COMP390

2023/24

Integrating Secure Multi-Party Computation (SMPC) in a Crypto Wallet

Student Name:	Jesan Akter
Student ID:	201496637
Supervisor Name:	Alexei Lisitsa

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Acknowledgments

I want to express my deep appreciation to my family for their unwavering support and encouragement throughout my academic journey. Their love and belief in me have been a source of immense strength, and I feel truly fortunate to have them by my side.

Additionally, I am grateful to my friends who have been there for me every step of the way. Their consistent reminders and assistance have made a profound impact on my university experience, and I am thankful for their unwavering friendship.

I dedicate this dissertation to Allah, the Most Gracious and Most Merciful, who granted me the inspiration and stamina to complete this humble work. This small contribution is merely a drop for His boundless munificence.

Abstract

This dissertation details the development and implementation of a secure cryptocurrency wallet application leveraging Secure Multi-Party Computation (SMPC). The primary goal of the project was to enhance the security of private keys while maintaining a seamless user experience.

The SMPC wallet is designed to split the user's private key into multiple shares then stored separately, significantly reducing the risk of key exposure. Key functionalities include user registration, login, wallet creation, and token transactions on the Sepolia ETH network via a public API. The system architecture includes a Dart-based frontend for a dynamic and responsive user interface, a backend utilizing PHP and SQL for user authentication and database management, and an additional backend in Python for blockchain interactions via the Moralis API.

During the development process, various challenges were encountered, such as integrating SMPC with string-based private keys and maintaining secure storage. Despite these obstacles, the project successfully demonstrates a practical application of SMPC in enhancing cryptocurrency wallet security.

Statement of Ethical Compliance

Since my project retrieves data from the Alchemy and Moralis APIs—which do not contain any personally identifiable information—it falls under Data Category B.
Because human participants are only used for the evaluation process, it also falls under Category 2 of Human Participation.

I confirm that I followed the university's ethical guidance throughout my project. This included treating human participants fairly and processing data in accordance with current regulations. My project adhered to the Code of Conduct of the Chartered Institute for IT, ensuring low-risk activities with aggregated, anonymous human data.
All data presented in this project is genuine, obtained legally, and analysed originally. No covert research or financial inducements were involved, and all permissions for data access were appropriately secured.

Contents

Introduction & Background

Project Description

The aim of this project is to develop a robust and secure crypto [1] wallet application leveraging Secure Multi-Party Computation (SMPC) [2], requiring both the user and the service provider for accessing the wallet. It should offer features enabling users to securely sign up, log in, manage their wallets, view balances, and send transactions functionalities.

The system architecture comprises three main components: frontend, backend, and database, each serving a specific role.
The frontend interface developed using Dart [3], prioritizes user experience with dynamic interfaces, seamless interactions, and responsive layouts. This approach ensures enhanced user engagement and satisfaction. For the backend, PHP handles user registration and login verification, while Python integrates with the Moralis API [4]- extracts data from the blockchain- for checking wallet balances. The utilization of PHP and Python enables efficient handling of various backend tasks, catering to different aspects of the application's functionality. A SQL database is employed to organize user information systematically, reducing redundancy and optimizing query performance.

Additionally, the private key generated is split into two shares, with one share provided to the user and the other stored in the database. Upon successful authentication, the user's share is combined with the database share, ensuring secure access to the wallet.

I encountered and overcome many obstacles during my development. These involved picking up a lot of new technologies and abilities on my own; I'll talk about these difficulties in the sections on design and implementation.

Aims & Objectives

These are the aims and objectives I initially set out for my project:
Aims:

A. Develop an authentication process.
B. Develop the functionality to create or import a wallet for users.
C. Create a mechanism for splitting and reconstructing the private key.
D. Enable secure token transactions and balance checking for users.
E. Design an intuitive and responsive user interface.

Objectives:

1. Design and implement frontend pages.
2. Develop a login process.
3. Develop a sign-up process.
4. Implement an input validation for email.
5. Integrate an API for blockchain interaction.
6. Design and optimize SQL database structure.
7. Design backend APIs for user authentication.
8. Generate a unique wallet address for user upon account creation.

9. Implement a backup for the shares of private key.
10. Implement Shamir Secret Sharing [5] for private key splitting and reconstructing.
11. Develop intuitive and responsive frontend user interface.
12. Conduct user testing for feedback.

Background

As a computer science enthusiast passionate about crypto, I decided to delve into the world of decentralized finance and set up my own crypto wallet using MetaMask [6]. Excited to begin my journey, I diligently followed the setup process and stored my mnemonic phrase [7] – a crucial piece of information needed to access my wallet- on my laptop. One fateful day, an attacker gained unauthorised access to my laptop through a malware in my email [8] and managed to steal my mnemonic phrase. With this key piece of information in hand, the attacker swiftly infiltrated your crypto wallet and drained the entire account of its funds, leaving me with nothing but a sense of betrayal and loss. According to BBC [9],in 2021 criminals funnelled $8.6 billion of cryptocurrency through various channels, with malware operators being one of the conduits.

The events bring to light the built-in flaws in conventional cryptocurrency wallets, whose reliance on centralised storage techniques and single points of failure can have disastrous effects on users [10]. While the allure of full control and decentralization may initially seem empowering, they also carry several serious risks that are readily exploited by bad actors.

There are some advantages to the conventional wallet such as offer users full control over their assets, empowering them with the principles of decentralization. With no central authority to prevent or censor their keys, users have the freedom to transact as they see fit [11]. Additionally, traditional wallets are not reliant on centralized organizations, mitigating the risk of financial loss in the event of bankruptcy.

However, users find it difficult to use traditional crypto wallets, particularly if they are unfamiliar with the nuances of blockchain technology. Users are primarily responsible for maintaining mnemonic phrases and making sure they are secure; they might find it difficult to locate a safe place to store them that guards against theft or loss [12]. Furthermore, the use of user devices for wallet management creates new security risks because self-custody wallets, such as Ledger [13] and MetaMask, are vulnerable to social engineering and phishing scams.

We talked about the user side, now let’s shift our focus on what if we left the funds to an exchange. On the positive side, exchanges usually have stronger security protocols and easily accessible customer service, which frequently helps with account recovery and password resets. But it's crucial to understand that not every exchange is made equal and that there may be dangers including fraud or bankruptcy, such as the FTX collapse [14]. Furthermore, exchanges depart from the fundamental decentralisation tenet of cryptocurrencies because they are centralised organisations. Furthermore, users should be aware that they might still be responsible for paying taxes on any cryptocurrency holdings even in the case of an exchange failure or bankruptcy [15].

When addressing security concerns in the context of our system, it's crucial to clarify the specific threats we aim to mitigate. Firstly, we're not primarily focused on scenarios where users intentionally attempt to defraud themselves or where service providers seek to pilfer from their own customers. Instead, our primary concern lies in safeguarding against external threats, particularly those stemming from malware infections on users' devices, breaches within service providers' systems, or the actions of malicious insiders. These risks pose significant challenges to the integrity and security of the system, as they can potentially compromise sensitive user data or facilitate unauthorized access to accounts and assets.

Over the years, there have been various implementations of different wallet types.
Below are just some common ones:

1. Zero-Knowledge Proof Wallet [16]: Provide high privacy and anonymity by verifying transactions without revealing sensitive details.
2. Multi-Signature Wallet [17]: Enhance security by requiring multiple cryptographic signatures to authorize transactions.
3. Time-lock Wallet [18]: Allow users to schedule transactions for future execution, offering convenience and automation options.

In the late 1970s, Secure Multi-Party Computation (SMPC) first appeared as a theoretical concept in research. Its development over the years was aided by many research papers that laid the foundation for useful applications. These days, SMPC is a lively, actively researched field with many real-world applications, rather than just a theoretical endeavour [19].

SMPC protocol behaves like a trusted black box- parties provide input and receive. The requirements for SMPC involve multiple parties with private inputs jointly computing a function. This process must ensure two crucial properties: privacy and correctness. Privacy ensures that nothing, but the final result is revealed to any party involved, safeguarding sensitive input data. Meanwhile, correctness ensures that the computed result is accurate and consistent with the intended computation, despite potential malicious behaviour from some participants [20]. These properties are fundamental for maintaining trust and integrity in multi-party computation protocols, especially in scenarios where confidentiality and accuracy are paramount.

My solution is self-custody with the experience of an exchange where key is shared between user and the service provider. Service provider can’t transact without user and malware on device don’t enable key theft as both don’t hold the key but a share.
The process involves taking a private key ‘k’ and splitting it into two random shares, ‘k_1’ and ‘k_2’, ensuring that their sum equals the original key ‘k_1 + k_2 = k’. Each share is then distributed, with one share given to the user and the other to the service provider. When a user attempts to access their wallet by combining their private key share, an additional layer of security is implemented. Before the key is retrieved from the database and combined with the user's input, authentication with the service provider is required. This authentication step ensures that only authorized users can access the private key, enhancing the overall security of the process. This method provides robust security guarantees against malicious adversaries, even if they have full control over one of the

devices. The attacker's inability to access both shares simultaneously ensures that they cannot learn anything about the key, as each share on its own holds no value. Additionally, the protocol guarantees correctness by preventing attackers from accessing the wallet through a different application. For an attacker to gain any useful information, they would need to compromise both devices, ensuring a high level of security and protection against unauthorized access [21].

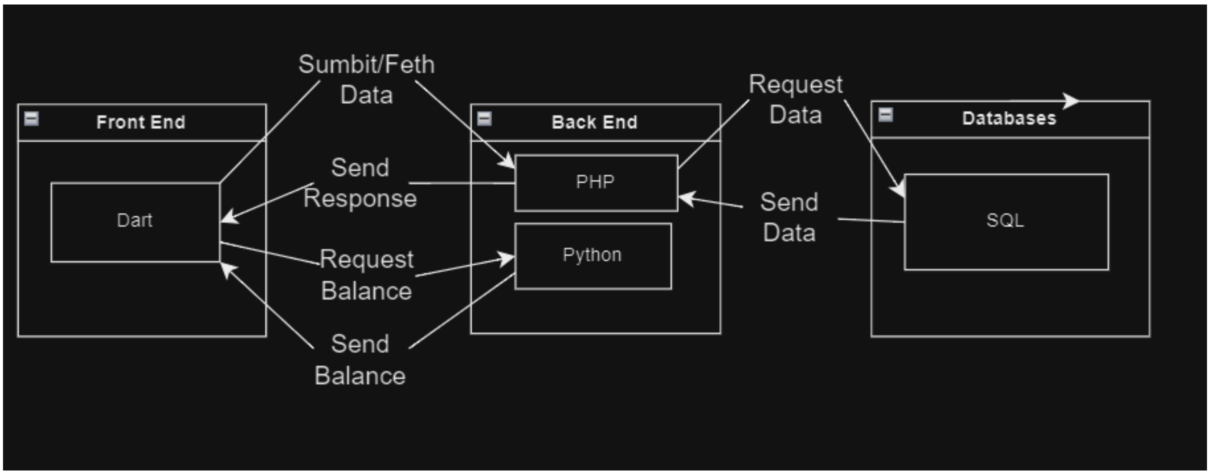
However, there are some disadvantages to my method:

1. Single Point of Failure: Despite the private key being split into shares, a single point of failure persists if either the user's device or the service provider's device is compromised. In the event of a compromise at the service provider's end, the user would be unable to access their wallet since the complete private key is required for access. As the user only possesses one share of the private key, access is contingent upon the service provider's key share. Thus, compromise of either device jeopardizes the security of the wallet.
2. Recovery Challenges: In the event of device loss or failure, recovering the private key may become complicated. Users must ensure they have access to both key shares and the necessary recovery mechanisms, adding another layer of complexity to the process.
3. Trust in Service Provider: Users must trust the service provider to securely store and manage their key share. If the provider experiences a breach or acts maliciously, it could compromise the security of the user's assets.
4. Limited Accessibility: Splitting private keys may limit the accessibility of funds, particularly in scenarios where quick access to the full key is necessary. Users may face challenges accessing their assets if their key share becomes inaccessible or if the service provider is unavailable.

In conclusion, the proposed method offers a robust defence against unauthorized access, particularly in scenarios where a user's device is compromised. With the additional key share held by the service provider, even if an attacker gains access to the user's device, they cannot access the wallet without both shares. This effectively prevents the unauthorized use of stolen mnemonic phrases to drain wallet funds. By implementing this method, users can significantly bolster the security of their crypto wallets, providing greater peace of mind and protection against potential attacks.

Design

System Architecture and Components



The frontend, backend, and database are the three separate parts of my system that are depicted in the diagram. I can schedule the implementation of each component and determine when to do so by using this diagram, which makes the dependencies evident.

The following provides an explanation of each technology utilised in each component:

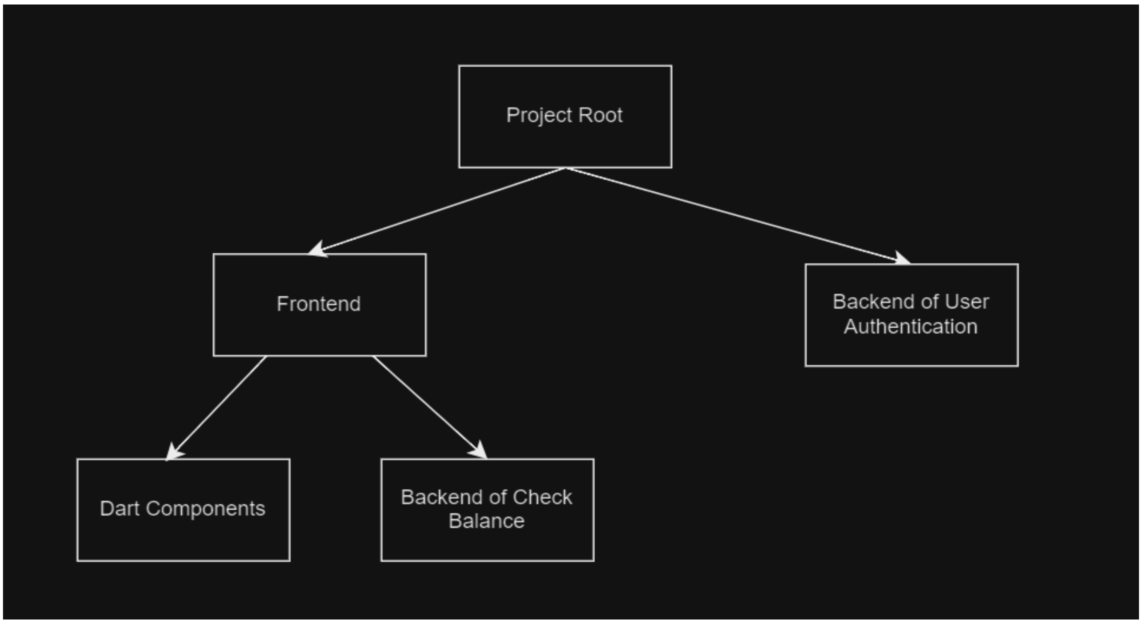
- Dart:
  - I chose Dart as the language for the Flutter [22] platform, where its renowned for its versatility and efficiency, emerged as the language of choice for constructing the frontend interface. Dart's concise syntax and comprehensive tooling ecosystem facilitated rapid development cycles, which helped me to iterate and refine the frontend components with ease. Moreover, Dart's compatibility with Flutter—a widely adopted framework for cross-platform app development—fostered the potential for future scalability and expansion across diverse platforms, ensuring broad accessibility and usability for end-users. A lot of packages from Flutter were utilized in the development process, facilitating accelerated development, robust functionality, and cross-platform compatibility. These packages provided pre-built solutions for common development tasks, ranging from UI components to complex cryptographic operations and integrations. Leveraging these packages not only helped me in the implementation of features but also ensured quality assurance, customization flexibility, and updates from the Flutter ecosystem.

- PHP:
  - PHP was chosen as the primary language for its robustness, scalability, and widespread adoption in web development. PHP's extensive ecosystem of libraries and frameworks, coupled with its compatibility with various web servers and operating systems, makes it a popular choice for building dynamic and secure web applications. With its built-in support for database integration and web services, PHP enables seamless interaction with SQL

databases, facilitating efficient data storage, retrieval, and manipulation. Additionally, PHP's simplicity and ease of use expedite the development process, allowing for rapid prototyping and iteration. By employing PHP for the backend, the application benefits from a reliable and flexible infrastructure, capable of handling data processing tasks with ease.

- Python
    - I chose Python due to its simplicity, versatility, and robust ecosystem. Allows rapid development and easy maintenance of backend systems. It has an extensive library, which includes frameworks such as Flask. Python's scalability and compatibility with the Moralis made it an ideal choice for building backend systems that interact with blockchain services. Also, it is a high-level language and is very readable, making debugging and coding a lot easier.
- SQL Database
    - I chose a SQL database primarily for its robust querying capabilities, especially its ability to join tables and retrieve data efficiently. This allows me to structure and manage the data in a relational manner, facilitating complex queries and analyses essential for a crypto wallet application. Additionally, my previous experience with SQL in a side project I did has provided me with a solid foundation in database management, making SQL a familiar and comfortable choice for me.

I plan to arrange my files into different directories, one for the frontend which contains all the dart components as well as the backend of check balance, the other one which is the backend of the user authentication.



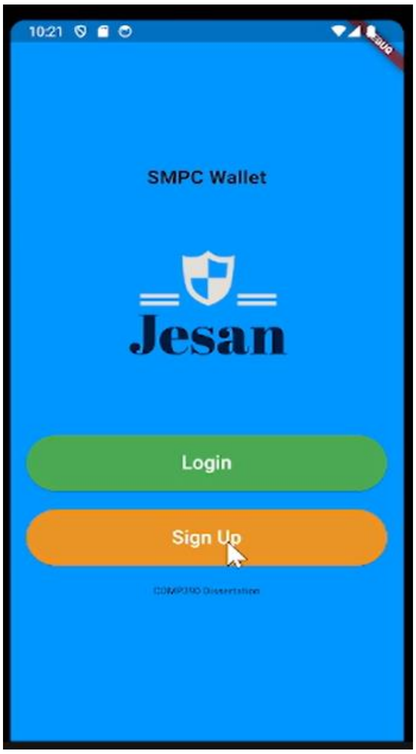
Blockchain Selection

Sepolia [23] Ethereum Network serves as the perfect development environment for several reasons. Firstly, it offers a sandboxed platform where I can experiment with my crypto wallet application without directly interacting with the main Ethereum network. This isolation ensures that any errors or issues encountered during development do not impact the integrity or security of the main network. Additionally, Sepolia Ethereum Network allows me to test various functionalities and transactions within a controlled environment, providing valuable insights into the behaviour of the application without risking real funds. Moreover, acquiring Sepolia ETH from faucets provides a convenient way to obtain tokens for testing purposes. These faucets typically distribute tokens to users who complete certain tasks or activities, such as mining or participating in specific challenges. By receiving Sepolia ETH from faucets, I can simulate real-world scenarios and test the functionality of my crypto wallet with actual tokens, enhancing the authenticity and effectiveness of the testing process.

Frontend Design

The first step for my project is to design the frontend for my crypto wallet. I have chosen dart because of its robust features and a reactive framework for building modern web and mobile applications made it an ideal candidate for implementing a seamless and responsive user experience, which will interact with the backend. The frontend will consist of the following pages: Home, Login, Sign Up, Wallet created, Import wallet, Wallet, Send tokens.

UI/UX Design:



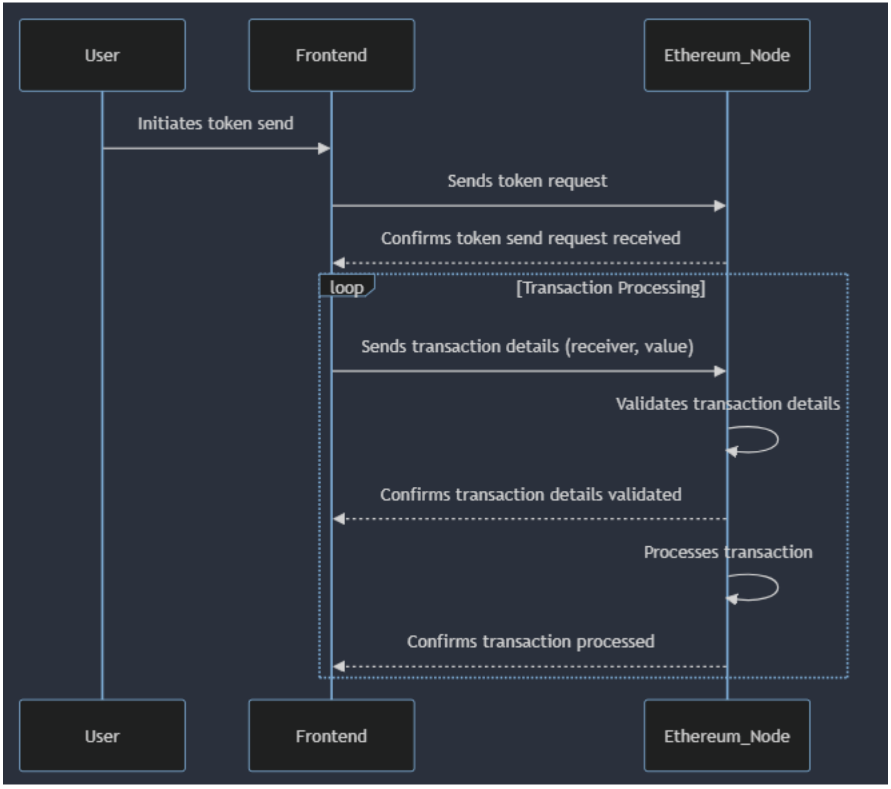
The focus is on essential actions: logging in and signing up.
Blue is the background colour choice and the wallet name ("SMPC Wallet") convey a sense of trust and security, aligning with typical banking and financial app conventions.
The absence of distracting elements and unnecessary features helps users focus on their login or signup tasks without distractions.
The central alignment of the wallet name and the logo enhances visual appeal and usability.
The "Login" and "Sign Up" buttons are prominently displayed, making it easy for users to initiate their desired action.
The colours green and orange are contrasting making them easily distinguishable and clickable.

The other UI/UX designs of the other pages can be found in the Appendix.
The UI flow of my application is also illustrated in the Appendices. This demonstrates how a typical user will use my app and navigate to each page.

Alchemy API

When designing the system to send tokens, the frontend will directly a send request to the Alchemy API, that will include details such as amount and receiver address to the Ethereum node provided by Alchemy. After the transaction has been completed, Ethereum node confirms to the frontend that the transaction has been completed.

Send tokens sequence diagram:

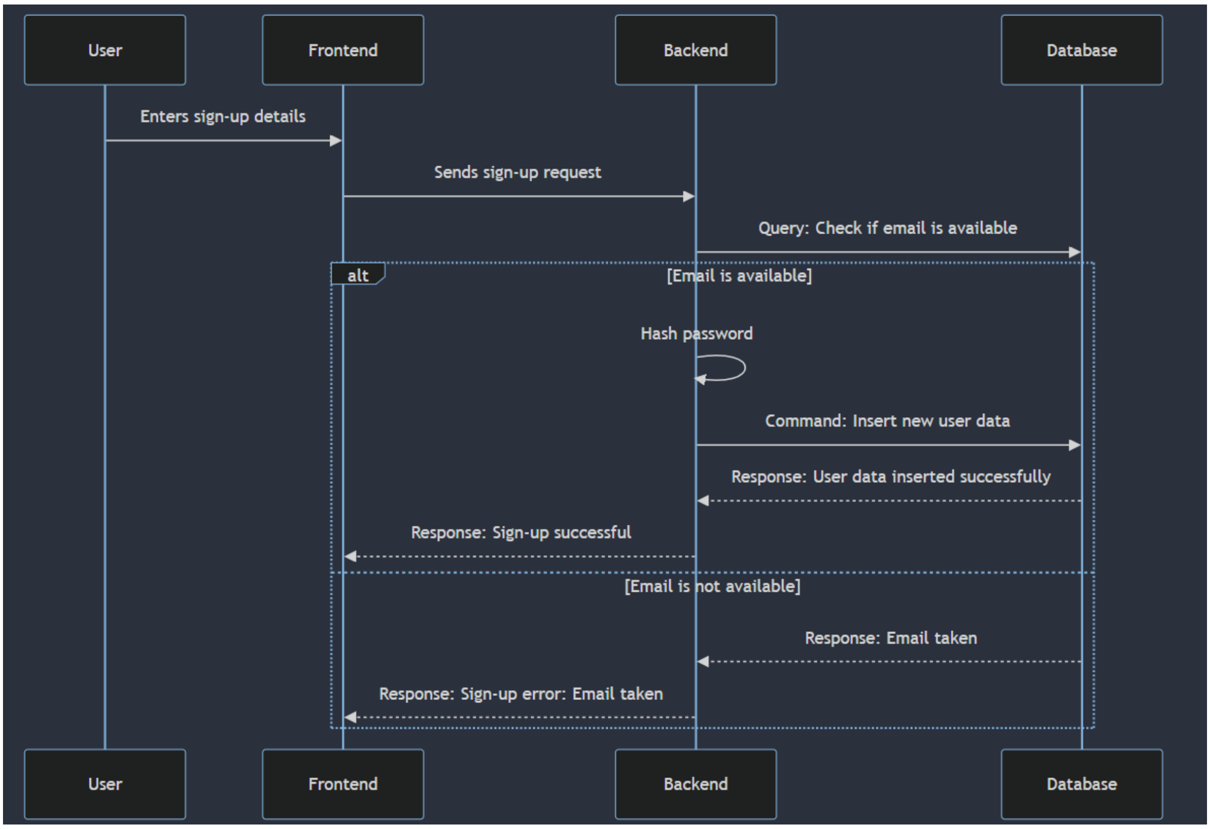

User
Frontend
Ethereum_Node
Initiates token send
Sends token request
Confirms token send request received
loop
[Transaction Processing]
Sends transaction details (receiver, value)
Validates transaction details
Confirms transaction details validated
Processes transaction
Confirms transaction processed
User
Frontend
Ethereum_Node

Backend Design

I'll design both the backend APIs that will have a set of endpoints that correspond to the functionalities required by the frontend. Each function of the backend will correspond to an endpoint. For example:

- User Authentication – handles POST requests containing email and password from the frontend, confirming if they match with the database before returning true or false.
- Check balance – handles GET requests containing wallet provider and address from the frontend and returns the balance.

I have created diagrams for one example of each category of interaction. The sequence diagram below illustrates how the backend endpoint functions when a user clicks the "sign up" button. The diagram below falls under User Authentication:

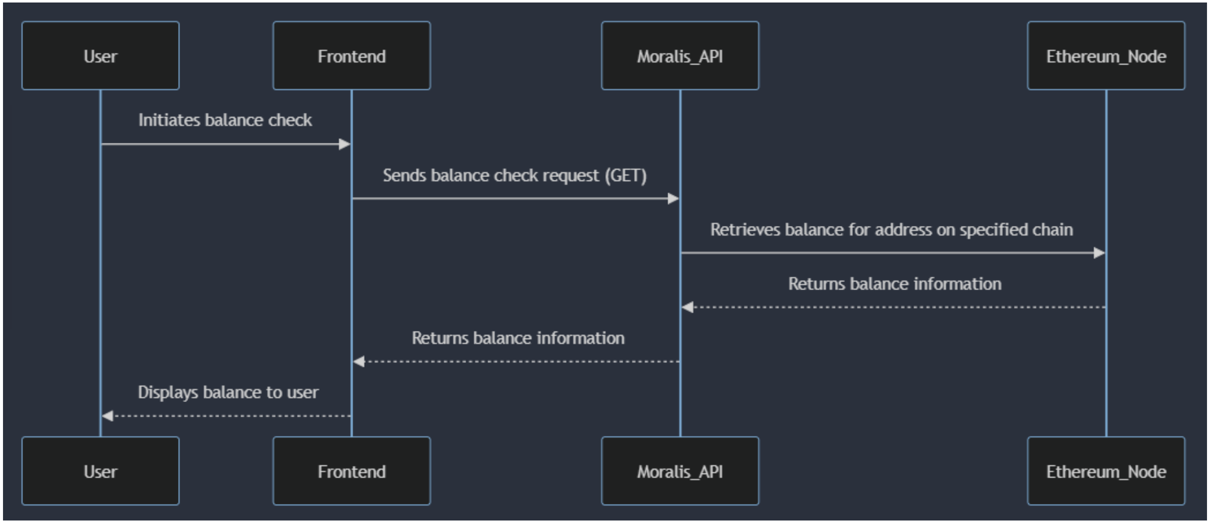


It first verifies that the email address is not already in use, if it is it hashes the password and stores the new user information before sending the appropriate response to the front end. There will be loads more that are similar the above that fall under the same category, these can be seen in the Appendix.
For the user authentication API, I'll be coding it in PHP because it facilitates seamless connections to the SQL database and enables the creation of endpoints that the frontend can utilize. PHP's compatibility with SQL databases streamlines database operations, ensuring efficient data handling. Additionally, SQL queries will be utilized to insert, retrieve, and manipulate user data within the database, ensuring optimal data organization and management. These SQL queries will be executed within the PHP backend, allowing for dynamic interaction with the database based on user requests and application logic.

Moralis API

The next part of my project is to design a backend way to check the balance of the user's wallet. For this, I have chosen the Moralis to implement my check balance.
Moralis API will be coded in python language and will be in the same directory as frontend code. I choose Moralis API as it offers a streamlined approach to extracting data from the blockchain, providing a straightforward method for accessing token balances and other relevant information. This simplifies the development process and saves valuable time and resources. Additionally, Moralis API is known for its exceptional speed, being touted as one of the fastest in the market. This ensures that token balance information can be retrieved quickly and efficiently, enhancing the responsiveness of the crypto wallet application.

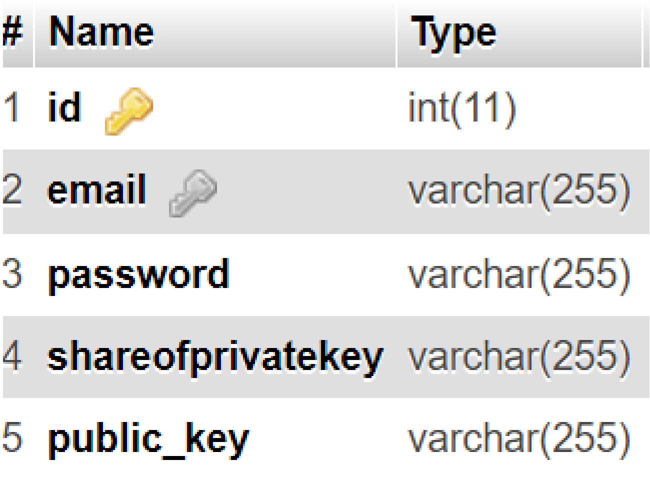
Below is a sequence diagram for “Check Balance”:


User
Frontend
Moralis_API
Ethereum_Node
Initiates balance check
Sends balance check request (GET)
Retrieves balance for address on specified chain
Returns balance information
Returns balance information
Displays balance to user
User
Frontend
Moralis_API
Ethereum_Node

When user “initiates balance check”, it means that it either refreshes the page, logs in or signs up. The application sends a GET request to get the token balance to the Moralis API. The Moralis API retrieves token balances by interacting with the Ethereum blockchain and it communicates with Ethereum nodes to retrieve blockchain data such as token balances.

Database Design

The database is the next element to be designed. All the data will be kept in a database, which will be arranged and maintained using a single table that is intended to reduce redundancy and improve query performance.
I created the table in my database using SQL statements. The layout of the table is as follows:



#	Name	Type
1	id 🔑	int(11)
2	email 🔑	varchar(255)
3	password	varchar(255)
4	shareofprivatekey	varchar(255)
5	public_key	varchar(255)

Since it is crucial to shield user credentials from unauthorised access in the event of a data breach, the password will be hashed before being stored to increase security. Hashing renders passwords irreversibly into fixed-length character strings, rendering it computationally impossible for adversaries to decipher the original passwords. In the event of a compromise, this practice helps protect user privacy and stops unauthorised access to other accounts.

I have chosen hashing over encryption because it permanently converts passwords into fixed-length strings of characters and is computationally difficult to reverse. On the other hand, encryption provides stronger security because it is reversible and could be exploited by attackers. Additionally, during user authentication, their input password undergoes hashing and is then compared to the stored hashed password. This technique accomplishes verification efficiently without requiring reversible decryption.

Splitting the Key

I choose Shamir's Secret Sharing to split keys because of its high level of security and flexibility. This method divides a secret into multiple shares, distributing them among different parties (the user and service provider). With Shamir's Secret Sharing, the original secret can only be reconstructed when a predefined number of shares are combined, ensuring that no single party can access the complete secret. This approach enhances security by reducing the risk of a single point of failure or compromise. Additionally, Shamir's Secret Sharing offers customization options, allowing me to specify the number of shares required for reconstruction and the threshold for access. This flexibility enables tailored security policies, providing robust protection for sensitive information like private keys in a crypto wallet.

Implementation

Development Environment and Setup

I made the decision to use my IDE, VS Code [24]. I selected this IDE over others because of its integrated Git support, which greatly streamlines version control, and its flexible extensions, which let me write code in Dart, Python, and PHP.

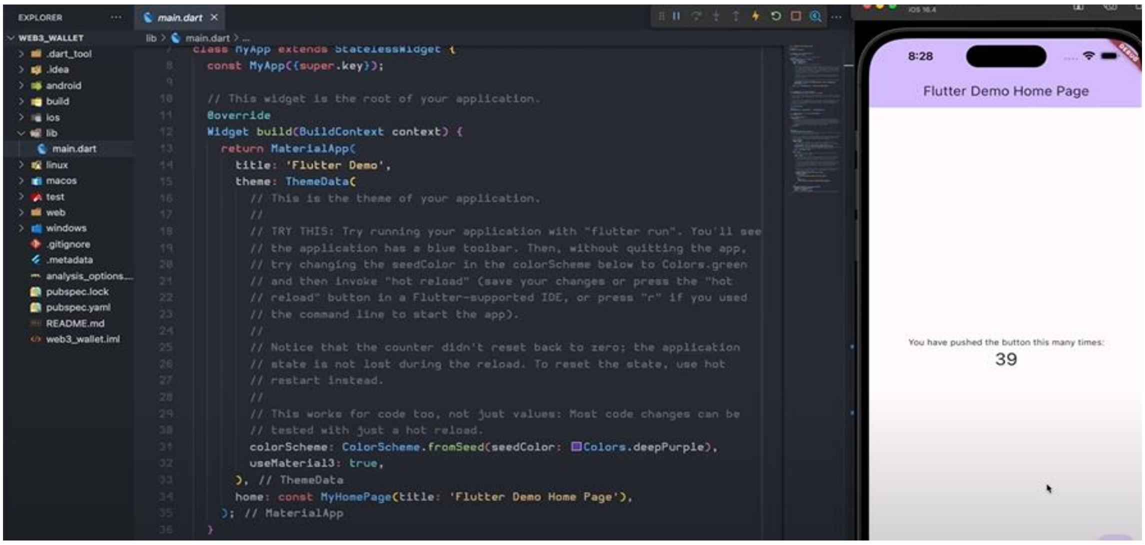
My SQL server is hosted by XAMPP [25] , which provides a lightweight, dependable, and easy-to-use method for me to quickly start a local server. Because it was so simple to set up, I was able to concentrate less on server management and more on the more crucial elements of my backend, like database design and interactions.

I used GitHub for source control because it provides me with a platform for allows me to see any conflicts when merging branches clearly and managing code Iterations.

I registered for both Alchemy and Moralis an account to get my API keys that provides a reliable and efficient infrastructure for interacting with the Ethereum blockchain. This involved me using my email address and creating a password for both services.

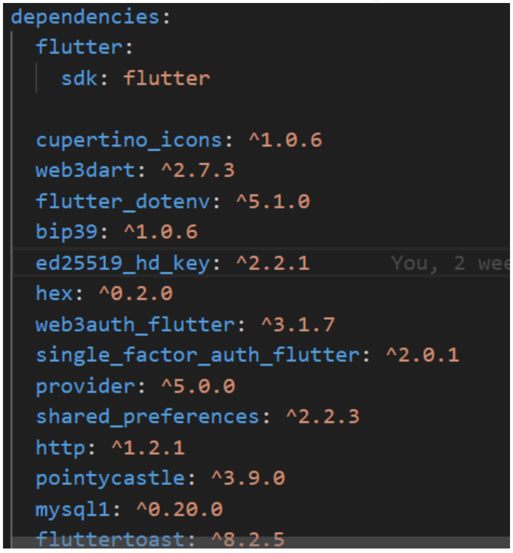
Frontend Development

I first had to set up Flutter environment within VS Code. To do this I initiated a new Flutter project using the command “flutter create” command, which created the base structure for a demo application, setting up the initial directories, configuration files etc. Below is an image of the demo application:



```dart
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // TRY THIS: Try running your application with "flutter run". You'll see
        // the application has a blue toolbar. Then, without quitting the app,
        // try changing the seedColor in the colorScheme below to Colors.green
        // and then invoke "hot reload" (save your changes or press the "hot
        // reload" button in a Flutter-supported IDE, or press "r" if you used
        // the command line to start the app).
        //
        // Notice that the counter didn't reset back to zero; the application
        // state is not lost during the reload. To reset the state, use hot
        // restart instead.
        //
        // This works for code too, not just values: Most code changes can be
        // tested with just a hot reload.
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
```

Throughout the implementation of my crypto wallet, I used a lot of packages from Flutter, which were added in a configuration file “pubsec.yaml”:



```yaml
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.6
  web3dart: ^2.7.3
  flutter_dotenv: ^5.1.0
  bip39: ^1.0.6
  ed25519_hd_key: ^2.2.1
  hex: ^0.2.0
  web3auth_flutter: ^3.1.7
  single_factor_auth_flutter: ^2.0.1
  provider: ^5.0.0
  shared_preferences: ^2.2.3
  http: ^1.2.1
  pointycastle: ^3.9.0
  mysql1: ^0.20.0
  fluttertoast: ^8.2.5
```

- cupertino_icons [26] -This is an asset repo containing the default set of icon assets which was used whenever I was implementing icons such as the logout icon.
- web3dart [27] – this is a library that interacts with the Ethereum blockchain. It connects to an Ethereum node that was able to help me in sending transactions, generating private keys and setting up new Ethereum addresses.
- flutter_dotenv [28] – I was able to easily configure my backend of getting the token balance with global variables using a ‘.env’ file which contained my API key from Moralis
- bip39 [29]: this generated deterministic keys for whenever a new wallet is created.
- ed255519_hd_key [30]: this is a key derivation for ed25519, this was used when getting the private key from the mnemonic.
- hex [31] – easy hexadecimal encoding and decoding used in generating a wallet or getting information from a wallet.

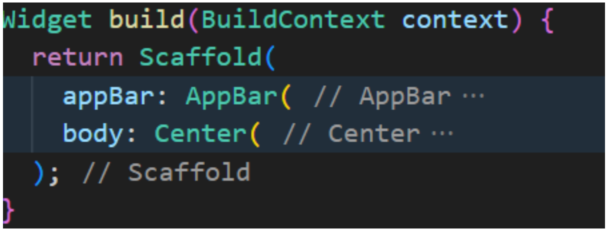
- provider [32] –In my application, the file “retry.dart”, it enables the “RetryPage” widget to access the “WalletProvider” instance and make decisions based on the state managed by the provider.
- shared_preferences [33] – wraps platform specific persistent storage for simple data, data is persisted to disk asynchronously. In my application, this is used to store and retrieve the private key associated with the user’s wallet.
- http [34] – this package contains a set of high-level functions and classes that make it easy to consume HTTP resources. In my crypto wallet, this is used to make HTTP requests to the Alchemy API for sending transactions to the Ethereum network. Also used to perform a HTTP request to a URL in order to fetch or send from backend server.
- fluttertoast [35] – this flutter library is designed to create toast messages effortlessly. Toast messages are brief notifications that appear on the user's screen for a short duration before disappearing. For example, after a successful sign-up, the user will receive a message similar to the one depicted below:

Sign Up Successfull

Widget build

As there’s a multitude of files and to avoid repeating myself, I’ll talk about common styling techniques applied across all pages.

All frontend files will have ‘build’ function, defining the page's structure and appearance. This function determines how widgets render their content and layout, adapting to current states and properties. Within it, a 'Scaffold' serves as a foundational framework, integrating key components like an app bar and body.

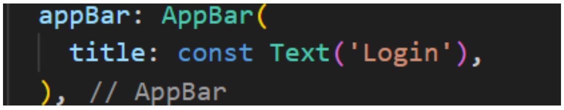


```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar( // AppBar ···
    body: Center( // Center ···
  ); // Scaffold
}
```

The provided code snippet is extracted from 'wallet_created_page.dart'. While the specific code details are hidden, the intention is to illustrate that in most of my files, the Scaffold widget typically encompasses these two main components.

The features ‘appBar’ and ‘body’ are typically included to define the appearance and behaviour of the widget:

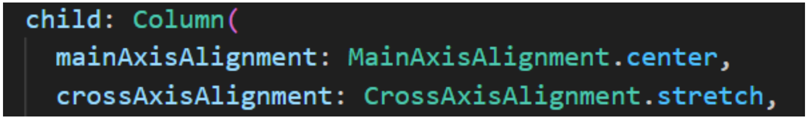
- appBar: an ‘appBar’ widget to provide a consistent top-level navigation bar with action, titles and other interactive elements.

  ```
  appBar: AppBar(
    title: const Text('Login'),
  ), // AppBar
  ```

  The code above is for the UI page of Login. The app bar “Login” is situated at the top of the screen and provides context and reassurance to users, indicating that they are in a secure section of the app.
- Body: various widgets are composed together to create the main content of the page:

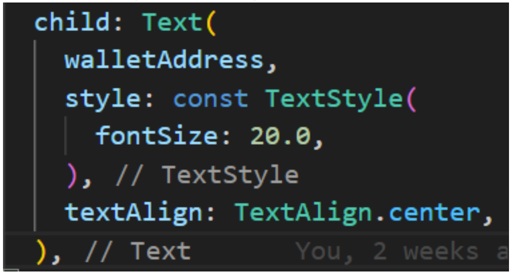
- ‘Column’ is used to arrange multiple child widgets vertically.



```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.stretch,
```

The above code is from ‘send_tokens.dart’ file where the layout is straightforward with a single-column arrangement of details. The ‘mainAxisAlignment’ and ‘crossAxisAlignment’ are controlling how the children are positioned within the column.

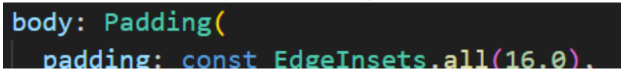
- Text: ‘Text’ is often used in my application to display text, sometimes there’s other widgets being used within it.



```
child: Text(
  walletAddress,
  style: const TextStyle(
    fontSize: 20.0,
  ), // TextStyle
  textAlign: TextAlign.center,
), // Text        You, 2 weeks a
```

The above code is from ‘wallet.dart’. The wallet address is displayed with a font size of 20 and also text is horizontally centred within its container.

- Padding: this was used to add space around its child widget, it allows you to add empty space, or padding, to the edges of a widget to adjust its position or size within its parent widget.



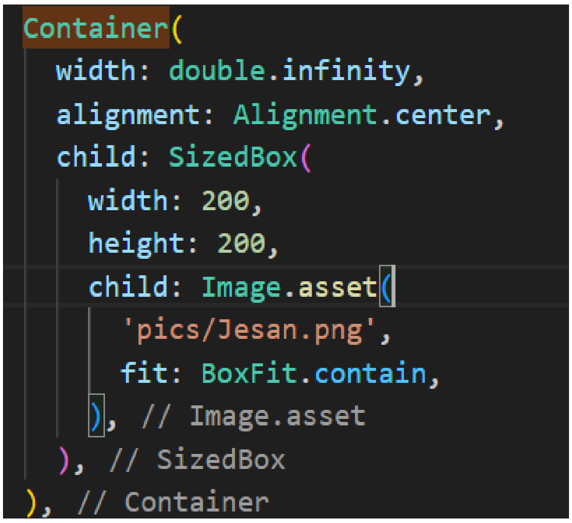
```
body: Padding(
  padding: const EdgeInsets.all(16.0),
```

The above code is from ‘emaillogin.dart’. The above code adds 16 pixels of padding on all sides of its child widget.

Home page

The first page I began to implement was the home page.

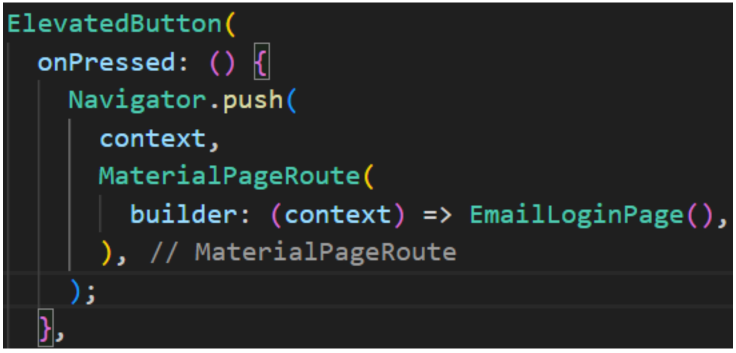
I am using an image logo that I made using a website called logo [36]. I made sure I followed the website’s terms of services before downloading the image I created.



```
Container(
  width: double.infinity,
  alignment: Alignment.center,
  child: SizedBox(
    width: 200,
    height: 200,
    child: Image.asset(
      'pics/Jesan.png',
      fit: BoxFit.contain,
    ), // Image.asset
  ), // SizedBox
), // Container
```

The above is code is the picture where it's constrained to fit within the bounds of the SizedBox with BoxFit.contain, ensuring it maintains its aspect ratio while fitting inside the specified width and height of 200 pixels each.

I created two widgets "Log In" and "Sign Up" when they are clicked, they are redirected to the login and sign-up page respectively.



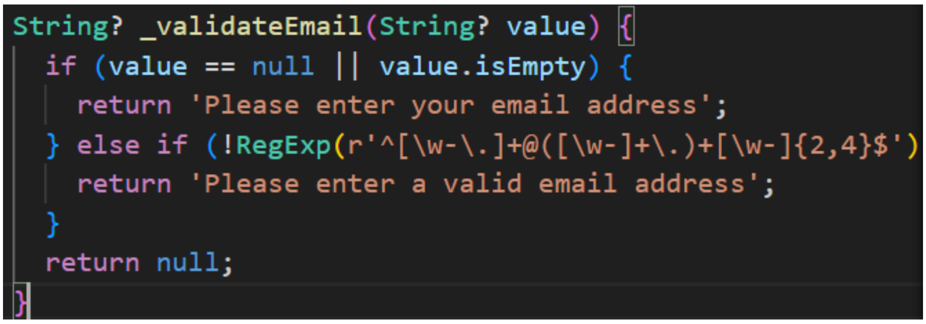
```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => EmailLoginPage(),
      ), // MaterialPageRoute
    );
  },
```

The 'onPressed', used in the code snippet above, utilizes the 'Navigator.push' method to navigate to another screen or page in the app. It takes two arguments: the first one is the current context, which provides information about the location of the widget in the widget tree, and the second argument is a MaterialPageRoute. This MaterialPageRoute specifies the route to the destination page, which is EmailLoginPage in this case.

The 'onPressed' handler will frequently be used in the implementation to dictate the action it takes when a user presses a button.

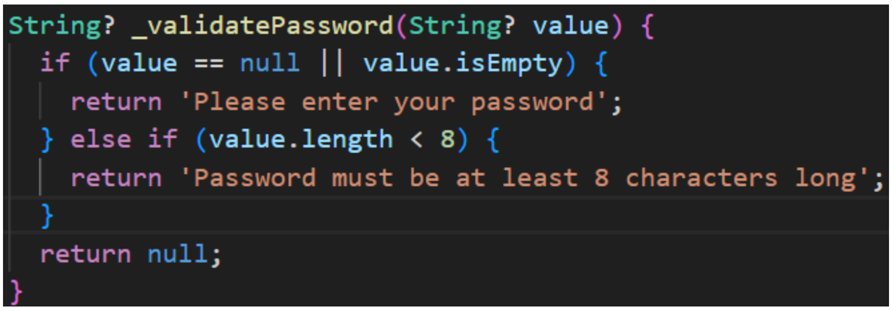
Sign Up

Next, I began to implement my sign-up page. I created the page layout that includes an app bar with a title and a body containing a form for user registration. Within the form, there are text form fields for entering an email address, a password, and a repeated password for confirmation. These fields are validated using the '_validateEmail' and '_validatePassword' methods, which return error messages if the input does not meet the specified criteria.



```
String? _validateEmail(String? value) {
  if (value == null || value.isEmpty) {
    return 'Please enter your email address';
  } else if (!RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$')
    return 'Please enter a valid email address';
  }
  return null;
}
```

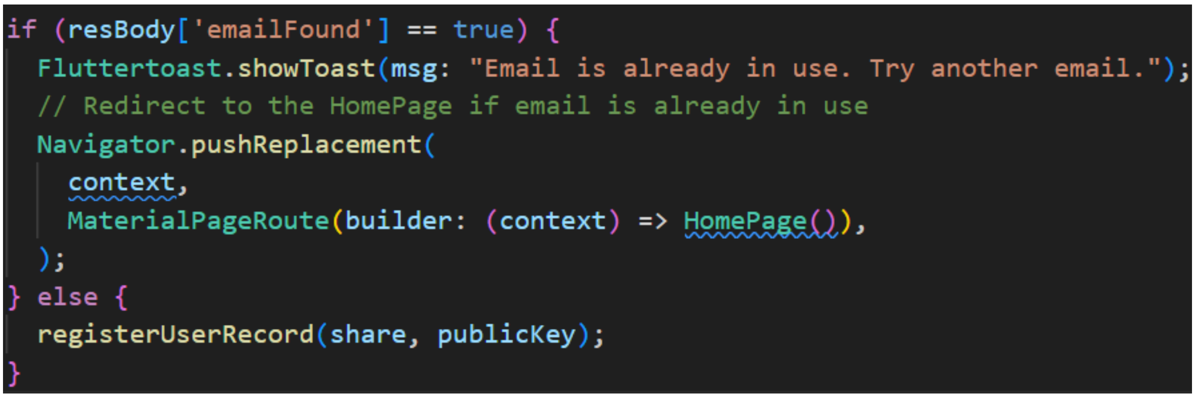
The code snippet above validates an email inputted by the user. It first checks if the value is empty, then it checks if it uses a regular expression to validate whether the email input matches the pattern of a valid email address. Otherwise, null is returned to indicate that email address is valid.



```
String? _validatePassword(String? value) {
  if (value == null || value.isEmpty) {
    return 'Please enter your password';
  } else if (value.length < 8) {
    return 'Password must be at least 8 characters long';
  }
  return null;
}
```

The code above validates a password inputted by the user. It first checks if the value is empty, then it checks if user password's length is less than 8. Otherwise, null is returned to indicate that the password is valid.

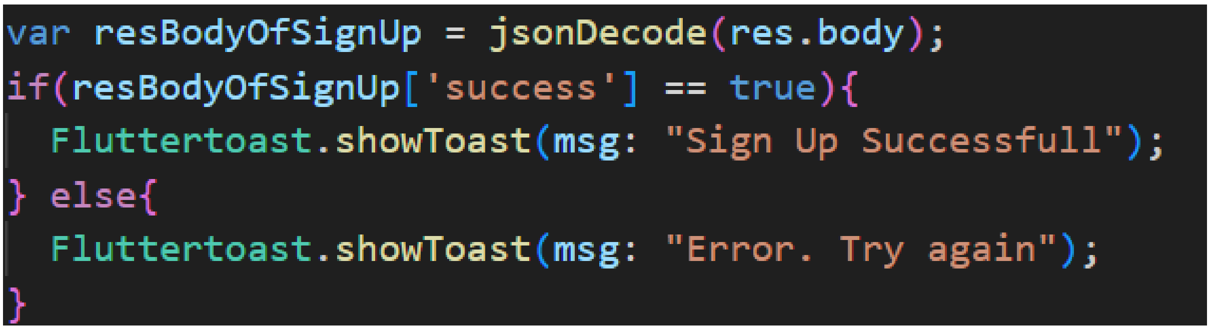
After implementing the backend of the User Authentication of the crypto wallet and the SQL database, I came back to this file to add 'validateUserEmail' and 'registerUserRecord' methods.



```
if (resBody['emailFound'] == true) {
  Fluttertoast.showToast(msg: "Email is already in use. Try another email.");
  // Redirect to the HomePage if email is already in use
  Navigator.pushReplacement(
    context,
    MaterialPageRoute(builder: (context) => HomePage()),
  );
} else {
  registerUserRecord(share, publicKey);
}
```

Above is a snapshot of 'validateUserEmail' method as I wanted to check if the email is already in the database, if it was then it would direct it back to the home page with a Fluttertoast message saying that the email is already in use. Otherwise, it would register the share of the private key and public key as well as the email in the database.

For both methods, it tries to connect to backend by sending a request. If the status code is 200, indicating a successful response, the response is decoded from JSON format.

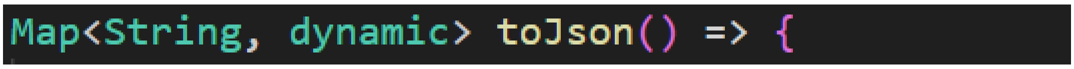


```
var resBodyOfSignUp = jsonDecode(res.body);
if(resBodyOfSignUp['success'] == true){
  Fluttertoast.showToast(msg: "Sign Up Successfull");
} else{
  Fluttertoast.showToast(msg: "Error. Try again");
}
```

The code above is from the 'registerUserRecord' method, where it checks if the decoded response body contains a key named 'success' with a value of true, meaning if it successfully registered the record of the user. If it does, a success message is displayed to the user using

Fluttertoast. If the value is not true, indicating an error on the server side, an error message is displayed.

I made a file ‘User.dart‘ where its convenient way to work with user data in Dart applications, facilitating the conversion between Dart objects and JSON representations.



```
Map<String, dynamic> toJson() => {
```

JSON is valued for its lightweight and readable data interchange format, widely supported across programming languages, simplifying parsing and generation. The `toJson()` method as shown in the code line above, formats data into JSON, as server typically expects it for storage. Conversely, `fromJson()` takes a Map<String**,** dynamic**>** as input, parses the JSON map, and returns a new User object. The keys in the JSON map correspond to the instance variables of the User class.

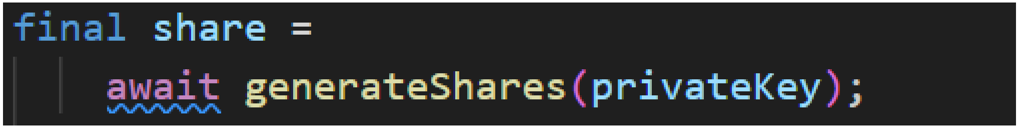
Generating an Ethereum Wallet

The file ‘wallet_provider.dart’ offers seamless functionality for generating, storing, and retrieving both private and public keys. It's particularly useful as it complements the sign-up page, which relies on another file to handle key generation.

While working on this section, I experimented with generating a wallet address without utilizing a mnemonic. Since my application does not require a mnemonic for wallet generation and only necessitates the private key, I aimed to simplify the process. However, upon reflection, I realized that I made the file more complex than necessary.

Splitting the Key

When user successfully signs up, it gets a share of the private key. This is initiated from the sign-up page as shown below:

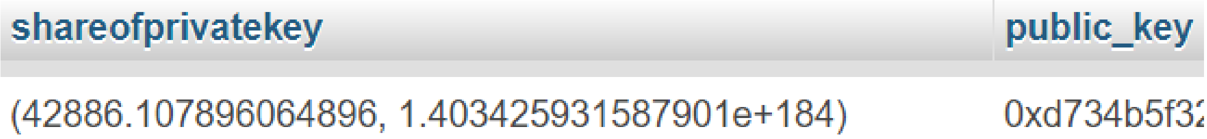


```
final share =
        await generateShares(privateKey);
```

This invokes the 'generateShares' function, which accepts the private key as input and returns a list containing two shares.

Initially, I opted to utilize Shamir Secret Sharing for splitting the key, but I faced certain challenges during the implementation phase.
One notable issue was that private keys are strings composed of a combination of numbers and integers randomly, whereas Shamir Secret Sharing exclusively operates with integers. This discrepancy led to errors in the process.

Hence, I opted for a different approach. I attempted to convert the private key into bytes, and then from bytes into decimals. While this method successfully generated shares, the resulting output did not align with my expectations. For instance, when one of the shares was stored in the database, it was saved as decimals, as illustrated below:



shareofprivatekey	public_key
(42886.107896064896, 1.403425931587901e+184)	0xd734b5f32

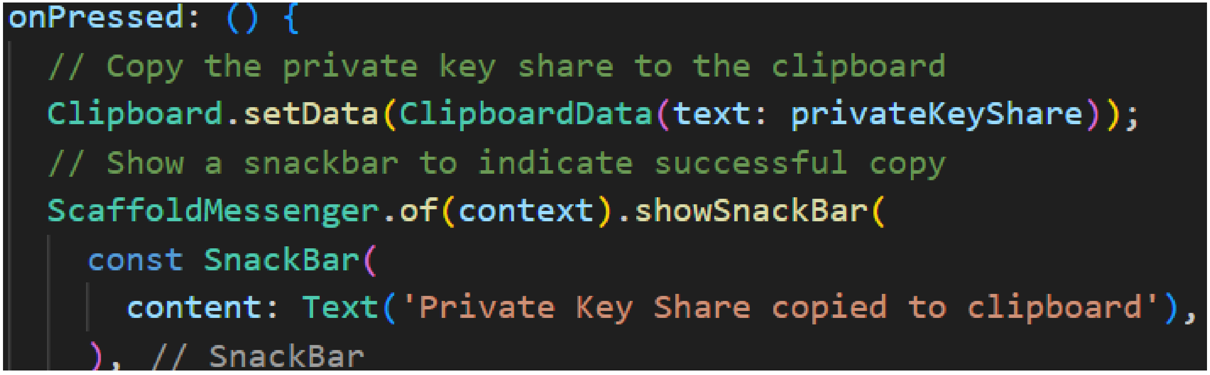
I double-checked by attempting to combine the database share with the user's share, but the output differed from my anticipated result.

Despite investing considerable time into troubleshooting this issue, I encountered no breakthroughs. Consequently, I made the decision to employ a simpler method for splitting the private key.

The method of splitting the key involved dividing the private key into two shares. The string is split at a randomly generated index within a specified range, this was to prevent the split from occurring too close to the edges of the string, which could cause issues to the security of the shares. If one share is significantly smaller than the other, it can introduce a vulnerability where an attacker, such as a service provider, may be able to reconstruct the private key by obtaining both shares. This is because the shorter share would provide less entropy, making it easier to guess or brute-force the missing portion of the key.

Wallet Created Page

I created a confirmation page after that displays that user has successfully created a wallet. It includes the public key and the share of the private key. It includes an AppBar with a title "Wallet Created" and an action button "Next" that navigates to the WalletPage when pressed.



```
onPressed: () {
  // Copy the private key share to the clipboard
  Clipboard.setData(ClipboardData(text: privateKeyShare));
  // Show a snackbar to indicate successful copy
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
      content: Text('Private Key Share copied to clipboard'),
    ), // SnackBar
```

The above code provides functionality to copy the private key share which is displayed along with an icon button to copy it to the clipboard, also a message reminding the user to keep the private key safe is displayed at the bottom of the page.

Wallet Page

Users are directed to the wallet page after a successful user authentication, where some of the UI elements displayed using the build function:

- Wallet Address and Balance: Displayed in a Column widget inside a Container, the wallet address is shown as text and can be copied to the clipboard by tapping on it, also the balance is displayed.
- Send and Refresh Buttons: Two FloatingActionButton widgets are displayed side by side, each representing the send and refresh functionalities. They have unique icons to differentiate them.

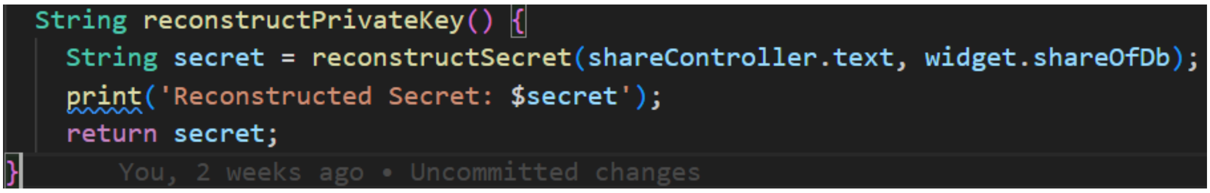
- TabBar and TabBarView: Used to implement tabbed navigation with two tabs: Assets and Options. The content of each tab is defined in the TabBarView.
- Assets Tab: Displays a Card widget containing Sepolia ETH network and the current balance.
- Options Tab: Displays a widget with an icon for logout. Tapping on it removes the private key from SharedPreferences and navigates the user back to the home page.

The 'loadWalletData()' method retrieves the private key from SharedPreferences and uses it to load the wallet address and balance.

Login Page

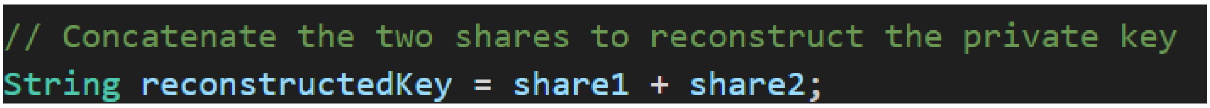
The log in page mirrors the structure and functionality of the sign-up page but is tailored for logging in. It follows a similar form layout, requesting the user's email and password. Upon submission, the page communicates with the backend server, linked to the database, to verify the user's credentials. If the provided information matches an entry in the database, the user successfully logs in and is directed to the import wallet page. Otherwise, an error message is displayed indicating that the login attempt failed.

At the 'import_ wallet.dart', it gets the info of the user that successfully logged in from the login page. It displays the public key and asks the user to enter their share of the private key that was given to the user when they signed up in the wallet created page.



```
String reconstructPrivateKey() {
  String secret = reconstructSecret(shareController.text, widget.shareOfDb);
  print('Reconstructed Secret: $secret');
  return secret;
}       You, 2 weeks ago • Uncommitted changes
```

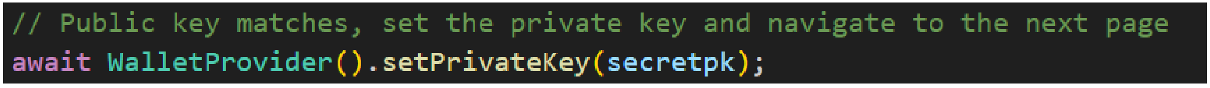
It calls on the function 'reconstructPrivateKey', shown in the snippet code above, which is located in 'import_wallet.dart' where it takes in the input of the share of the user private key with the share of the private key that's stored in the database.



```
// Concatenate the two shares to reconstruct the private key
String reconstructedKey = share1 + share2;
```

The two shares are simply combined to return the full private key.

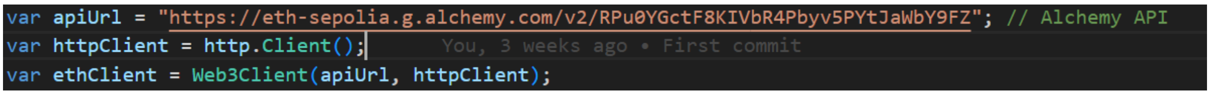
It then checks if the public key from the database matches with the public key generated from the private key. If it does, then it sets the private key into the Shared Preferences in 'wallet_provider.dart', and navigates to the wallet page as shown below.



```
// Public key matches, set the private key and navigate to the next page
await WalletProvider().setPrivateKey(secretpk);
```

Send tokens

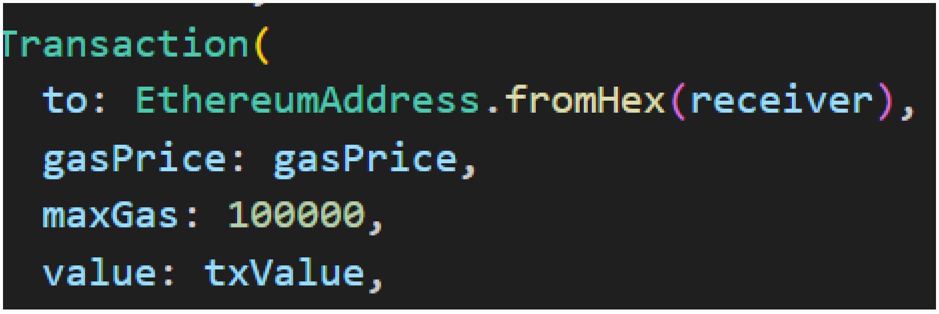
The page 'send_token.dart' was implemented so users can send tokens using the sepolia ETH network. It contains text fields widgets for entering the recipient's address and the amount of tokens to be sent.



```
var apiUrl = "https://eth-sepolia.g.alchemy.com/v2/RPu0YGctF8KIVbR4Pbyv5PYtJaWbY9FZ"; // Alchemy API
var httpClient = http.Client();       You, 3 weeks ago • First commit
var ethClient = Web3Client(apiUrl, httpClient);
```

The above code initializes an Ethereum client using the Alchemy API endpoint and a HTTP client, it sets up the necessary components for interacting with the Ethereum network.

It then sends a transaction using the method ethClient with the parameters shown below. ‘maxGas’ sets the maximum amount of gas to be 100000, Gas limits help prevent malicious behaviour or prevent infinite loops in smart contracts.



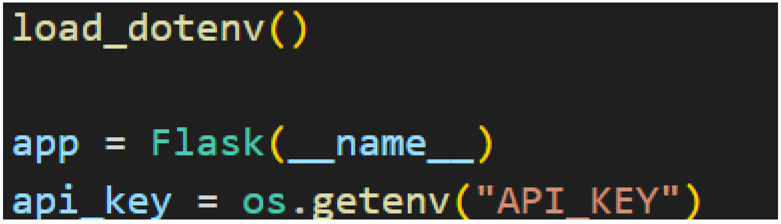
```
Transaction(
  to: EthereumAddress.fromHex(receiver),
  gasPrice: gasPrice,
  maxGas: 100000,
  value: txValue,
```

Backend Development

I implemented two backends, check balance and user authentication (sign up and log in).

Whilst implementing the wallet page, I had to create the backend for checking the balance of the token in of wallet address provided in Python.
I imported various modules. One of the modules I had to install was ‘moralis’: Moralis is a platform that allowed me to interact with the blockchain. The ‘evm_api’ submodule of Moralis is used to interact with Ethereum Virtual Machine, which is later used in the code by calling ‘balance’ that gets the balance by getting the address and the chain.

The load_dotenv() function loads environment variables from a .env file which contains the Moralis API, the ‘getenv(‘API_KEY’) retrieves the API key as shown below.

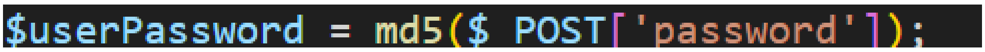


```
load_dotenv()

app = Flask(__name__)
api_key = os.getenv("API_KEY")
```

This backend is connected by ‘get_balances.dart’ in the frontend. The IP4 address has to the same and be updated regularly as it keeps changing. I used ‘ipconfig’ to check IP4 address regularly throughout the development.

I needed to establish a connection between my application and the database to manage user details effectively. This involved creating a backend infrastructure, which I deployed under XAMPP server for local development.
The first step was configuring a file to handle database connectivity, where I provided the necessary server details to establish a connection.

The next file I created was the signup which was handling requests related to user registration and saving user data to a database.
For the user's password, md5 function was used for basic encryption before sending it to the database.



```
$userPassword = md5($_POST['password']);
```

MD5 is known to be one of the weakest encryption methods due to its susceptibility to collision attacks, where different inputs can result in the same hash output. In contrast, SHA-256 offers significantly stronger security measures. It produces a 256-bit hash, which is more robust compared to MD5's 128-bit output.

While SHA-256 would have been the preferred encryption method for enhanced security, I encountered some challenges during implementation. As a result, I opted to proceed with MD5 for simplicity's sake.

I created another file that checks if the given user email already exists in the database, ensuring data integrity and preventing duplicate entries.

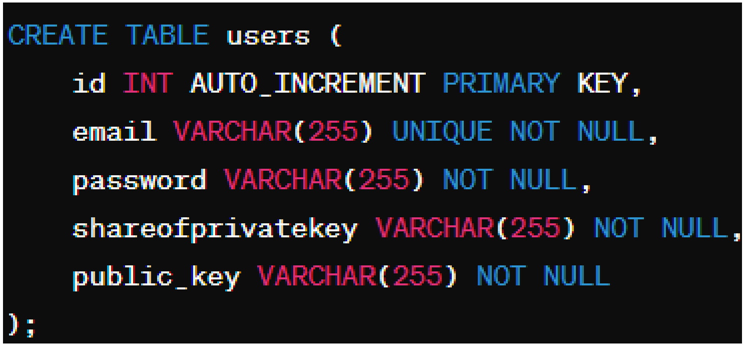
Lastly, I then created a file for login where it checks the provided email and password exists against those stored in the database. If it is, it retrieves the user record and sends a JSON response indicating success to the frontend.

The frontend code can easily access the backend API endpoints without hardcoding the URLs in the file 'api_connection.dart'. This enhances code maintainability and makes it easier to update endpoints if the backend URL changes in the future.

For all these authentication API files, SQL queries were used to insert or retrieve user data from the database.

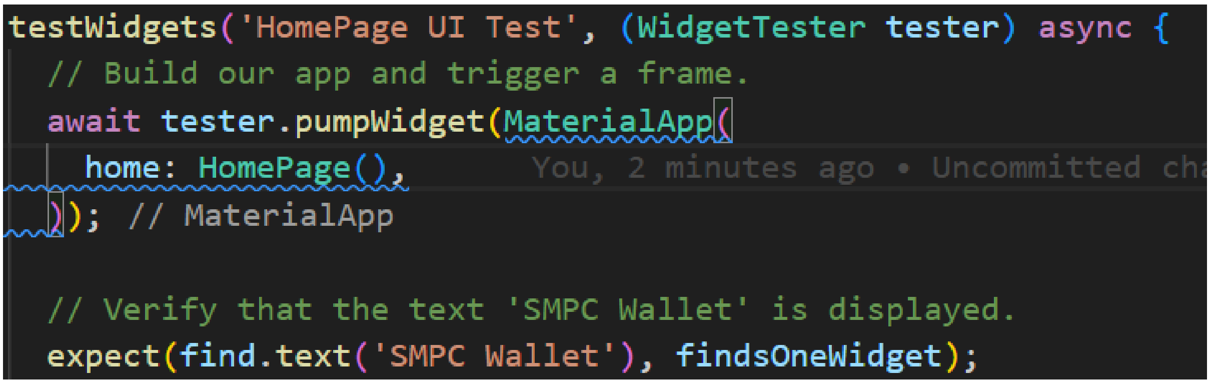
Database

I created the database 'userinfo' and created the 'users' table. I wrote SQL statements and executed this in phpMyAdmin [37], which was the interface provided by XAMPP to interact with the database.
I executed the query below to create the table:



```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    shareofprivatekey VARCHAR(255) NOT NULL,
    public_key VARCHAR(255) NOT NULL
);
```

'id' is an auto-increasing integer that serves as the primary key. The rest of the column names are a variable-length string field, 'NOT NULL' ensures that field is not empty.



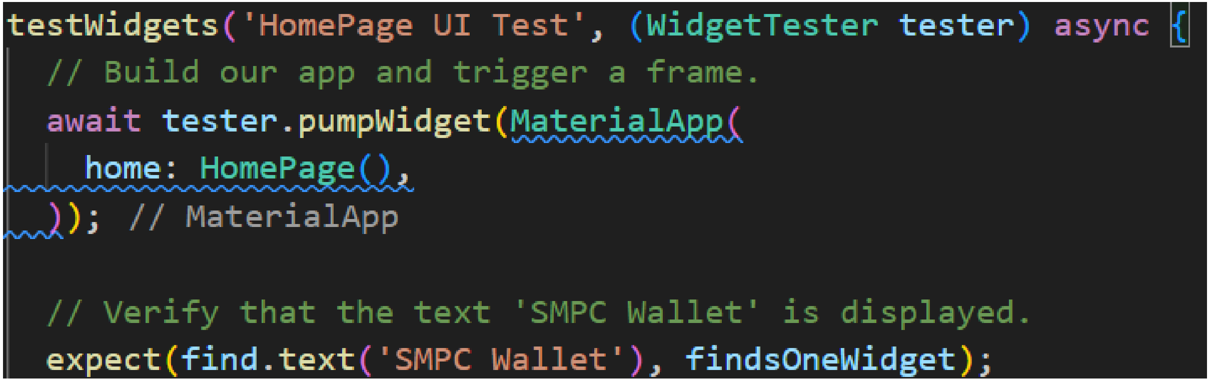
```
testWidgets('HomePage UI Test', (WidgetTester tester) async {
  // Build our app and trigger a frame.
  await tester.pumpWidget(MaterialApp(
    home: HomePage(),        You, 2 minutes ago • Uncommitted cha
  )); // MaterialApp

  // Verify that the text 'SMPC Wallet' is displayed.
  expect(find.text('SMPC Wallet'), findsOneWidget);
```

Testing & Evaluation

I used a variety of testing methodologies, these were: unit testing, endpoint testing and manual testing. For automated testing, I used a package from Flutter called ‘flutter_test’ [38] which allowed me to conduct testing of the frontend pages. I chose this as package has extensive functionalities, such as 'expect' and 'testWidgets'. These functionalities were instrumental in verifying the correctness and behaviour of various UI components and interactions within the application. To test the backend, I used endpoint testing. Finally, I did some manual testing to ensure full functionality.

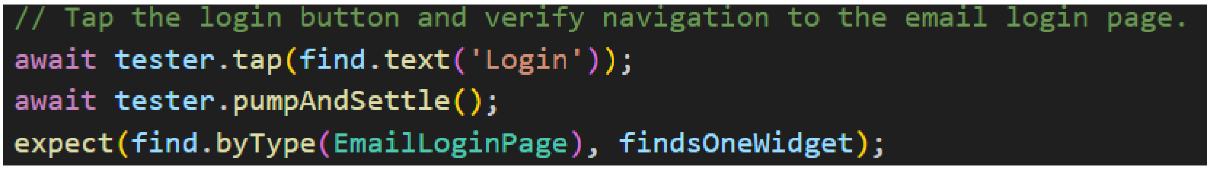
Automated Testing

I created automated test files, I started by creating ‘widget_test.dart’, that tested the home page. Every file that I tested had a similar structure to each other. For example:
The code below tests the Home page UI by verifying if the ‘HomePage’ widget is rendered within a ‘MaterialApp’, and if the text ‘SPMC Wallet is being displayed on the screen.
For the backend, I employed Postman [39] to dispatch HTTP requests and validate the responses from each endpoint, ensuring conformity with the expected outcomes.



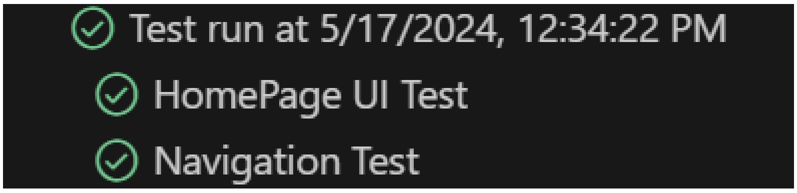
```
testWidgets('HomePage UI Test', (WidgetTester tester) async {
  // Build our app and trigger a frame.
  await tester.pumpWidget(MaterialApp(
    home: HomePage(),
  )); // MaterialApp

  // Verify that the text 'SMPC Wallet' is displayed.
  expect(find.text('SMPC Wallet'), findsOneWidget);
```

I also test the navigation of the page by verifying that tapping the login button in the app's UI leads to navigation to the email login page.



```
// Tap the login button and verify navigation to the email login page.
await tester.tap(find.text('Login'));
await tester.pumpAndSettle();
expect(find.byType(EmailLoginPage), findsOneWidget);
```

When I run these test files, results such as the one illustrated below were displayed:



✓ Test run at 5/17/2024, 12:34:22 PM

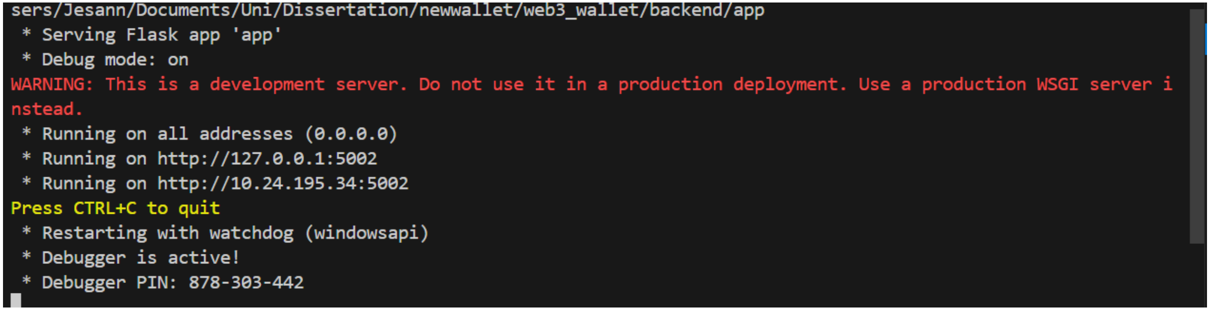
✓ HomePage UI Test

✓ Navigation Test

Showing a tick sign, meaning that all the tests were passed.

Endpoint testing

For my unit test of checking the balance of the wallet, I needed to check if the code was executed successfully and information about the server.

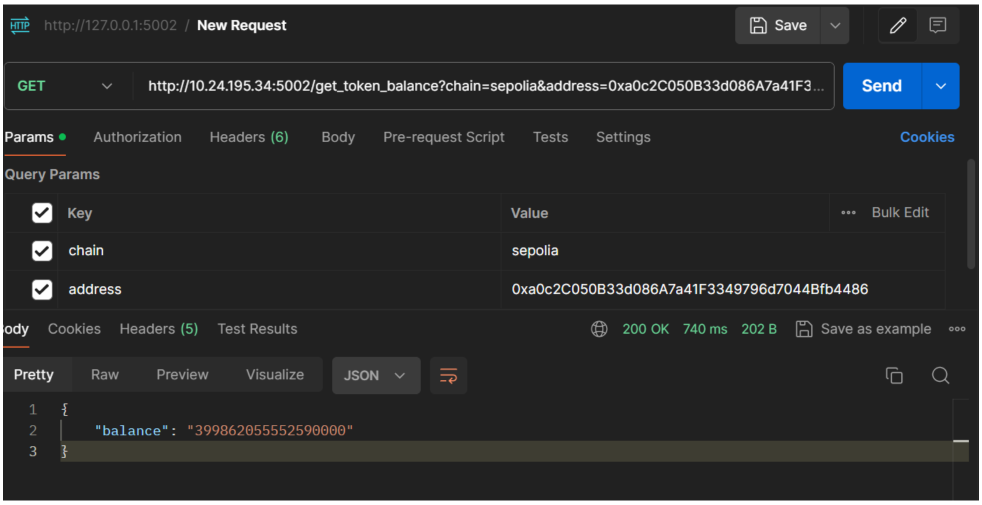


```
sers/Jesann/Documents/Uni/Dissertation/newwallet/web3_wallet/backend/app
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server i
nstead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://10.24.195.34:5002
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 878-303-442
```

Illustrated above shows my code was successfully executed, it provides information about the server's. The first address 'http://127.0.0.1:5002' is referred as localhost, is a loopback address that points back at my own computer. The next address 'http://10.24.195.34:5002' is my local IP address, which regularly needs to be updated as it changes every day. The fact that it's running on these addresses means that the backend server is up and running, and it's ready to handle requests for checking token balances.

I tested the endpoints using Postman to verify that valid inputs were accepted and that it returned an accurate error in the event that an error occurred.

For the backend of checking the balance, I tested with the chain 'sepolia' and the address '0xa0c2C050B33d086A7a41F3349796d7044Bfb4486'. I did this by adding it to the Postman Params, result is illustrated below:



GET http://10.24.195.34:5002/get_token_balance?chain=sepolia&address=0xa0c2C050B33d086A7a41F3...

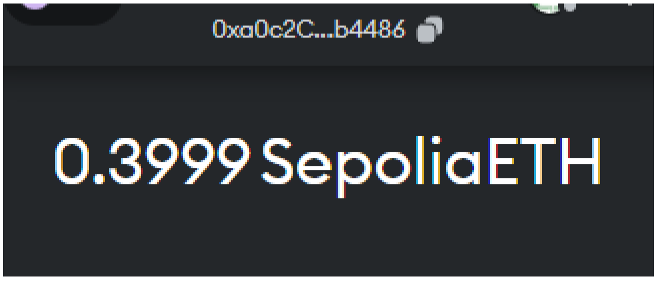
Query Params

☑ Key	Value
☑ chain	sepolia
☑ address	0xa0c2C050B33d086A7a41F3349796d7044Bfb4486

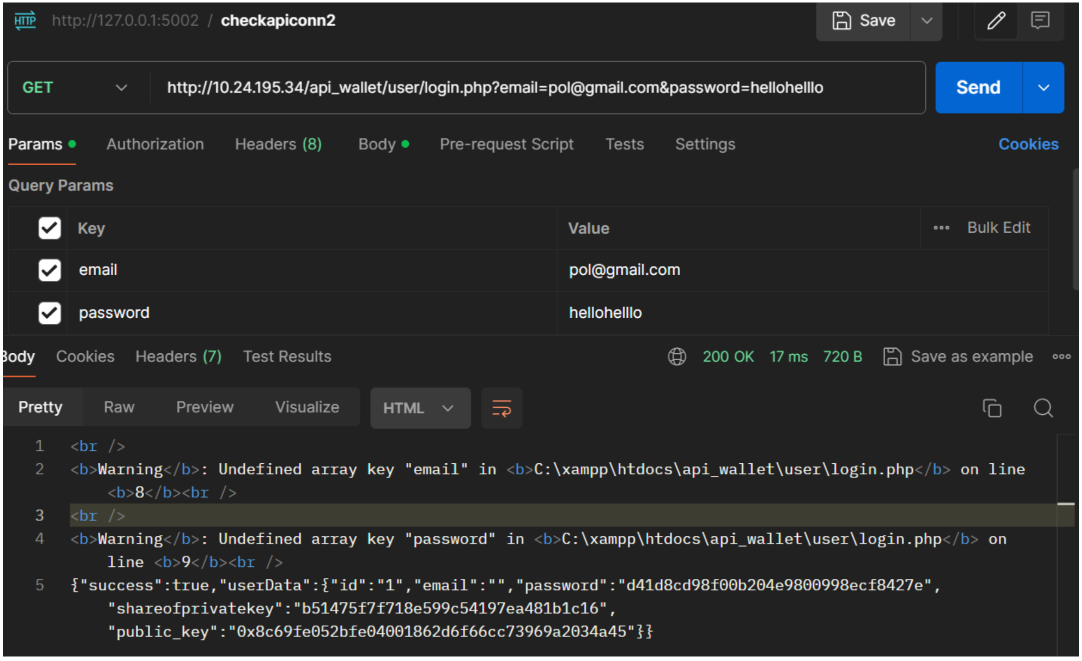
200 OK 740 ms 202 B

```
{
    "balance": "399862055552590000"
}
```

After sending the POST request I checked to see if the output was the same as the wallet balance in MetaMask, it was as shown below.



However, when I tried testing my other backend, Authentication PHP files, I keep getting errors despite being successfully connected to the database.



GET http://10.24.195.34/api_wallet/user/login.php?email=pol@gmail.com&password=hellohelllo

Query Params

☑ Key	Value
☑ email	pol@gmail.com
☑ password	hellohelllo

200 OK 17 ms 720 B

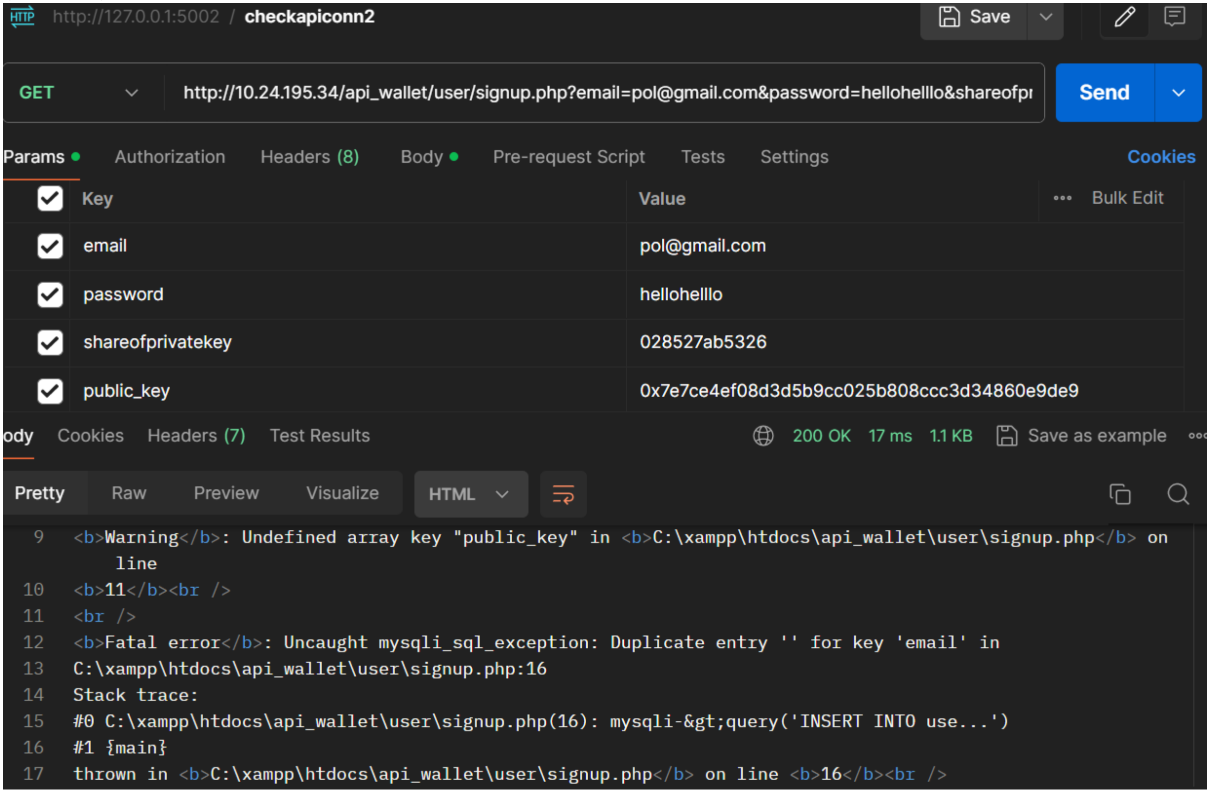
```
<br />
<b>Warning</b>: Undefined array key "email" in <b>C:\xampp\htdocs\api_wallet\user\login.php</b> on line
    <b>8</b><br />
<br />
<b>Warning</b>: Undefined array key "password" in <b>C:\xampp\htdocs\api_wallet\user\login.php</b> on
    line <b>9</b><br />
{"success":true,"userData":{"id":"1","email":"","password":"d41d8cd98f00b204e9800998ecf8427e",
    "shareofprivatekey":"b51475f7f718e599c54197ea481b1c16",
    "public_key":"0x8c69fe052bfe04001862d6f66cc73969a2034a45"}}
```

Above shows the testing of the backend of ‘login.php’, I was trying to insert the query parameters email and password that was already stored in the database. But it seems I need to review my file and make sure that its correctly accessing the requested parameters, I may have to use functions like ‘isset()’ or ‘array_key_exists()’ to check if the keys are set before accessing them.

A successful login would print out ‘success’ is true and the user data row that’s stored in the database, otherwise it would return an ‘success’ is false.

The same error occurs when testing my other files in the backend as shown in the Appendix.

Here's a screenshot of an error that indicates that my database connection to my backend is successful. Given that the input parameters I entered were already included in data from the database, the fatal error in line 12 indicates that there is a duplicate entry ('') for my key ('email').



http://10.24.195.34/api_wallet/user/signup.php?email=pol@gmail.com&password=hellohelllo&shareofpr

	Key	Value
☑	email	pol@gmail.com
☑	password	hellohelllo
☑	shareofprivatekey	028527ab5326
☑	public_key	0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9

```
 9  <b>Warning</b>: Undefined array key "public_key" in <b>C:\xampp\htdocs\api_wallet\user\signup.php</b> on
       line
10  <b>11</b><br />
11  <br />
12  <b>Fatal error</b>: Uncaught mysqli_sql_exception: Duplicate entry '' for key 'email' in
13  C:\xampp\htdocs\api_wallet\user\signup.php:16
14  Stack trace:
15  #0 C:\xampp\htdocs\api_wallet\user\signup.php(16): mysqli-&gt;query('INSERT INTO use...')
16  #1 {main}
17  thrown in <b>C:\xampp\htdocs\api_wallet\user\signup.php</b> on line <b>16</b><br />
```

Manual Testing

I employ manual testing techniques alongside these methods to thoroughly assess more complex pages, such as the Wallet page. Through manual testing, I ensure that each page's functionality operates as intended. For instance, I conduct basic tests to verify the functionality of the wallet page:

- Verify that wallet address is correctly displayed.
- Check if tapping on the wallet address copies it to the clipboard.
- Verify that balance is correctly displayed.
- Check that the refresh button works.
- Verify that user can switch between the tabs “Assets” and “Options”.
- Verify that the Assets tab displays the correct asset name and balance.
- Check if tapping on logout, user is directed to the Home page.

I developed my crypto wallet using an iterative development process, following the approaches described. As soon as I put each feature into practice, I tested it to make sure it achieved the goals and the functional requirements.

To formalise these tests, I created tables to show the input, the expected, and actual outputs as well as justification for each test. An example for the testing of checking balance can be shown below in a table, for more refer to Appendix.

Input (chain & address)	Expected Output	Actual Output	Justification for test

C: sepolia
A: 0xa0c2C050B33d086A7 a41F3349796d7044Bfb4486
	{
	"balance": "399862055552590000"
	}
		Expected.	See if valid entries are handled correctly.

C: sepol A: 0xa0c2C050B33d086A7 a41F3349796d7044Bfb4486	APiValueError: Invalid value sepol passed in to &lt;class	Expected	See if invalid entries are handled correctly
C: A:	ApiTypeError: Invalid type. Required value type is str and passed type was NoneClass at [&#39;args[0]&#39;]	Expected	See if blank inputs are handled correctly
C: A: 0xa0c2C050B33d086A7 a41F3349796d7044Bfb4486	ApiValueError: Invalid value &lt;NoneClass: None&gt; passed in to &lt;class	Expected	See if blank chain is handled correctly
C: sepolia A:	ApiTypeError: Invalid type. Required value type is str and passed type was NoneClass at [&#39;args[0]&#39;]	Expected	See if blank address is handled care.

User Evaluation

I asked 2 participants to use my software before having an open discussion with them. The Appendix contains an example of a participation consent form. After 7 minutes of using my software and signing up with sample data (using fake emails and passwords), I had an open discussion with the participants during which I posed the following open-ended questions:

1. What's your opinion on the user interface?
2. How confident do you feel about the security measures implemented in the application, especially regarding the handling of private keys?
3. Were all the features and functionalities you expected available and working as intended?
4. Are there any additional features or functionalities you would like to see added to the application?

These questions seek to assess my project's success and identify areas for development and next steps.

I have summarised the feedback from the users:

1. User Interface: Both users found the interface sleek and intuitive, praising its clean design and ease of navigation.
2. Security measures: A expressed confidence in the security measures, especially regarding the handling of private keys, while the other user found the private key share too cumbersome to manage but appreciated the security measures upon logout.
3. Functionalities working: Both users reported that all expected features worked smoothly, with one user suggesting an easier way to navigate back from the send tokens page to the wallet page.
4. Additional features: Both users suggested enhancements, such as adding support for additional cryptocurrencies to appeal to a broader user base. Additionally, another user emphasized the need add an activity tab.

I've learned from these conversations what aspects of my project have worked well and what still needs to be done in the future.
If I had more time, I would like to have added a more visible and clearer button to navigate between pages, such as creation of a button to go back to the wallet page from the send tokens page.
More details about each participant's feedback in the Appendix.

Evaluation

I compared my finished project to the initial Aims & Objectives I had set out, to evaluate the success of my project.

Aims	Evaluation
Develop a user authentication process.	I successfully implemented a login, sign up and an input validation for email functionality by connecting my frontend to my authentication backend which is then connected to a database. This demonstrates my projects depth as well the technical complexity.
Build a crypto wallet	I successfully implemented this as my application can generate a wallet and import a wallet
Create a mechanism for splitting and reconstructing the private key.	I couldn't implement Shamir's Secret Sharing, instead I implemented a simple splitting and reconstructing method
Enable secure token transactions and balance checking for users.	I was able to implement a backend to check balances and integrated an API to send transactions
Design an intuitive and responsive user interface.	I achieve this aim for most of my application, however, some participants highlighted a better navigation is needed on some pages

Unmet Objectives

Due to technical and time constraints, I was unable to meet the objective of implementing Shamir Secret Sharing as originally planned. As outlined in the implementation details, I opted for a simpler method to split the private key and reconstructing it. This was because I wanted to focus on getting the other objectives completed to a good standard. This in turn affected my implementation of adding a backup key share for both the user and the service provider.

Project Ethics

I followed the University's Ethical Guidelines throughout my project.

My project falls under Data Category B as it involves retrieving information from the Alchemy API and the Moralis API. The data accessed through the Alchemy API pertains primarily to blockchain transactions and does not include any personally identifiable information about the users. Similarly, the Moralis API is used to fetch data related to users' wallets, which is considered anonymous data from a public source.

My project falls under Human Participation Category 2. Each participant signed a consent form in compliance with the ethical guidelines and received a participant information sheet. I ensured they understood that participation is entirely voluntary and that stopping at any moment would have no repercussions. I strictly anonymized the data I collected during these sessions, and I only used it to assess my software. All electronically stored participant data was safeguarded on university servers and will be removed upon graduation to further protect participant data.

Conclusion & Future Work

Conclusion

In conclusion, this project successfully achieved its primary objectives of developing a robust and secure crypto wallet application using Secure Multi-Party Computation. The application provides a user-friendly interface, enabling users to sign up, log in, manage their wallets, view balances, and conduct transactions with ease. The integration of SMPC ensures enhanced security by splitting the private key between the user and the service provider, thereby addressing the critical issues of key management and security.

Throughout the development process, I adhered to the project's aims, implementing features that prioritize user experience and security. The frontend, developed with Dart, offers a dynamic and responsive user interface, while the backend, utilizing PHP and Python, ensures efficient handling of user authentication and blockchain interactions via the Moralis API. The SQL database was effectively used to organize user information, enhancing performance and reducing redundancy.

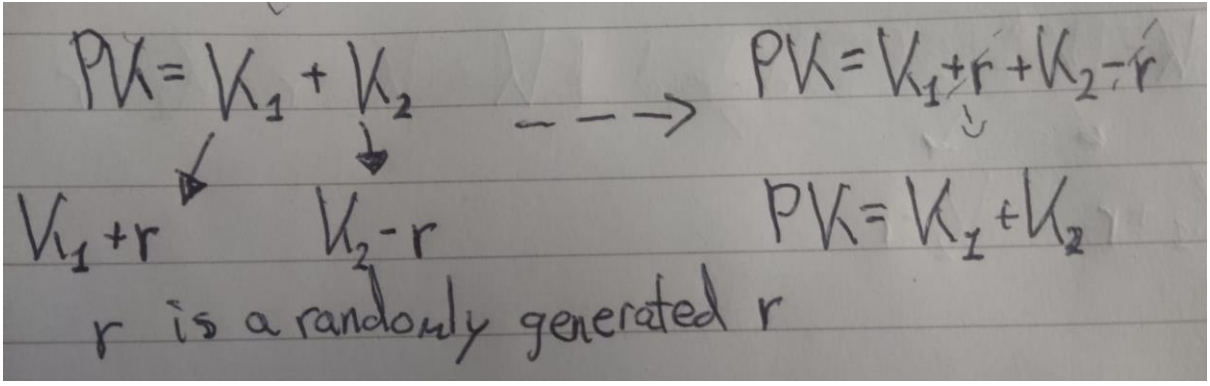
Despite some challenges, particularly with implementing Shamir Secret Sharing due to technical constraints, alternative methods were employed to handle the private key securely. This project has laid a solid foundation for future enhancements, such as adding support for more cryptocurrencies and improving user navigation features.

Overall, this project demonstrates a comprehensive approach to building a secure and user-friendly crypto wallet, addressing both usability and security concerns effectively.

Future Work

While my project has achieved its primary objectives, there are still areas that require attention to enhance its capabilities:

- Complete the incomplete objective - the implementation of Shamir's Secret Sharing to bolster security measures, this will help my other future implementations such as making a backup. I could research more about how to convert to and from the shares to the original private key successfully.
- Remove the single point of failure, I could implement a backup share for both the service provider and the user. Each device encrypts the share they generate using a backup key, ensuring a publicly verifiable backup. For a regular and easy-to-use backup, the user holds a private decryption key in a cloud backup, while the service provider backs up their share locally. The user also backs up their share by encrypting it with a backup encryption key and sends it to the service provider. In case the user loses their device and all associated data, the service provider sends the encrypted backup key to the user that would require the user to provide more proof that it's the rightful owner. The user then retrieves the decryption key from the cloud backup and decrypts the share, allowing for data recovery.
- Refresh mechanism at periodic intervals to fortify security defences as it forces the attackers to breach simultaneously.
  This can be done by updating the shares of the private key in such a way that full private key remains unchanged but requires the attacker to breach both shares simultaneously to gain access to the refreshed key. This is illustrated below:

  $PK = K_1 + K_2 \dashrightarrow PK = K_1 + r + K_2 - r$

  $K_1 + r \qquad K_2 - r \qquad PK = K_1 + K_2$

  r is a randomly generated r

  To generate the value r, use MPC protocol to choose a random r that neither side can influence.
- Make the wallet page more comprehensive- by adding more additional cryptocurrency networks for users to check their balance. Additionally, introducing an activity tab will provide users with comprehensive insights into their transaction history and account activities, ensuring transparency.

By fixing these issues, the platform would address every criticism made by users as well as any places where I thought it could be made better. These are in order of priority where I would implement this if I had more time. These features would increase the security as well as making it more appealing to the user, which would in turn help to reach a greater audience.

BCS Project Criteria & Self-Reflection

BCS Project Criteria Compliance

- Application Of Practical and Analytical Skills: I used a wide range of practical skills from my degree programme, including database management, software engineering principles, and programming in multiple languages (Dart, PHP, and Python). The careful selection of technologies and the detailed description of each role in my system design highlights my analytical skills.
- Innovation and Creativity: I split the private key into two shares in a unique approach that enhances security by ensuring no single entity possess the entire key. Also, implementing an authentication process to verify user before share from service provider is combined with input share of user adds an extra layer of security.
- Synthesis of Information, Ideas and Practices: This was shown by the way I integrated different technologies and API to create my final product. The compilation of user feedback demonstrated how my project can change and progress based on real world input.
- Solution Evaluation: Unit and manual testing were used throughout the development process to assess the produced platform. This enabled the system to be improved and ultimately achieve its initial aims and objectives.
- Meeting Real Needs in a Wider Context: My project provides a hybrid solution that combines self-custody with the security measures of typical exchanges. This dual approach ensures that neither the user nor the service provider holds complete control over the private key, significantly reducing the risk of unauthorized access providing a trust less security model and enhancing security.
- Ability to Self-manage a Significant Piece of Work: To ensure that this project was finished by the academic deadlines, careful planning, organisation, and time management were needed. This was crucial because a large project like this would show signs of poor time management and produce a subpar product that fell well short of the goals and objectives.

Self-reflection

Strengths:

- Problem Solving: I successfully overcame every obstacle I encountered while working on my project, particularly the integration of various technologies.

- Technical Development: I was unfamiliar with the language Dart, but I was able to learn it whilst implementing my project, demonstrating my ability to expand my technical skills and ability to adapt.
- Integrating technologies: I was able to integrate the backend to a database and from the frontend to the backend or how to use and call a public API, which has helped me to expand my scope in software development.

Weakness:

- Security concerns: I used MD5 for password hashing, it would be better to implement a stronger hash algorithm. I implemented a simple method for the splitting of the key instead of Shamir Secret Sharing.
- Automated testing: I only tested my application using automated testing at the end. If I had used it earlier during the development, it would have saved a lot of time as it would pick any errors or anything I missed.
- User evaluation: The delay in conducting user evaluations until the project's later stages meant that minor inconsistencies and navigation issues were not addressed promptly. I should have done early evaluations which would have allowed me to timely identify and resolve these issues.

References

[1] H.E Arslanian, *The book of crypto*.: Springer Books, 2022. [Online]. https://link.springer.com/content/pdf/10.1007/978-3-030-97951-5.pdf

[2] Inpher. (2015) 'What is secure multiparty computation'. [Online]. https://inpher.io/technology/what-is-secure-multiparty-computation/

[3] Google. (2017) Dart. [Online]. https://dart.dev/

[4] Moralis. (2021) Web3 Data Api. [Online]. https://docs.moralis.io/web3-data-api/evm

[5] Geeksforgeeks. (2022, June) Shamirs secret sharing cryptography. [Online]. https://www.geeksforgeeks.org/shamirs-secret-sharing-algorithm-cryptography/

[6] ConsenSys. (2016) Metamask. [Online]. https://metamask.io/

[7] Romanosky S, Cranor LF C Kuo, "Human selection of mnemonic phrase-based passwords," *InProceedings of the second symposium on Usable privacy and security*, pp. 67-78, July 2006. [Online]. https://dl.acm.org/doi/abs/10.1145/1143120.1143129

[8] Roderic, and Harshit Trivedi Broadhurst, "Malware in spam email: Risks and trends in the Australian Spam Intelligence Database," *Trends and Issues in Crime and Criminal Justice*, pp. 1-18, September 2020. [Online]. https://search.informit.org/doi/abs/10.3316/INFORMIT.447452190420867

[9] BBC. (2022, January) Crypto money laundering rises 30%, report finds. [Online]. https://www.bbc.co.uk/news/technology-60072195

[10] CoinCover. (2023, October) Risk Review: Single Point of Failure in the World Cryptocurrency. [Online]. https://www.coincover.com/blog/risk-review-single-point-of-failure-in-the-world-of-cryptocurrency

[11] HeyAlfie. (2021, September) Transactions If There's No Central Authority?" Let's Look At How Blockchain Works. [Online]. https://medium.com/@heyalfie.io/how-safe-are-my-crypto-transactions-if-theres-no-central-authority-33d7380659b2

[12] FMTAD. (2023, May) Recovery Phrase Backup: How to Secure It. [Online]. https://freemindtronic.com/recovery-phrase-backup-how-to-secure-it/

[13] Pascal Gauthier. (2014) Ledger. [Online]. https://www.ledger.com/

[14] Wikipedia. (2022, November) Bankruptcy of FTX. [Online]. https://en.wikipedia.org/w/index.php?title=Bankruptcy_of_FTX&action=history&dir=prev

[15] Joe Light. (2023, February) You Still Owe the IRS Even if Your Crypto Lender Collapsed. [Online]. https://www.barrons.com/articles/crypto-firm-bankruptcy-tax-return-51675879096

[16] Aman Ladia. (2019, July) ZeroWallet: A ZKP based Wallet Authentication Mechanism. [Online]. https://medium.com/@amanladia1/zerowallet-a-zkp-based-wallet-authentication-mechanism-9871dcca0a01

[17] THE INVESTOPEDIA TEAM. (2023, October) Multi-Signature Wallets: Definition and Use Cases. [Online]. https://www.investopedia.com/multi-signature-wallets-definition-5271193

[18] Radek Ostrowski. (2018, March) Time-locked Wallets: An Introduction to Ethereum Smart Contracts. [Online]. https://www.toptal.com/ethereum-smart-contract/time-locked-wallet-truffle-tutorial

[19] Wikipedia. (2004, May) Secure multi-party computation. [Online]. https://en.wikipedia.org/wiki/Secure_multi-party_computation

[20] Oscar, Kemal Akkaya, and Soamar Homsi Bautista, "Outsourcing Secure MPC to Untrusted Cloud Environments with Correctness Verification," in *46th Conference on Local Computer Networks (LCN)*, Edmonton, 2021, pp. 178-184. [Online]. https://ieeexplore.ieee.org/abstract/document/9524971

[21] Yomtov O, Galansky A Makriyannis N. (2023) Practical key-extraction attacks in leading mpc wallets. [Online]. https://eprint.iacr.org/2023/1234

[22] Google. (2017, May) Flutter. [Online]. https://flutter.dev/

[23] Alchemy. (2023, March) What is the Sepolia testnet? [Online]. https://www.alchemy.com/overviews/sepolia-testnet

[24] Microsoft. (2015, April) Visual Studio Code. [Online]. https://code.visualstudio.com/

[25] Apache Friends. (2002, September) XAMPP. [Online]. https://microdata.gov.in/dashboard/

[26] Pierre-Louis Victor Sanni. cupertino. [Online]. https://github.com/flutter/flutter/tree/master/packages/flutter/lib/src/cupertino

[27] Dart. web3dart package. [Online]. https://pub.dev/documentation/web3dart/latest/

[28] Dart. flutter_dotenv package. [Online]. https://pub.dev/documentation/flutter_dotenv/latest/

[29] Dart. bip39 package. [Online]. https://pub.dev/documentation/bip39/latest/

[30] Dart. ed25519_hd_key package. [Online]. https://pub.dev/documentation/ed25519_hd_key/latest/

[31] Dart. hex package. [Online]. https://pub.dev/documentation/hex/latest/

[32] Dart. provider package. [Online]. https://pub.dev/documentation/provider/latest/

[33] Dart. shared_preferences package. [Online]. https://pub.dev/documentation/shared_preferences/latest/

[34] Dart. http package. [Online]. https://pub.dev/documentation/http/latest/

[35] Dart. fluttertoast package. [Online]. https://pub.dev/documentation/fluttertoast/latest/

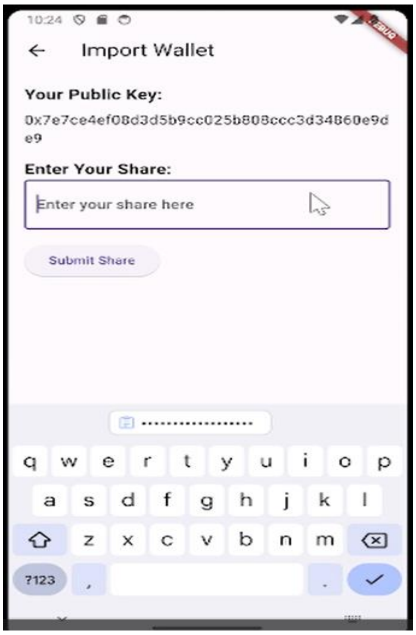
[36] logo. [Online]. https://logo.com/

[37] The phpMyAdmin Project. (1998, September) phpMyAdmin. [Online]. https://www.phpmyadmin.net/

[38] longhoangwkm. bip39. [Online]. https://github.com/dart-bitcoin/bip39

Appendices

UI/UX:



10:24

Import Wallet

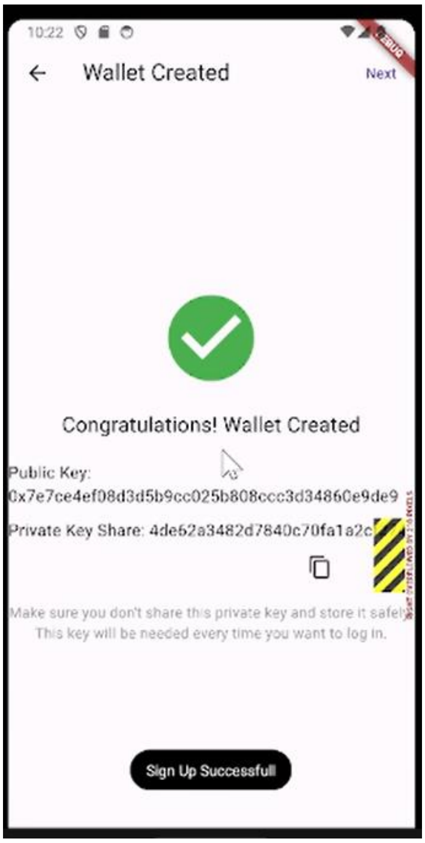
Your Public Key:

0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9

Enter Your Share:

Enter your share here

The layout is straightforward, with a clear title “Import Wallet”, and clear labels provides context and reassurance to users, indicating that they are in a secure section of the app.
The focus is on essential tasks: displaying the user’s public key and a box that allows the user to enter their share of private key.



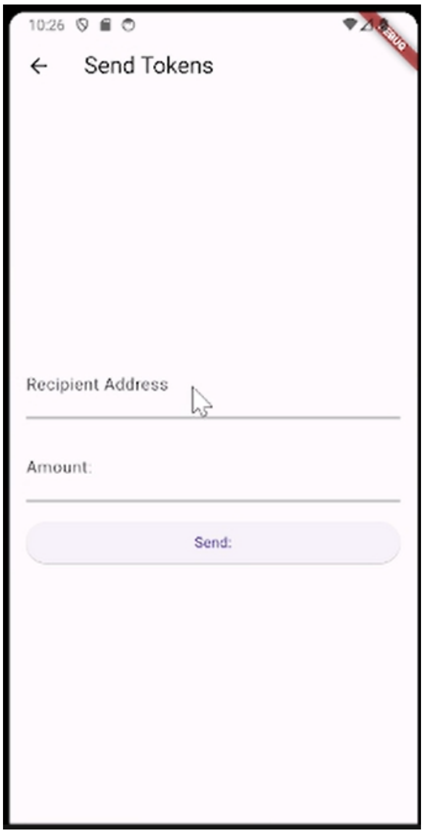
Congratulations! Wallet Created

Public Key:
0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9

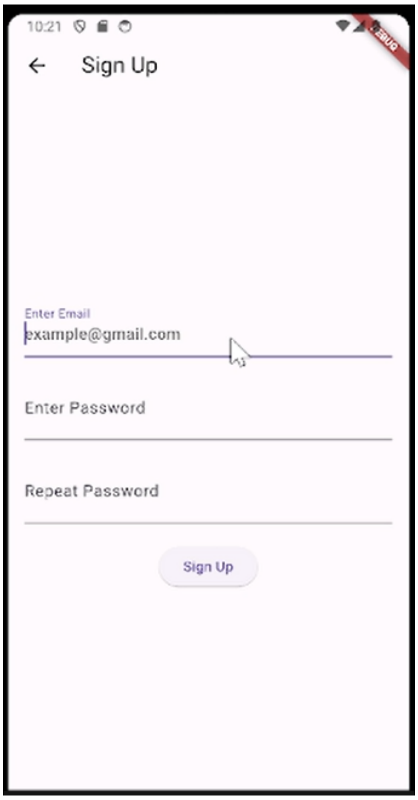
Private Key Share: 4de62a3482d7840c70fa1a2c

Make sure you don't share this private key and store it safely
This key will be needed every time you want to log in.

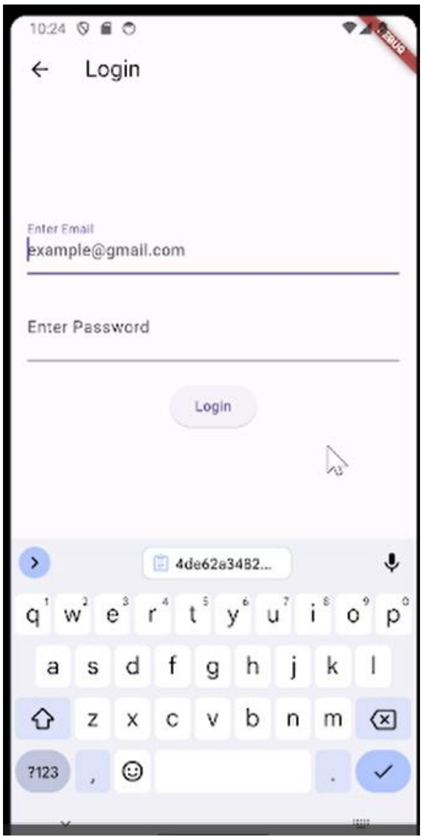
The layout is centred, with elements positioned for the user to easily view.
The checkmark icon and the bold text convey a sense of importance and accomplishment.
The app bar titled "Wallet Created" reinforces the significance of the action and assures users of the security of their newly created wallet.
Private key shares are provided with an icon option that allows the user to easily copy.
There are text guides below that say "Make sure you don't share this private key and store it safely. This key will be needed every time you want to log in.", this emphasises the importance of safeguarding the private key share and informs user that this phrase will be needed if he wants to log in.



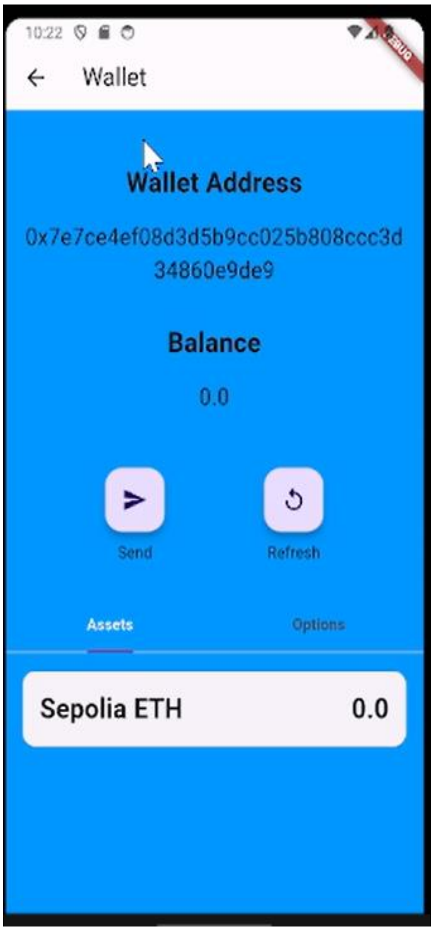
The layout is straightforward with a single-column arrangement of details, the focus is on required user input: recipient address and amount to send.
The app bar "Send Tokens" provides context and reassurance to users, showing that they are in a secure section of the app.
The labels are clear "Recipient Address" and "Amount" and guides users through the token sending process.



The layout maintains a clean and uncluttered appearance. The essential user inputs are prominently featured- email, password, and repeat password.
The app bar “Sign Up” provides a clear indication of the user’s current action within a secure section of the app.
Password fields obscure text for enhanced privacy, ensuring that passwords remain hidden from view.
Clear and consistent labelling “Enter Email”, “Enter Password”, “Repeat Password” facilitates intuitive interaction.
The auto validation prompts user to enter valid email and password formats, increasing the form completion accuracy.
Asking the user to repeat the password ensures that users confirm their password accurately before submission.
A small message prompt appears below on the next page (walletcreated)if user is successfully signed up, displaying “Sign Up Successful”, otherwise redirects it to the previous page(homepage), displaying “Error. Try again.”



The layout is clean and uncluttered, the focus is on essential inputs: email and password.
The app bar “Login” provides context and reassurance to users, indicating that they are in a secure section of the app. The alignment of the elements and consistent typography enhance readability and usability.
Clear labels “Enter Email” and “Enter Password” guide users through the login process.
If successfully logged in, a small message prompt appears in the next page (ImportWallet) below with a message “Log In Successful”, otherwise it stays on this same page (login) and a message “Wrong email or password. Try again”.



Wallet Address

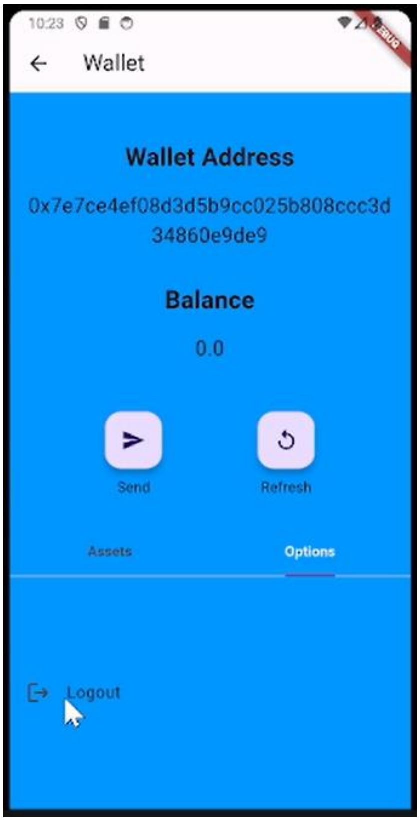
0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9

Balance

0.0

Sepolia ETH	0.0

The layout is organised to prioritize key wallet details, which are the wallet address and balance, presented at the top of the page.
There are concise and clear text labels that guide users through my interface, ensuring ease of understanding and interaction.
There's an option tapping to copy the wallet address to the clipboard, this use of gesture enhances the usability and user engagement as wallet address tend to be long strings and mistakes can occur whilst copying if done manually.
The floating action buttons for sending tokens and refreshing the page offers the user to further actions, promoting efficiency in wallet management.
The tab bar allows users to switch between the sections "Assets" and "Options" making it easy exploration of the wallet features and settings.
Under the "Assets" tab, there's a card widget that provides a visually distinct container, enhancing the presentation of the asset details. There's only one asset name present "Sepolia ETH" with the asset's balance.



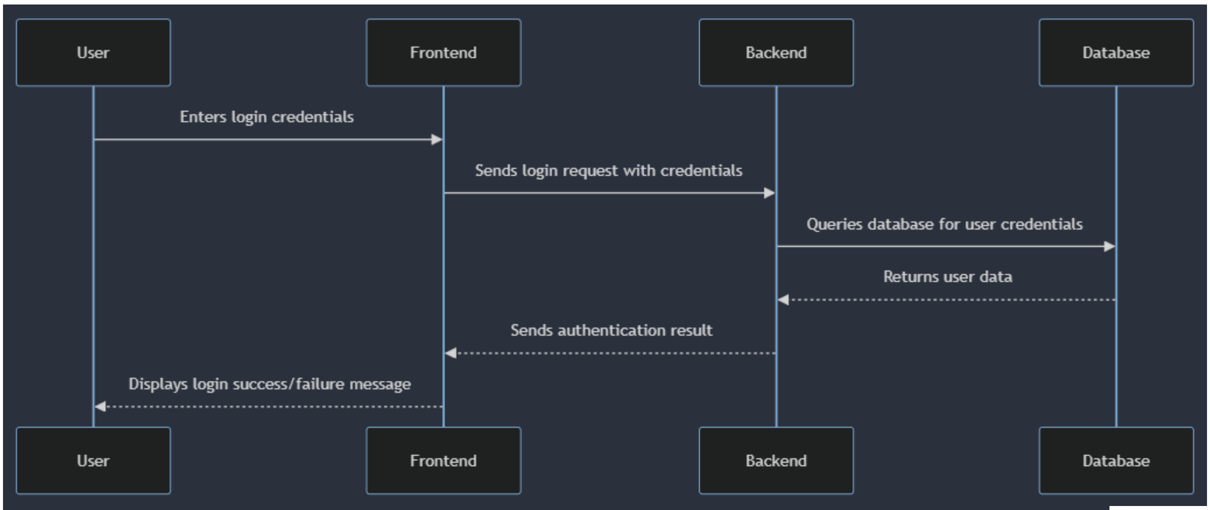
Wallet Address

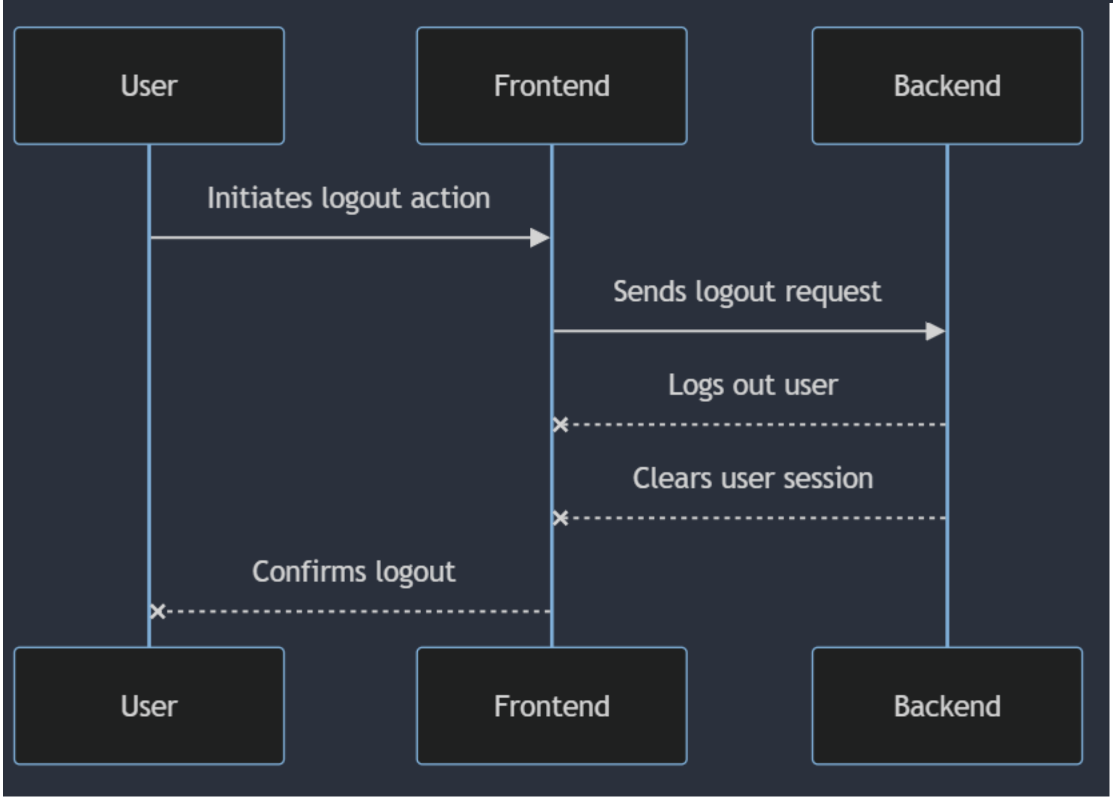
0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9

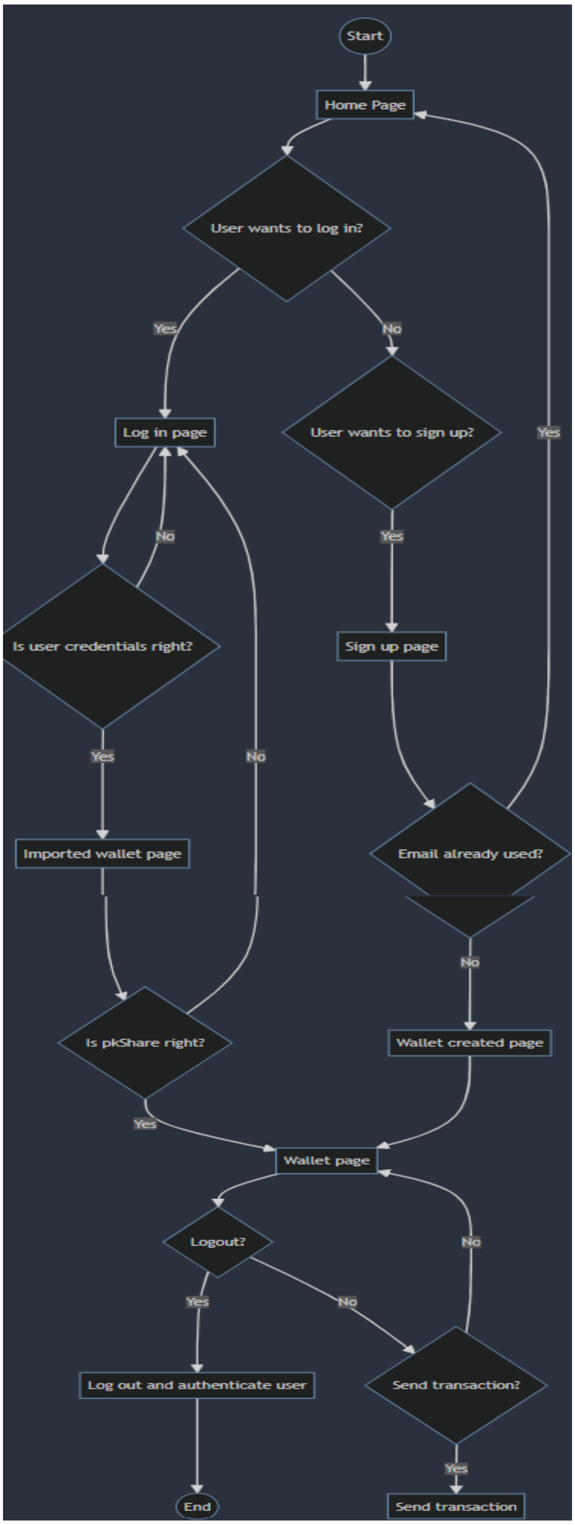
Balance

0.0

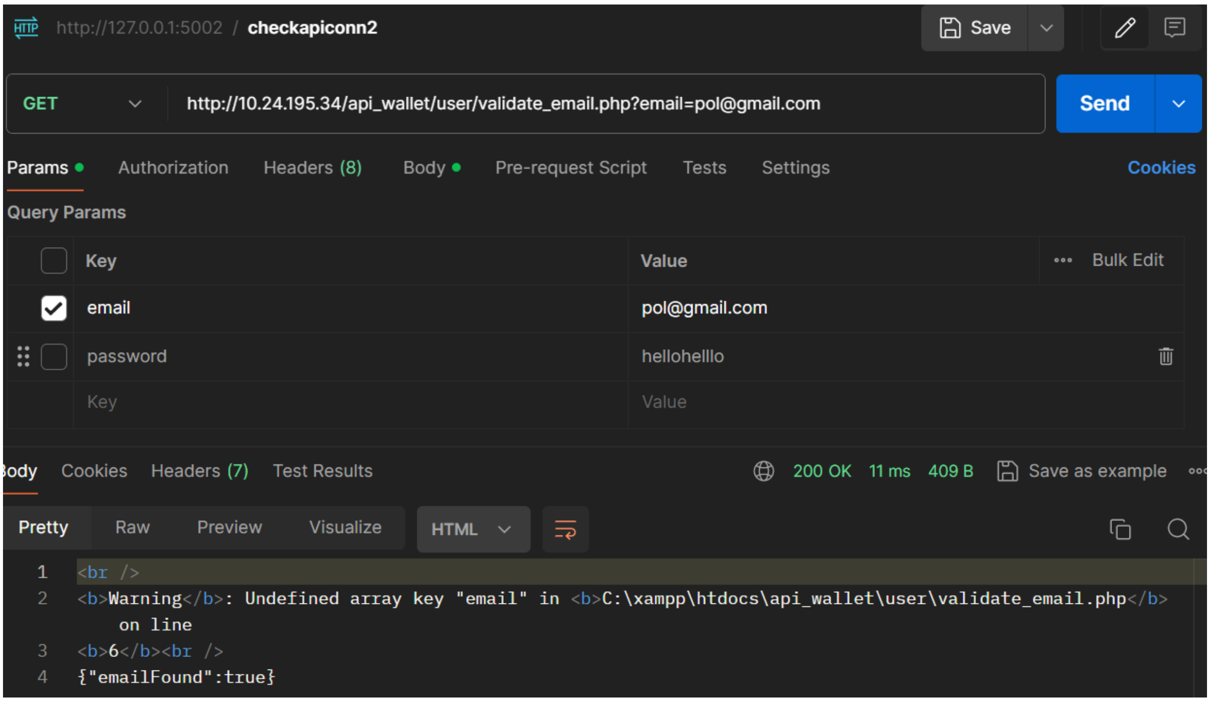
This is a continuation from the top wallet page (wallet).
The option to logout is provided within the "Options" tab, increasing security by allowing users to securely log out of their wallet when needed. The icon next to the "Logout" text ensures that users can quickly recognise its meaning without ambiguity.


User
Frontend
Backend
Database
Enters login credentials
Sends login request with credentials
Queries database for user credentials
Returns user data
Sends authentication result
Displays login success/failure message
User
Frontend
Backend
Database


User
Frontend
Backend
Initiates logout action
Sends logout request
Logs out user
Clears user session
Confirms logout
User
Frontend
Backend


Start
Home Page
User wants to log in?
Yes
No
Log in page
User wants to sign up?
Yes
No
Is user credentials right?
Yes
Yes
Sign up page
No
Imported wallet page
Email already used?
No
Is pkShare right?
Wallet created page
Yes
Wallet page
Logout?
No
Yes
No
Log out and authenticate user
Send transaction?
Yes
End
Send transaction

Endpoint testing:



GET http://10.24.195.34/api_wallet/user/validate_email.php?email=pol@gmail.com

	Key	Value
☑	email	pol@gmail.com
□	password	hellohellio
	Key	Value

200 OK 11 ms 409 B

```
<br />
<b>Warning</b>: Undefined array key "email" in <b>C:\xampp\htdocs\api_wallet\user\validate_email.php</b>
    on line
<b>6</b><br />
{"emailFound":true}
```

Login

Input (email & password)	Expected Output	Actual Output	Justification for test

E: pol@gmail.com
P: hellohello
	success => True
	userData => email = pol@gmail.com
	password = 23b431acfeb41e15d466d75de822307c
	shareofprivatekey = 028527ab5326
	public_key = 0x7e7ce4ef08d3d5b9cc025b808ccc3d34860e9de9
		Undefined array key 'email'
		Undefined array key 'password'
			See if successful logins work as expected

E: pol@gmail.com P: hello	success => False	Undefined array key 'email' Undefined array key 'password'	See if invalid password is handled
E: p@gmail.com P: hellohello	success => False	Undefined array key 'email' Undefined array key 'password'	See if invalid email is handled

E: P:	success => False	Undefined array key ‘email’ Undefined array key ‘password’	See if blank entries are handled
E: P: hellohello	success => False	Undefined array key ‘email’ Undefined array key ‘password’	See if blank email is handled
E: pol@gmail.com P:	success => False	Undefined array key ‘email’ Undefined array key ‘password’	See if blank password is handled

Sign Up

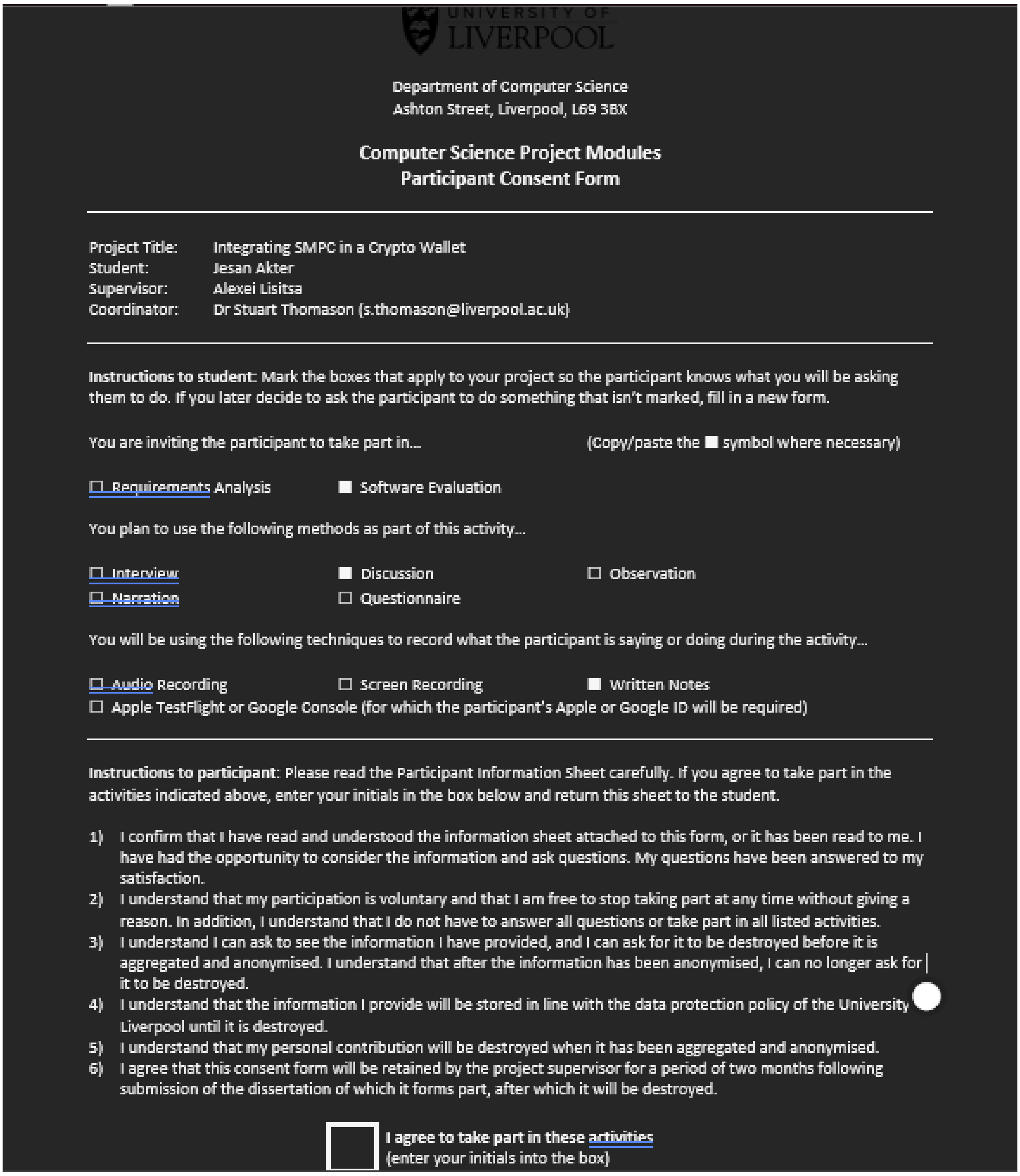
Input (email & password)	Expected Output	Actual Output	Justification for test
E: plo@gmail.com P: hellohello	success => True	Undefined array key ‘email’ Undefined array key ‘password’ Undefined array key ‘shareofprivatekey’ Undefined array key ‘public key’	See if successful sign-up work as expected
E: plo@gmail.com P: hellohello	Uncaught mysqli_sql_exception: Duplicate entry '' for key 'email' in	Expected	See if duplicate entries are accepted
E: P:	success => False	Undefined array key ‘email’ Undefined array key ‘password’ Undefined array key ‘shareofprivatekey’ Undefined array key ‘public_key’	See if blank entries are handled
E: P: hellohello	success => False	Undefined array key ‘email’	See if blank email is handled

		Undefined array key ‘password’ Undefined array key ‘shareofprivatekey’ Undefined array key ‘public_key’	
E: pol@gmail.com P:	success => False	Undefined array key ‘email’ Undefined array key ‘password’ Undefined array key ‘shareofprivatekey’ Undefined array key ‘public key’	See if blank password is handled

Validate email

Input (email)	Expected Output	Actual Output	Justification for test
E: pol@gmail.com	emailFound: true	Expected	See if it validates an email that exists in database
E: polll@gmail.com	emailFound: false	Expected	See if it validates an email that doesn’t exist in database
E:	emailFound: false	Expected	See if it validates an empty email

Example Participant Consent Form:



UNIVERSITY OF LIVERPOOL

Department of Computer Science
Ashton Street, Liverpool, L69 3BX

Computer Science Project Modules
Participant Consent Form

Project Title:	Integrating SMPC in a Crypto Wallet
Student:	Jesan Akter
Supervisor:	Alexei Lisitsa
Coordinator:	Dr Stuart Thomason (s.thomason@liverpool.ac.uk)

Instructions to student: Mark the boxes that apply to your project so the participant knows what you will be asking them to do. If you later decide to ask the participant to do something that isn't marked, fill in a new form.

You are inviting the participant to take part in… (Copy/paste the ■ symbol where necessary)

□ Requirements Analysis ■ Software Evaluation

You plan to use the following methods as part of this activity…

□ Interview ■ Discussion □ Observation
□ Narration □ Questionnaire

You will be using the following techniques to record what the participant is saying or doing during the activity…

□ Audio Recording □ Screen Recording ■ Written Notes
□ Apple TestFlight or Google Console (for which the participant's Apple or Google ID will be required)

Instructions to participant: Please read the Participant Information Sheet carefully. If you agree to take part in the activities indicated above, enter your initials in the box below and return this sheet to the student.

1) I confirm that I have read and understood the information sheet attached to this form, or it has been read to me. I have had the opportunity to consider the information and ask questions. My questions have been answered to my satisfaction.
2) I understand that my participation is voluntary and that I am free to stop taking part at any time without giving a reason. In addition, I understand that I do not have to answer all questions or take part in all listed activities.
3) I understand I can ask to see the information I have provided, and I can ask for it to be destroyed before it is aggregated and anonymised. I understand that after the information has been anonymised, I can no longer ask for it to be destroyed.
4) I understand that the information I provide will be stored in line with the data protection policy of the University Liverpool until it is destroyed.
5) I understand that my personal contribution will be destroyed when it has been aggregated and anonymised.
6) I agree that this consent form will be retained by the project supervisor for a period of two months following submission of the dissertation of which it forms part, after which it will be destroyed.

□ **I agree to take part in these activities**
(enter your initials into the box)

Each participant inputted their initals.

User feedback:
User 1

1. User Interface Opinion: Interface was sleek and intuitive. The design was clean and is easy to navigate between different sections and pages.

2. Confidence in Security Measures: I felt quite confident about the security measures implemented in the application, particularly when it came to handling private keys. The option to split the private key and store it securely gave me peace of mind.
3. Working functionality: Yes, all the features and functionalities I expected were available and seemed to work as intended. It was a smooth experience.
4. Future Features: I was impressed with the existing features, but one thing I'd like to see added is support for additional cryptocurrencies. This would make the wallet more versatile and appealing to a boarder range of users.

User 2

1. User Interface Opinion: Found it user-friendly but mentioned that the ‘Next’ option in the wallet created page needs to be more visible or replace it with another button.
2. Confidence in Security Measures: My private key share was too big for me, this made it harder to extract and store it safely. However, I do like the fact that private is taken out when someone logs out.
3. Working functionality: I was able to sign up, log in, manage my wallet, view balances, and send transactions without encountering any major issues. However, it would be beneficial it there was an easier way to go back to the wallet page from the send tokens page.
4. Future Features: Easier navigation from send page to wallet page, maybe add a button that can easily go back. Also, add an activity tab to see the transactions that occurred.