data saved in the Ethereum blockchain is encrypted and secure.

3.6.1 Etherscan transactions when “Track Location” is turned on:

The application is tested in real-time multiple times. The below figure shows the location updates sent to the blockchain when the Child moved for a distance of 800 meters.

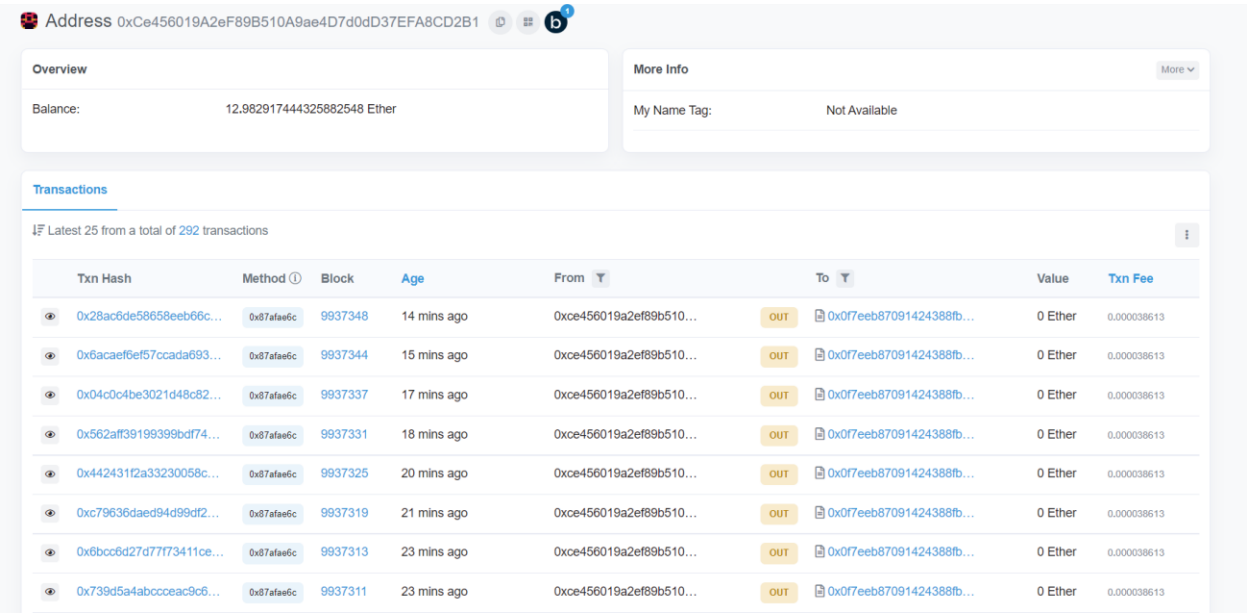


Figure 3.20: New transactions are recorded when Track Location is Enabled

In the above figure, eight transactions got recorded over a distance of 800 meters (1 transaction per 100 meters). The below table shows the values of the coordinates present in those transactions.

Distance (in meters)	Coordinates
100	Latitude: 48.4515805 Longitude: -123.3749278
200	Latitude: 48.4507439 Longitude: -123.374427
300	Latitude: 48.4507177 Longitude: -123.3744531
400	Latitude: 48.4498741 Longitude: -123.3739816

500	Latitude: 48.440552 Longitude: -123.3734093
600	Latitude: 48.4482182 Longitude: -123.3728984
700	Latitude: 48.4473973 Longitude: -123.3723228
800	Latitude: 48.4465531 Longitude: -123.3718532

Table 3.2: Location Updates over a distance of 800 meters

Note: The screenshots of mobile dApp when “Track Location” is enabled are shown in the appendix of this report.

Chapter 4: Cost and Performance Analysis

Different costs are associated with deploying a smart contract and sending or storing data on the blockchain. They are:

4.1 Cost for Smart Contract Deployment

The costs of deploying the smart contract to the Ethereum blockchain depend on multiple factors. The cost to deploy a smart contract is calculated in terms of Gas units. As Gas is the total amount of computational effort required to execute a smart contract, it varies depending on the number of functions, storage variables etc.,

The smart contract is deployed on the Rinkeby test network in this project. A smart contract can be deployed and tested on a test network before deploying it on the main network. The costs incurred for both networks are similar, but in the main network, real Ether is associated.

In section 3.3, the Gas values and the total Ether took to deploy the contract are shown using images. The below information shows the total cost in USD.

Total Gas = 382432

Total Gas Fee (or Total Cost) = 0.00095608 ETH

= 3.60 USD

4.2 Cost per Transaction

In this project, the data of the GPS coordinates is sent in the form of a Transaction to the smart contract. The smart contract then stores the data in the Ethereum Blockchain network. A Gas fee is charged whenever a new transaction is made, and this Gas fee is paid to the miner.

Section 3.6 shows various transactions visible in Etherscan and the Gas associated with each transaction. The Gas fee or Transaction fee that is charged for adding a transaction is:

Gas per Transaction = 38613

Gas Fee or Transaction Fee = 0.00003861300038613 ETH

= 0.15 USD

4.3 Cost incurred when Location Updates in enabled

In this application, if the Child turns on the “Track Location” feature, the dApp sends the GPS coordinates to the blockchain if the Child crosses a certain distance (by default, it is 100 meters). And the dApp will continue to send the coordinates until the “Track Location” is turned off. Thus, when the data is sent every time to the blockchain, a new transaction is recorded.

In section 3.6.1, the dApp was tested in real-time. When the Child travelled a distance of 800 meters and the “Track Location” is enabled for that distance and location updates are sent, then

Total number of Location Updates = 8 (800m = 8*100m)

Total number of Transactions = 8

Total Gas = 8*38613

= 308904

Total Cost for that distance = 0.000308904 ETH

= 1.18 USD

The above calculation can be verified using figure 3.20. It shows eight transactions where each transaction has a fee of “0.000038613 ETH”.

The user can also modify the code to send the coordinates after regular intervals.

4.4 Comparison with Google Map and Apple Maps

Although the cost associated with using dApp is relatively higher than the most used location tracking applications – Google Maps and Apple Maps, few features of decentralized applications outweigh the cost factor.

The below table provides the comparison of the developed dApp with existing Google or Apple Maps.

Features	**Google Maps and Apple Maps**	**dApp**
Authority	Centralized	Decentralized
Trusted third party	Yes	No
Data Privacy	No	Yes
Data Mutability	Mutable – control is with the central authority	Immutable
Cost	Most features are free	A small fee is associated
Code Visibility	No – underlying code cannot be viewed by the public	Yes – the smart contract is visible to the public, but it cannot be altered

Table 4.1: Comparison of dApp with Google Maps and Apple Maps

Chapter 5: Conclusion and Future Work

The proposed decentralized application is successfully developed, and the corresponding smart contract is deployed on the Rinkeby test network. The features such as Get Current Location, Track Location, View in Maps, Sending Coordinates to the smart contract and retrieving coordinates from the smart contract are explained in detail in previous chapters. The data that flows through the smart contract and wallet is visible in Etherscan.

Although the proposed dApp meets all the primary requirements, it can be further developed by adding many useful features. This application introduces the scenario where there is a single parent and a single Child. Still, this feature can be developed on a large scale by introducing a database used to authenticate multiple users. And also, the cryptocurrency wallet can be added as a feature to the dApp. This will allow the user to check the balance, monitor transactions etc.,

The Blockchain network used along with this application is Ethereum, and the cryptocurrency used is Ether. However, real-time costs associated with this cryptocurrency will possibly increase in future. Thus, an alternate blockchain and alternate cryptocurrency can be used or developed.

Bibliography

[1] Razi, A., Hardy, L., Roehrer, E., Yeom, S., & Kang, B. H, “GPSPiChain-Blockchain based self-contained family security system in smart home”

[2] Johnson, M., Jones, M., Shervey, M., Dudley, J. T., & Zimmerman, N. (2019), “Building a secure biomedical data sharing decentralised app (Dapp): Tutorial”, Journal of Medical Internet Research

[3] Calastry Ramesh, Vinay Kumar, "Storing IOT Data Securely in a Private Ethereum Blockchain" (2019). UNLV Theses, Dissertations, Professional Papers, and Capstones. 3582. http://dx.doi.org/10.34917/15778410

[4] de Camargo Silva, Lucas & Samaniego, Mayra & Deters, Ralph. (2019). “IoT and Blockchain for Smart Locks”. 0262-0269. 10.1109/IEMCON.2019.8936140.

[5] National Institute of Standards and Technology. (2001). Advanced encryption standard (AES). National Institute of Standards and Technology.

[6] Smith, J. (2020, January 1). Blockchain tutorial: Learn blockchain technology (examples). Guru99. https://www.guru99.com/blockchain-tutorial.html

[7] Share your real-time location with others. (n.d.). Google.Com. Retrieved January 17, 2022, from https://support.google.com/maps/answer/7326816?hl=en&co=GENIE.Platform%3DAndroid

[8] Find friends and share your location with Find My. (2021, January 7). Apple Support. https://support.apple.com/en-ca/HT210514

[9] What is Ethereum? (n.d.). Ethereum.Org. Retrieved January 17, 2022, from https://Ethereum.org/en/what-is-Ethereum/

[10] Flutter tutorials. https://docs.flutter.dev/reference/tutorials

[11] A crypto wallet & gateway to blockchain apps. (n.d.). MetaMask.io. Retrieved January 17, 2022, from https://MetaMask.io/

[12] Remix - Ethereum IDE. (n.d.). Ethereum.Org. Retrieved January 17, 2022, from https://remix.Ethereum.org/

Appendix A: Smart Contract

```
pragma solidity ^0.5.16;

contract GPSTracker {

    string[2] coordinates = ["0.00", "0.00"];

    function sendCoordinates(string memory _latitude, string memory _longitude)
public{
        coordinates[0] = _latitude;
        coordinates[1] = _longitude;
    }

    function readCoordinates() view public returns(string memory, string memory)
{
        return (coordinates[0], coordinates[1]);
    }
}
```

Appendix B: Contract Application Binary Interface (ABI)

This ABI must be saved in the “assets/abi.json” file of the project directory.

```
[
    {
        "constant": true,
        "inputs": [],
        "name": "readCoordinates",
        "outputs": [
            {
                "internalType": "string",
                "name": "",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "",
                "type": "string"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
            {
                "internalType": "string",
                "name": "_latitude",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "_longitude",
                "type": "string"
            }
        ],
        "name": "sendCoordinates",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    }
]
```

Appendix C: Dependencies and Permissions

```yaml
dependencies:
  flutter:
    sdk: Flutter

  geolocator: ^5.1.5
  http: ^0.13.4
  url_launcher: ^6.0.17
  location: ^4.3.0
  web3dart: ^2.3.3
  provider: ^6.0.1
  encrypt: ^5.0.1
```

```xml
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"
/>
   <queries>
        <intent>
            <action android:name="android.intent.action.VIEW" />
            <data android:scheme="https" />
        </intent>
        <intent>
            <action android:name="android.intent.action.DIAL" />
            <data android:scheme="tel" />
        </intent>
        <intent>
            <action android:name="android.intent.action.SEND" />
            <data android:mimeType="*/*" />
        </intent>
    </queries>
```

Appendix D: Home Page

```dart
import 'package:flutter/material.dart';
import 'package:project/ParentPage.dart';
import 'package:project/parentModel.dart';
import 'package:project/childModel.dart';
import 'package:provider/provider.dart';
import 'package:project/ChildPage.dart';

var password;

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(
          create: (context) => parentModel(),
        ),
        ChangeNotifierProvider(
          create: (context) => childModel(),
        )
      ],
      child: MaterialApp(
        title: 'GPS Tracker Demo',
        theme: ThemeData(
          primarySwatch: Colors.blue,
        ),
        home: const MyHomePage(title: 'Demo'),
        debugShowCheckedModeBanner: false,
      ),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;
```

```
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final TextEditingController _passwordController = TextEditingController();
  int _value = 1;
  String error = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              "dApp\nGPS Tracker",
              style: TextStyle(
                fontSize: 50,
                color: Colors.indigo,
              ),
              textAlign: TextAlign.center,
            ),
            SizedBox(
              height: 20.0,
            ),
            Padding(
              padding: const EdgeInsets.only(
                left: 50,
                right: 50,
                bottom: 10,
              ),
              child: DropdownButton(
                value: _value,
                items: [
                  DropdownMenuItem<int>(
                    child: Text("Parent"),
                    value: 1,
                  ),
                  DropdownMenuItem(
                    child: Text("Child"),
                    value: 2,
                  )
                ],
                onChanged: (value) {
                  setState(() {
                    _value = value as int;
```

```
            });
          },
        ),
      ),
      Padding(
        padding: const EdgeInsets.only(
          left: 50,
          right: 50,
          bottom: 10,
        ),
        child: TextFormField(
          controller: _passwordController,
          obscureText: true,
          style: const TextStyle(
            color: Colors.black,
            fontSize: 22,
          ),
          cursorColor: Colors.black,
          decoration: const InputDecoration(
            hintText: 'Input Password',
            border: UnderlineInputBorder(
              borderSide: BorderSide(color: Colors.black),
            ),
            focusedBorder: UnderlineInputBorder(
              borderSide: BorderSide(color: Colors.black),
            ),
          ),
        ),
      ),
      Text("(Password must be 16 Characters)"),
      SizedBox(
        height: 20.0,
      ),
      Container(
        width: MediaQuery.of(context).size.width / 1.3,
        height: 50,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(35.0),
          color: Colors.amber,
        ),
        child: MaterialButton(
          onPressed: () {
            if (_passwordController.text.length == 16) {
              if (_value == 1) {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => const ParentPage(),
```

```
                      ),
                    );
                  } else if (_value == 2) {
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => const ChildPage(),
                      ),
                    );
                  } else {
                    Text("Invalid User");
                  }
                  //print('Password: ' + _passwordController.text);
                  print('Secret Key: s***************');
                  password = _passwordController.text;
                } else {
                  setState(() {
                    error =
                        "Invalid length: \nPlease input 16 character secret
password";
                  });
                }
              },
              child: const Text('Login'),
            ),
          ),
          SizedBox(
            height: 20.0,
          ),
          Text(
            error,
            style: TextStyle(color: Colors.red[900]),
            textAlign: TextAlign.center,
          ),
          SizedBox(
            height: 20.0,
          ),
        ],
      ),
    ),
  );
 }
}
```

Appendix E: Child Page and Child Model

Child Page Code:

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:provider/provider.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:project/childModel.dart';

class ChildPage extends StatefulWidget {
  const ChildPage({Key? key}) : super(key: key);

  @override
  _ChildPageState createState() => _ChildPageState();
}

class _ChildPageState extends State<ChildPage> {
  var locationMessage = "";

  String lat_1 = "0.00";
  String lon_1 = "0.00";

  String status = "not sending";

  bool isSwitched = false;
  int i = 0;

  late StreamSubscription<Position> positionStream;

  void stopUpdate() {
    positionStream.cancel();
  }

  @override
  Widget build(BuildContext context) {
    var listModel = Provider.of<childModel>(context);
    return Scaffold(
      appBar: AppBar(
        title: Text("My Location"),
      ),
      body: Center(
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Icon(
```

```
            Icons.location_on,
            size: 46.0,
            color: Colors.blue,
          ),
          SizedBox(
            height: 10.0,
          ),
          Text(
            "Get User Location",
            style: TextStyle(fontSize: 26.0, fontWeight: FontWeight.bold),
          ),
          SizedBox(
            height: 20.0,
          ),
          Text("Lask Known Location:\n" + locationMessage),
          SizedBox(
            height: 20.0,
          ),
          Container(
            width: MediaQuery.of(context).size.width / 1.3,
            height: 50,
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(35.0),
              color: Colors.amber,
            ),
            child: MaterialButton(
              onPressed: () async {
                var position = await Geolocator().getCurrentPosition(
                    desiredAccuracy: LocationAccuracy.high);
                var lat = position.latitude;
                var lon = position.longitude;

                lat_1 = lat.toString();
                lon_1 = lon.toString();

                print("Not Encrypted \nLatitude: $lat, Longitude: $lon");

                setState(() {
                  locationMessage = "Latitude: $lat_1\nLongitude: $lon_1";
                });
              },
              child: Text('Get Current Location'),
            ),
          ),
          SizedBox(
            height: 30.0,
          ),
          Text(
```

```
                "Track Location",
                style: TextStyle(fontSize: 26.0, fontWeight: FontWeight.bold),
              ),
              SizedBox(
                height: 5.0,
              ),
              Switch(
                value: isSwitched,
                onChanged: (value) {
                  setState(() {
                    if (value) {
                      // final LocationSettings locationSettings =
LocationSettings(
                      //   accuracy: LocationAccuracy.high,
                      //   distanceFilter: 100,
                      // );
                      var geolocator = Geolocator();
                      var locationOptions = LocationOptions(
                        accuracy: LocationAccuracy.high,
                        distanceFilter: 100,
                      );
                      // ignore: unused_local_variable

                      positionStream =
                          geolocator.getPositionStream(locationOptions).listen(
                        (Position position) {
                          setState(
                            () {
                              print("Tracking On");
                              value = true;
                              isSwitched = true;
                              print(position == null
                                  ? 'Unknown'
                                  : position.latitude.toString() +
                                      ', ' +
                                      position.longitude.toString());

                              lat_1 = position.latitude.toString();
                              lon_1 = position.longitude.toString();

                              listModel.addCoordinates(
                                lat_1,
                                lon_1,
                              );
                              status = "Latitude: $lat_1\nLongitude: $lon_1";
                              locationMessage =
                                  "Latitude: $lat_1\nLongitude: $lon_1";
                            },
```

```
                        );
                      },
                    );
                  } else {
                    print("Tracking Off");
                    value = false;
                    isSwitched = false;
                    stopUpdate();
                  }
                });
              },
              activeTrackColor: Colors.lightGreenAccent,
              activeColor: Colors.green,
            ),
            SizedBox(
              height: 20.0,
            ),
            listModel.isLoading ? Text("No Data") : Text(status),
            SizedBox(
              height: 100.0,
            ),
            Container(
              width: MediaQuery.of(context).size.width / 1.3,
              height: 50,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(35.0),
                color: Colors.amber,
              ),
              child: MaterialButton(
                onPressed: () async {
                  final String googleMapsUrl =
                      "https://www.google.com/maps/search/?api=1&query=$lat_1,$
lon_1";
                  final String appleMapsUrl =
                      "https://maps.apple.com/?q=$lat_1,$lon_1";

                  if (await canLaunch(googleMapsUrl)) {
                    await launch(googleMapsUrl);
                  }
                  if (await canLaunch(appleMapsUrl)) {
                    await launch(appleMapsUrl, forceSafariVC: false);
                  } else {
                    throw "Couldn't launch URL";
                  }
                },
                child: Text('View in Maps'),
              ),
            ),
```

```
            ]),
      ),
    );
  }
}
```

Child Model:

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/services.dart';
import "package:flutter/widgets.dart";
import 'package:project/Encrypt-Decrypt.dart';
import 'package:http/http.dart';
import 'package:web3dart/web3dart.dart';
import 'package:encrypt/encrypt.dart';
import 'main.dart' as pass;

class childModel extends ChangeNotifier {
  bool isLoading = true;

  late Client _httpClient;

  late String _contractAddress;
  late String _abi;

  late Web3Client _client;
  late EthPrivateKey _credentials;

  late DeployedContract _contract;

  late String x;
  late String y;
  late String latitude;
  late String longitude;

  late ContractFunction _readCoordinates;
  late ContractFunction _sendCoordinates;

  childModel() {
    initiateSetup();
  }

  Future<void> initiateSetup() async {
    _httpClient = Client();
    _client = Web3Client(
        "https://rinkeby.infura.io/v3/b1c2f7fc53144772a4d6dd43879b2fd7",
        _httpClient);
```

```
  await getAbi();
  await getCredentials();
  await getDeployedContract();
}

Future<void> getAbi() async {
  _abi = await rootBundle.loadString("assets/abi.json");
  _contractAddress = "0x0f7eeb87091424388fbF51882bA55D8365582D4f";

  //print(_abi);
  //print(_contractAddress);
}

Future<void> getCredentials() async {
  _credentials = EthPrivateKey.fromHex(
      "e3cd24eb42110f5e460bfb03b7fef6c92d8058c7d9f0367bc0eb35f39ec02442");
  //print(_credentials);
}

Future<void> getDeployedContract() async {
  _contract = DeployedContract(ContractAbi.fromJson(_abi, "GPSTracker"),
      EthereumAddress.fromHex(_contractAddress));
  _readCoordinates = _contract.function("readCoordinates");
  _sendCoordinates = _contract.function("sendCoordinates");
}

getCoordinates() async {
  List readCoordinates = await _client
      .call(contract: _contract, function: _readCoordinates, params: []);
  x = readCoordinates[0];
  y = readCoordinates[1];
  //print(x);
  //print(y);
}

addCoordinates(String lat, String lon) async {
  //print(lat + "received");
  //print(lon + "received");

  //print(pass.password);

  latitude = EncryptionDecryption.encryptAES(lat);
  longitude = EncryptionDecryption.encryptAES(lon);

  await _client.sendTransaction(
    _credentials,
    Transaction.callContract(
```

```
        contract: _contract,
        function: _sendCoordinates,
        parameters: [latitude, longitude],
        maxGas: 100000,
      ),
      chainId: 4,
    );
    //print("Sent Coordinates");
    print("Encrypted Data:");
    print(latitude);
    print(longitude);
    print("Sent Encrypted Data");

    getCoordinates();
    isLoading = false;
    notifyListeners();
  }
}
```

Appendix F: Parent Page and Parent Model

Parent Page:

```
import 'package:flutter/material.dart';
import 'package:project/parentModel.dart';
import 'package:provider/provider.dart';
import 'package:url_launcher/url_launcher.dart';

class ParentPage extends StatefulWidget {
  const ParentPage({Key? key}) : super(key: key);

  @override
  _ParentPageState createState() => _ParentPageState();
}

class _ParentPageState extends State<ParentPage> {
  String locationMessage = "No data yet";

  String lat_1 = "0.00";
  String lon_1 = "0.00";

  @override
  Widget build(BuildContext context) {
    var listModel = Provider.of<parentModel>(context);
    return Scaffold(
      body: Center(
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              Container(
                width: MediaQuery.of(context).size.width / 1.3,
                height: 50,
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(35.0),
                  color: Colors.amber,
                ),
                child: MaterialButton(
                  onPressed: () {
                    listModel.getCoordinates();
                    lat_1 = listModel.latitude;
                    lon_1 = listModel.longitude;
                    //print('hi' + lat_1);
                    //print('hi' + lon_1);
                    setState(() {
                      locationMessage = "Latitude: $lat_1\nLongitude: $lon_1";
                    });
```

```
                    Text(locationMessage);
                },
                child: Text('View Child Location'),
              ),
            ),
            SizedBox(
              height: 20.0,
            ),
            listModel.isLoading
                ? Text(
                    "Child Last known Location: \nNo Data",
                    textAlign: TextAlign.center,
                  )
                : Text(
                    "Coordinates:\n" + locationMessage,
                    textAlign: TextAlign.center,
                  ),
            SizedBox(
              height: 20.0,
            ),
            Container(
              width: MediaQuery.of(context).size.width / 1.3,
              height: 50,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(35.0),
                color: Colors.amber,
              ),
              child: MaterialButton(
                onPressed: () async {
                  if (lat_1 == "0.00" && lon_1 == "0.00") {
                    print("No data yet");
                  }
                  final String googleMapsUrl =
                      "https://www.google.com/maps/search/?api=1&query=$lat_1,$
lon_1";
                  final String appleMapsUrl =
                      "https://maps.apple.com/?q=$lat_1,$lon_1";

                  if (await canLaunch(googleMapsUrl)) {
                    await launch(googleMapsUrl);
                  }
                  if (await canLaunch(appleMapsUrl)) {
                    await launch(appleMapsUrl, forceSafariVC: false);
                  } else {
                    throw "Couldn't launch URL";
                  }
                },
                child: Text('View in Maps'),
```

```
                ),
              ),
            ]),
      ),
    );
  }
}
```

Parent Model:

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/services.dart';
import "package:flutter/widgets.dart";
import 'package:project/Encrypt-Decrypt.dart';
import 'package:http/http.dart';
import 'package:web3dart/web3dart.dart';
import 'main.dart' as pass;

class parentModel extends ChangeNotifier {
  bool isLoading = true;

  late Client _httpClient;

  late String _contractAddress;
  late String _abi;

  late Web3Client _client;
  // ignore: unused_field
  late Credentials _credentials;

  late String x;
  late String y;
  late String latitude;
  late String longitude;

  late DeployedContract _contract;

  late ContractFunction _readCoordinates;

  Future<void> initiateSetup() async {
    _httpClient = Client();
    _client = Web3Client(
        "https://rinkeby.infura.io/v3/b1c2f7fc53144772a4d6dd43879b2fd7",
        _httpClient);
    await getAbi();
    await getCredentials();
    await getDeployedContract();
```

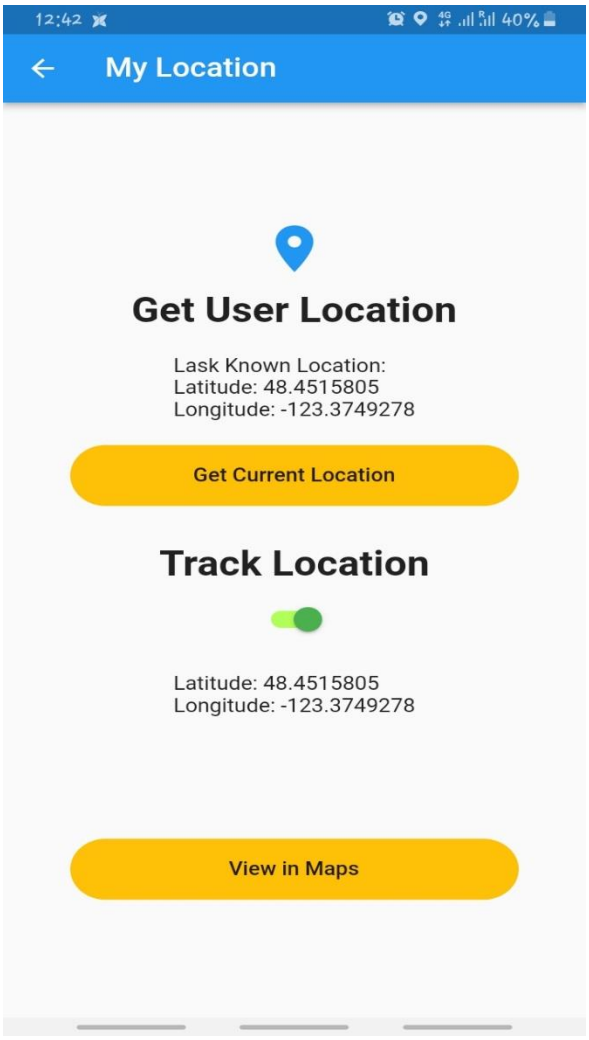
```
  }

  Future<void> getAbi() async {
    _abi = await rootBundle.loadString("assets/abi.json");
    _contractAddress = "0x0f7eeb87091424388fbF51882bA55D8365582D4f";

    //print(_abi);
    //print(_contractAddress);
  }

  Future<void> getCredentials() async {
    _credentials = EthPrivateKey.fromHex(
        "e3cd24eb42110f5e460bfb03b7fef6c92d8058c7d9f0367bc0eb35f39ec02442");
    //print(_credentials);
  }

  Future<void> getDeployedContract() async {
    _contract = DeployedContract(ContractAbi.fromJson(_abi, "GPSTracker"),
        EthereumAddress.fromHex(_contractAddress));
    _readCoordinates = _contract.function("readCoordinates");
    print(_contract);
    //print("stopped here");
  }

  getCoordinates() async {
    initiateSetup();

    List readCoordinates = await _client
        .call(contract: _contract, function: _readCoordinates, params: []);

    x = readCoordinates[0];
    y = readCoordinates[1];
    print("data retrieved");
    print(x);
    print(y);
    //print(pass.password);

    latitude = EncryptionDecryption.decryptAES(x);
    longitude = EncryptionDecryption.decryptAES(y);

    print("Decrypted");
    print(latitude);
    print(longitude);

    isLoading = false;
    notifyListeners();
  }
}
```

Appendix G: Encryption and Decryption

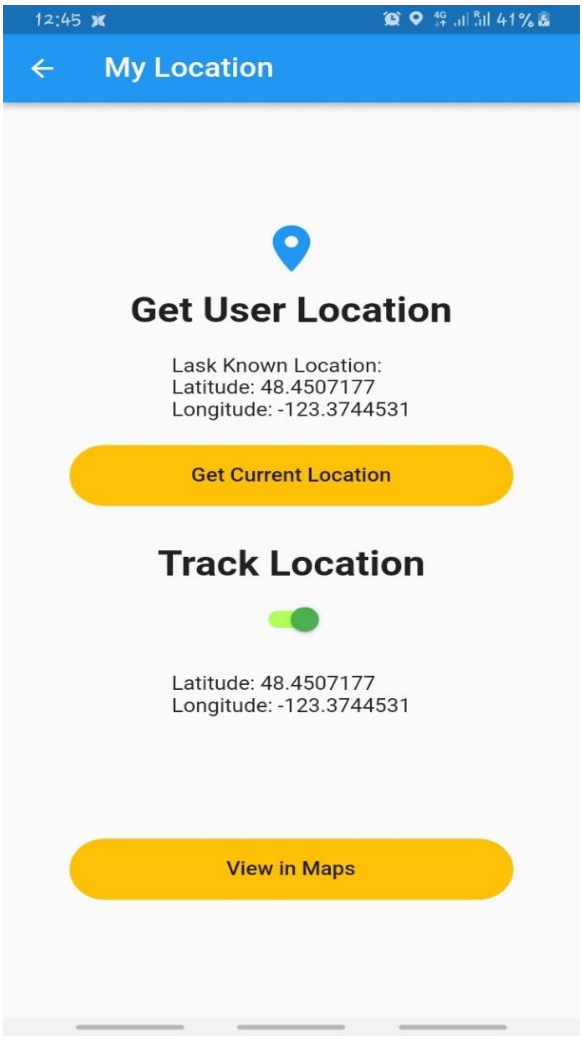
```dart
import 'package:encrypt/encrypt.dart';
import 'package:project/main.dart' as pass;

class EncryptionDecryption {
  static final Encrypter encrypter =
      Encrypter(AES(Key.fromUtf8(pass.password)));
  static final iv = IV.fromLength(16);

  static String encryptAES(String text) {
    return encrypter.encrypt(text, iv: iv).base64;
  }

  static String decryptAES(String encryptedText) {
    final encrypted = Encrypted.fromBase64(encryptedText);
    return encrypter.decrypt(encrypted, iv: iv);
  }
}
```

Appendix H: Screenshots when “Track Location” is enabled

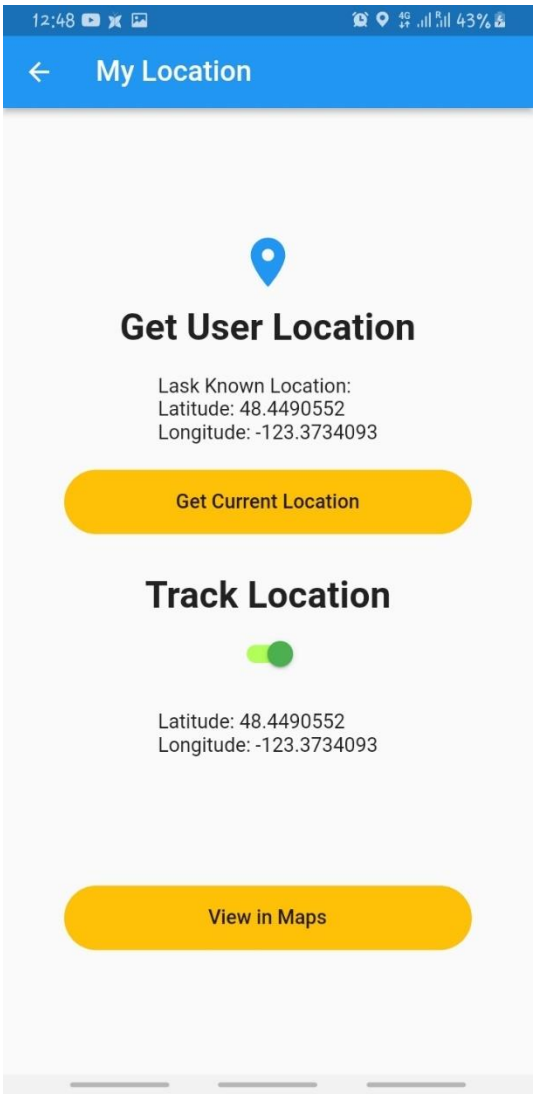
The below screenshots show the coordinates displayed in the application when the Child moves over a distance of 800 meters.



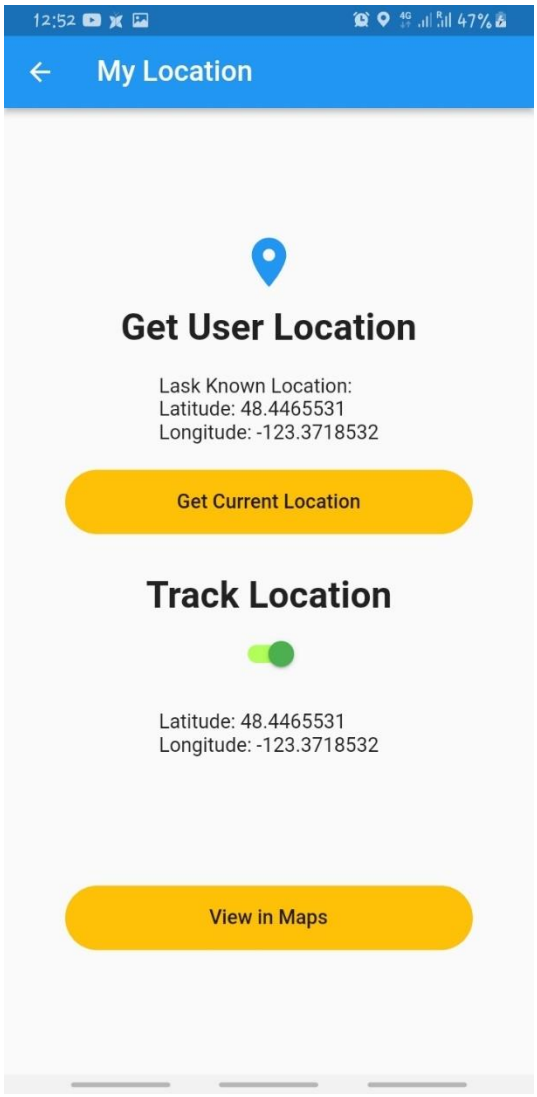
100 meters



300 meters



500 meters



800 meters