G52GRP Final Group Project
Project title: Blockchain-based Dead Man's Switch

Group No: O

Student Name	Student ID	SOC Username
Carmel Natasha Barnabas	20509430	hcycb2
Anshana Manoharan	20506329	hcyam4
Lai Ken Siang	20409289	hfykl13
Adyan Dean bin Wafdi Kamil	20413774	hfyaw2

Supervisor: Dr Ioini El Nabil

Date: 02nd May 2024

Contents

1 Background

The evolution of the internet can be broadly categorised into two phases: Web2 and Web3. Web2, describes the current state of the internet where users consume and contribute content through centralised platforms such as social media, search engines, and e-commerce websites.

1.1 Web2: Overview and Limitations

Web2 connects billions of users globally, offering unmatched convenience and connectivity. However, it comes with drawbacks, notably centralisation, where a handful of powerful corporations control the user data. For instance, Microsoft, with its suite of services like Windows and LinkedIn, collects user data, including interaction patterns and productivity metrics.
This centralisation leads to issues such as data privacy concerns, censorship, and the exploitation of user data for profit. Furthermore, it introduces vulnerabilities to hacking, data breaches, and manipulation of information. Users have limited control over their personal data, often entrusting it to third-party entities (i.e. the corporations) without full transparency or ownership rights.

1.2 Web3: Addressing Web2's Shortcomings

Enter Web3, the next generation of the internet designed to address the shortcomings of Web2 and empower users with greater control, privacy, and security (Ivanov and Pashkov 2023). At its core, Web3 is built upon decentralised protocols, peer-to-peer networks, and cryptographic principles, offering a paradigm shift from centralised to decentralised systems.
Blockchain technology serves as the foundational infrastructure of Web3, providing a decentralised and unchangeable log for recording transactions and storing data. Unlike traditional databases controlled by centralised authorities, blockchain operates on a distributed network of computers, ensuring transparency, security, and censorship resistance.

1.3 Key Innovations of Web3

A prominent application of blockchain technology is cryptocurrency. It enables peer-to-peer financial transactions without the need for intermediary centralised corporations such as banks or payment processors. Users retain full ownership and control of their digital assets through cryptographic keys, eliminating the risks associated with centralised financial systems.

1.3.1 Identities

One of the key innovations of Web3 is the concept of self-sovereign identity, where users have full control over their digital identities and personal data. Decentralized identity solutions utilise cryptographic techniques to verify identity without relying on centralized authorities, enhancing privacy and reducing the risk of identity theft.

1.3.2 Crypto Wallets

Users manage their digital assets through cryptocurrency wallets, which store cryptographic keys granting access to funds. Each wallet is associated with a unique public key visible on the blockchain, while the private key remains accessible solely to the owner, facilitating secure transactions. Hence, these wallets provide users with full control over their assets and identities, mitigating the risks associated with centralised systems.

1.3.3 Smart Contracts

In Web3, smart contracts are autonomous and self-executing contracts deployed on blockchain networks. Smart contracts play a crucial role in enabling interactions and processes. These programmable contracts execute predefined actions when specific conditions are met, eliminating the need for intermediaries, and ensuring transparency and fairness in transactions.

In summary, Web3 represents a fundamental shift towards a more distributed, secure, and user-centric internet. By leveraging blockchain technology, cryptocurrencies, and decentralised protocols, Web3 aims to empower individuals with greater control over their data, finances, and digital identities, paving the way for a more inclusive and equitable digital future.

2 Project Overview

2.1 Problem

The following user story encapsulates the core objective of our project.

> *"As a holder of sensitive data concerned about the secure transfer of my assets to my beneficiaries in the event of my demise, I want a solution that automatically triggers the release of assets to my beneficiaries without compromising the security of my private key. This should ensure that my beneficiaries can access their entitled assets securely and eliminate the risk of unauthorized access and fraudulent activities."*

A potential conventional solution for this problem would be sharing the benefactor's private key with the beneficiaries. Despite being functional, it poses significant security risks; unauthorized access or theft of the private key could result in severe consequences.

2.2 Solution

The aforementioned security dilemma becomes particularly challenging in cases where the owner passes away, as access to the private key becomes inaccessible, thereby hindering beneficiaries from accessing their entitled assets.
To address this issue, we propose a solution inspired by a real-life dead man's switch, tailored for blockchain storage systems.

2.2.1 Dead Man's Switch

A dead man's switch is a switch designed to halt all processes of a machine/system in the case where the human operator is no longer able to operate the machine. It is found on exercise machines such as treadmills or Stairmasters but is commonly found on assistive technology such as wheelchairs or on a train's vigilance device (Macii and Poncino 2014).

2.2.2 Implementing the Dead Man's Switch

Our solution aims to provide a mechanism for securely transferring assets to designated beneficiaries in the event of the owner's incapacitation or demise using a dead man's switch. This involves the implementation of smart contracts within the blockchain network. In the case of the owner's

inactivity or inability to access the wallet for a specified period, the smart contract triggers the release of assets to predefined beneficiaries, without compromising the security of the private key.

2.2.3 Security

All assets uploaded by the owner (benefactor) must be encrypted and the benefactor is required to hand over the decryption key to the beneficiary upon election. However, our application provides the service of generating the decryption key securely (refer to Section 5.2). By leveraging the transparency and immutability of blockchain technology, our solution ensures the integrity and security of asset transfer processes while mitigating the risk of unauthorized access and fraudulent activities.

2.2.4 Storage

All assets uploaded by the benefactor are to be securely stored in Inter-Planetary File System (IPFS) utilising the Pinata gateway. IPFS operates on a decentralised network, ensuring robustness and accessibility by distributing data across multiple nodes rather than relying on centralised servers. Pinata, a service built on top of IPFS, provides an interface for managing content on the IPFS network. This approach not only enhances data security but also promotes accessibility and integrity, aligning with the principles of decentralised applications.

2.3 Project Insight

In addition to providing a detailed explanation of the proposed solution, this report will delve into the technical aspects of its implementation, including the design of smart contracts, integration with existing blockchain infrastructure, designs of our application, and considerations for scalability and interoperability.
Furthermore, we will discuss potential challenges encountered during the development process and outline areas aimed at enhancing the efficiency and effectiveness of our solution in facilitating secure asset inheritance within blockchain ecosystems.
In this report, we will discuss existing solutions, explain the technologies that we used, our project implementation and a conclusive reflection.

2.4 Literature Review

This digital solution, akin to a dynamic digital living will, constitutes a novel approach to safeguarding personal data and digital assets in the event of the owner's incapacity or death. Such systems, often termed "dead man's switches," activate pre-defined actions upon the owner's absence, ensuring the secure and responsible management of their digital legacy.Furthermore, the decentralised nature of blockchain networks presents significant advantages in terms of censorship resistance (Merre 2020).

Unlike centralised servers, blockchain networks operate without a single point of control, minimising the risk of data manipulation or access restrictions. This inherent resilience makes decentralised dead man's switches particularly suitable for protecting sensitive data and ensuring the execution of the owner's wishes, even in challenging or unpredictable circumstances.

2.4.1 Existing Solutions

One existing solution is Sarcophagus. It is an unaudited decentralised dead man's switch built on blockchain technology. Sarcophagus adopts a dual encryption method. Although it ensures controlled access to the assets upon the benefactor's incapacitance and stores all the assets in a decentralised manner, it does require a third party operator. This operator is in charge of decrypting the assets once the benefactor is incapacitated. Only after the operator decrypts the assets can the beneficiaries decrypt the assets completely. This method is risky as it relies on third party interactions. Furthermore, Bequest Finance is another existing solution that performs the necessary operations for digitised will management. But the only issue is that it stores the users' data on Google Data. This means it relies on centralised storage.

These solutions require the benefactor to log in periodically and prove that their are indeed alive. Instead, our solution will be minimising unnecessary user interaction (Pereira et al. 2017) by'requiring beneficiaries to trigger the dead man's switch and only then will a countdown begin for the benefactor to disable.

As mentioned above, our application will utilise IPFS storage system to store the assets in a distributed manner, as well as simplify the decryption process to ensure robust controlled access to the assets at the right time.

2.5 Project Aims

The aims of this project are as follows:

1. Develop an application based on blockchain technology to facilitate secure and efficient transfer of personal data ownership to designated beneficiaries upon the benefactor's passing, ensuring the integrity of private keys.
2. Create a user-friendly interface allowing benefactors to assign and prioritize beneficiaries, and enabling beneficiaries to securely access their inheritance.
3. Integrate a dead man's switch mechanism into the application to ensure timely release of personal data in the event of the benefactor's incapacitation or demise, without compromising the security of private keys.
4. Prioritize blockchain implementation over server-side scripting to bolster security and decentralisation to minimise vulnerabilities and enhance overall system integrity.

2.6 Project Objectives

The objectives of this project are as follows:

1. Facilitate seamless connectivity by integrating users' MetaMask wallets with the application to ensure secure transactions and interactions within the blockchain framework.
2. Develop and deploy a smart contract system allowing benefactors to efficiently manage beneficiary registration and priority levels to optimise inheritance distribution processes.
3. Implement a decentralized storage solution to safeguard personal data, leveraging blockchain technology for enhanced security and privacy measures.

4. Provide benefactors with standard functionalities for managing personal data, including uploading, downloading, and viewing files, ensuring user-friendly access and control over assets.

5. Incorporate a dead man's switch mechanism to automatically trigger asset distribution in the absence of benefactor activity for a specified period to ensure timely inheritance delivery.

6. Design an intuitive user interface enabling benefactors to deactivate the countdown mechanism if they are alive, ensuring seamless user control and interaction.

7. Deploy the application on a blockchain network, conducting comprehensive testing to ensure robust functionality, scalability, and security compliance.

3 Product Overview

Our final product, *dWill*, is a Progressive Web Application (PWA) designed for managing wills in a decentralised manner. The application caters to two distinct user roles:

- **Benefactor:** These users seek to allocate their files to pre-assigned beneficiaries.
- **Beneficiary:** These users are entrusted with a benefactor's files in the event of the benefactor's incapacitation.

dWill offers benefactors a subscription-based service, while beneficiaries follow a standard login procedure. Beneficiaries are required to provide both their benefactor's address and their own during login.

Benefactors have access to several functionalities within the platform, including the ability to assign beneficiaries, encrypt files, upload files, assign these files to beneficiaries, and disable the dead man's switch. Conversely, beneficiaries have the authority to enable the dead man's switch. Upon activation, if no response is received from the benefactor within *7 days*, beneficiaries gain access to the assigned files.

This configuration guarantees a streamlined and secure procedure for both benefactors and beneficiaries, simplifying the management and distribution of digital assets in unforeseen circumstances.

4 Methodology

4.1 Requirements Specification

4.1.1 Functional Requirements

1. **Benefactor subscription**: the application should allow benefactors to register their wallet and pay the subscription fee in 6 month installments.

2. **Beneficiary log-in:** the application should allow beneficiaries to log in without any subscription payments.

3. **Beneficiary management**: Benefactors should be able to elect beneficiaries.

4. **Data storage**: encrypted files should be stored on a decentralised storage system.

5. **Encrypted files**: files must be encrypted before uploading and assigning to beneficiaries.
6. **Common encryption and decryption key**: benefactor and beneficiaries should be able to use the common secret key for encryption and decryption.
7. **Dead man's switch**: beneficiaries should be able to trigger the dead man's switch if they suspect the benefactor is deceased.
8. **Access control**: beneficiaries should only gain access to data after the confirmation of the benefactor's passing.
9. **Authentication**: all triggers and interactions with the application should be authenticated with user's wallet.

4.1.2 Non-functional Requirements

1. **Security**: the application should be secure and protect user data from unauthorised access, modification, or deleted.
2. **Privacy**: user data should be kept private and confidential so ensure that it is uploaded into the decentralised file storage securely.
3. **Availability**: the application should be highly available and accessible to users at all times.
4. **Performance**: the application should handle transactions efficiently.
5. **Scalability**: the application should be able to accommodate a growing number of users and data.
6. **Usability**: the application should be user-friendly for both benefactors and beneficiaries.
7. **Maintainability**: the application code should be well-structured and easy to maintain.

4.2 Design

4.2.1 User Interface (UI)

As dWill operates on blockchain technology, our priority was to design a user interface that seamlessly integrates with users' experiences. We consciously avoided the use of complex blockchain terminology or jargon, opting instead for a straightforward presentation of information and visual elements. Refer to Figure . Our aim is to ensure that users can effortlessly navigate the platform without encountering any unnecessary technical barriers. Because users already need to have MetaMask wallets and need to know how to use them, we prefer to keep other sections of our application very intuitive, further enhancing the user experience.

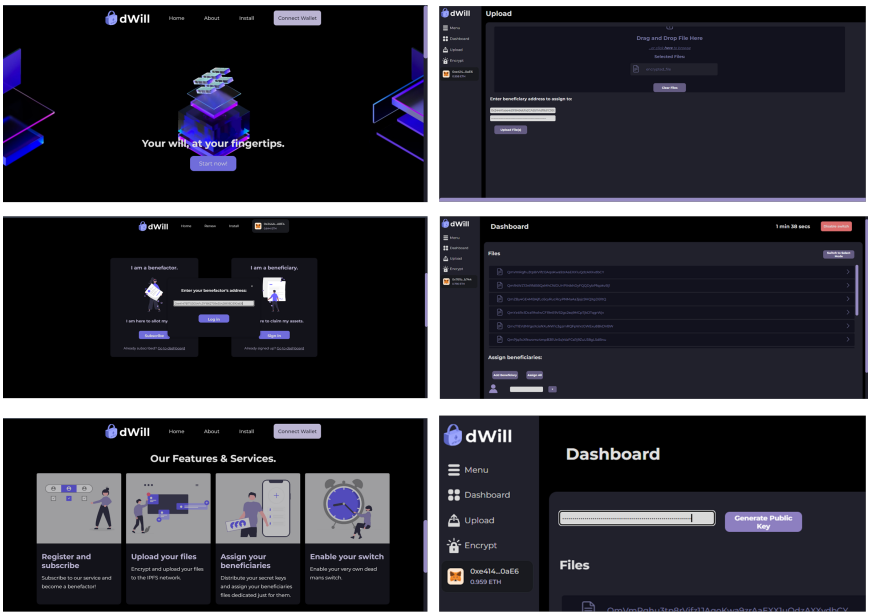

Figure 1: This figure presents the final design of our application.

4.2.2 Use Case Diagram

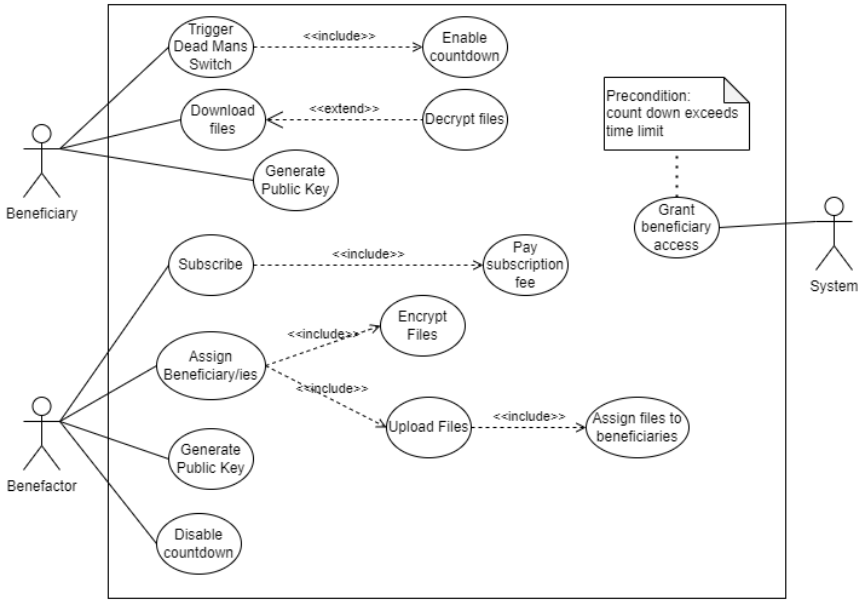

Figure 2: This use case diagram describes the basic interactions between the system, benefactor and beneficiary.

4.2.3 Activity Diagram

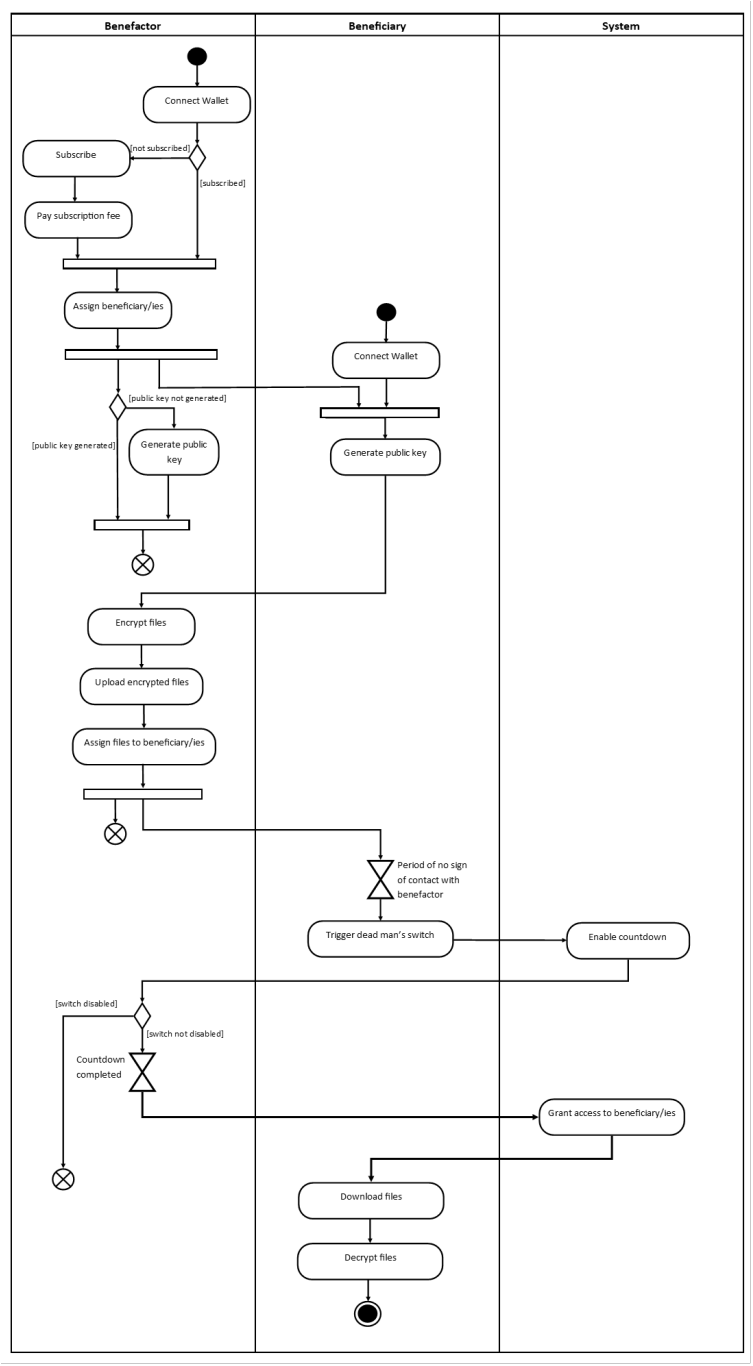

Figure 3: This activity diagram portrays the complete flow of events between the system, benefactor and beneficiary.

4.2.4 Architecture Diagram

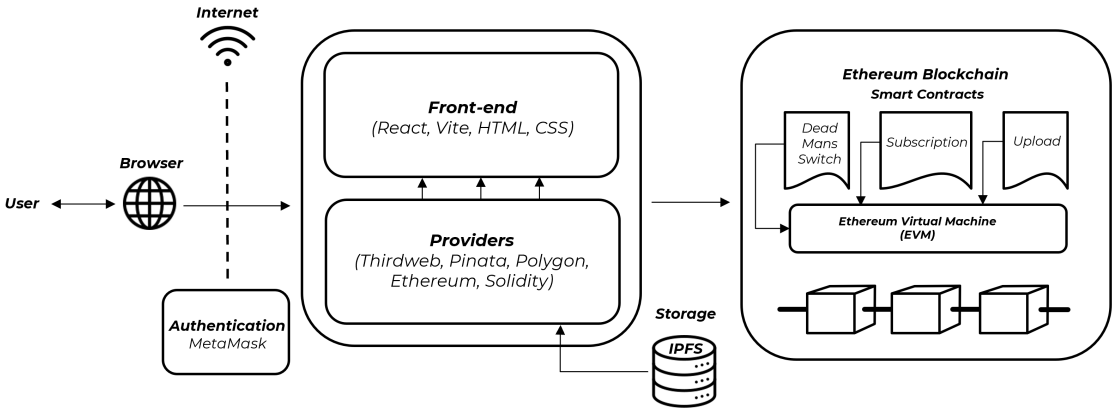

Figure 4: This architecture diagram represents the structure of how the dWill application was built.

5 Implementation

5.1 Smart Contracts

5.1.1 Subscription

The *SubscriptionContract* is a smart contract developed for managing subscription-based payments. This smart contract facilitates the registration of benefactors, subscription to dWill services, validates subscription status and allows for renewal.
This contract was written by the group.

- Contract Structure
    1. **Owner:** The contract is deployed by the owner who has control over retrieving the funds sent to the contract when benefactors pay the subscription fee.
    2. **Subscription Parameters:** Key parameters such as subscription amount, first payment amount, duration and grace period are defined.
    3. **Subscriber Structure:** The contract maintains subscriber details including the last payment timestamp and subscription status.
- Functions:
    1. **checkSubscriptionStatus:** Checks the subscription status of an address and returns the status code (i.e. valid, expired, requires renewal, or new subscriber).
    2. **subscribe:** Handles the benefactor's subscription payments, validates payment amounts, transfers funds, and updates subscriber status.
    3. **withdrawBalance:** Allows the owner to withdraw the contract balance.
    4. **unsubcribe:** Enables subscribers to unsubscribe by resetting their subscription status.

5. **receive:** Fallback function to receive Ether payments.

- Subscription Workflow

1. **Subscription:** Benefactors subscribe to dWill services by sending the required subscription amount to the contract.
2. **Payment Validation:** The smart contract validates payment amounts based on the subscription status (expired, requires renewal, or new subscriber).
3. **Renewal:** Renewal payments are processed within the grace period to maintain uninterrupted service.
4. **Unsubscription:** Benefactors can unsubscribe from dWill services, restoring their subscription status.

5.1.2 Dead Man's Switch

The *DeadManSwitch* contract is a crucial component of dWill, enabling benefactors to implement a dead man's switch mechanism with assigned beneficiaries. This smart contract ensures that the benefactor's assets can be securely transferred to designated beneficiaries in the event of the benefactor's inactivity or demise.
This contract was written by the group.

- Contract Structure

1. **Benefactor and Beneficiary Structures:** The smart contract defines structures to store benefactor and beneficiary information, including IPFS CIDs assigned to the beneficiaries.

- Functions

1. **setBenefactor:** Allows benefactors to set up their accounts.
2. **removeBenefactor:** Enables benefactors to remove their accounts from the system.
3. **addBeneficiary and removeBeneficiary:** Functions for benefactors to manage their beneficiaries.
4. **addIpfsCID and removeIpfsCID:** Functions for benefactors to assign or revoke IPFS CIDs from beneficiaries.
5. **enableSwitch and respondToSwitch:** Mechanisms for beneficiaries and benefactors to enable and respond to the dead man's switch, respectively.
6. **checkAliveStatus:** Checks the alive status of the benefactor based on their response to the dead man's switch.
7. **getCIDs:** Allows beneficiaries to access assigned IPFS CIDs only upon benefactor's incapacitance.

- Workflow

1. **Benefactor Setup:** Benefactors initialize their accounts and configure switch settings using `setBenefactor`.
2. **Beneficiary Management:** Benefactors manage their list of beneficiaries by adding or removing them using `addBeneficiary` and `removeBeneficiary`.

3. **IPFS CID Assignment:** Benefactors assign IPFS CIDs to beneficiaries using `addIpfsCID` and can remove them if necessary.
4. **Switch Activation and Response:** Beneficiaries activate the dead man's switch using `enableSwitch`, and benefactors respond to it using `respondToSwitch` to confirm their activity.
5. **Asset Transfer:** If the benefactor fails to respond within the specified duration (i.e. 7 days), assets are automatically transferred to the designated beneficiaries.

5.1.3 Uploading files

The *Upload* contract is a crucial component in dWill, allowing users to upload their files or assets and manage access permissions to their data.
This contract referred to a source contract: https://github.com/kshitijofficial/Dgdrive3.0

- Contract Structure
  1. **Access Structure:** Defines a structure to store user access information, including the user's address and their access status.
  2. **Mappings:**
     - **value:** Maps user addresses to arrays of URLs, storing the uploaded data.
     - **ownership:** Maps user addresses to another mapping representing ownership of data between users.
     - **accessList:** Maps user addresses to arrays of access structures, representing access permissions granted to other users.
     - **previousData:** Maps user addresses to another mapping to track previous data ownership.
- Functions:
  1. **add:** Allows users to add assets to their data storage.
  2. **allow and disallow:** Functions for users to grant or revoke access to their data for their beneficiaries.
  3. **display:** Enables users to display their uploaded data, subject to access permissions.
  4. **shareAccess:** Provides the beneficiaries with a view of their access control list.
- Workflow
  1. **Data Upload:** Benefactors upload data by adding URLs using the `add` function.
  2. **Access Management:** Benefactors manage access to their data by granting (`allow`) or revoking (`disallow`) access to other users (beneficiaries).
  3. **Data Display:** Benefactors can display their uploaded data using the `display` function, subject to access permissions.
  4. **Access Control List:** Benefactors can view their access control list using the `shareAccess` function, which provides insights into who has access to their data.

5.2 Encryption

Due to the transparent nature of blockchain transactions, all data passed to and from smart contracts is susceptible to global access by participants in the blockchain network. As dWill necessitates storing IPFS file hashes in smart contracts, there is a risk of unauthorized access by beneficiaries or other users. To mitigate this risk, it is necessary to encrypt the hashes before storing them in the smart contract.

However, encryption introduces the challenge of securely managing decryption keys. The key must be safeguarded to prevent unauthorized access, yet it must also be readily accessible for legitimate storage and retrieval operations. Traditional storage methods are vulnerable to hacking attempts.

To address this challenge, we propose implementing the Diffie-Hellman Key Exchange Algorithm. This algorithm utilises the user's private key to generate a corresponding public key, enabling secure key exchange without the need to store the key itself. By leveraging this algorithm, we enhance the security of our system while ensuring efficient key management.

5.2.1 Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman Key Exchange Algorithm is a method for securely exchanging cryptographic keys over a public channel. It allows two parties to agree upon a shared secret key without explicitly transmitting the key itself. Here's how it works:

1. **Setup:** Before any communication takes place, the benefactor and beneficiaries agree on certain parameters, which are publicly known. These parameters include a large prime number p (set as 23 in our context) and a primitive root modulo g (set as 5 in our context). These parameters are common to both parties but can be publicly known without compromising security.

2. **Key Generation:** Both the benefactor and the beneficiaries choose their private keys.

   - Benefactor selects a private key a.
   - Beneficiary selects a private key b.

   Note that while a private key is necessary for the operation, it need not be the user's true wallet's private key, as using the wallet's private key contradicts the fundamental purpose of developing this application (refer section 2.1).

3. **Public Key Calculation:** Using the private key and the agreed-upon parameters, each party calculates their corresponding public key:

   - Benefactor calculates their public key as $A = g^a \bmod p$
   - Beneficiary calculates their public key $B = g^b \bmod p$

4. **Exchange of public keys:** This algorithm requires both parties to exchange their public keys over a public channel. In our context, the public keys are stored in the smart contract and can be accessible publicly.

5. **Shared Secret Calculation:** The benefactor uses the beneficiary's public key and their own private key to calculate the shared secret key. The same process is carried out by the beneficiary but with the benefactor's public key and their own private key:

- Benefactor calculates the shared secret key $S = B^a \bmod p$
- Beneficiary calculates the shared secret key $S = A^b \bmod p$

Due to the mathematical properties of exponentiation and modulo arithmetic, both the benfactor and beneficiary will arrive at the same shared secret S. This shared secret can be used to encrypt and decrypt hashes as well as the files before uploading into IPFS.[1]

5.3 Integration

The thirdweb SDK provided boilerplate code to implement the PWA. We utilised this code to provide the features of the PWA and adapted it to the needs of the application by enabling the PWA to be accessible as an app and as a webiste rather than the webiste serving as a download platform for the app.

Figure 4 shows our source code hierarchy or in simpler terms, our file structure. With all of our dependencies stored in package.json and yarn.lock, the bulk of our code is stored within the src directory (i.e. page layouts, smart contracts etc)[2]

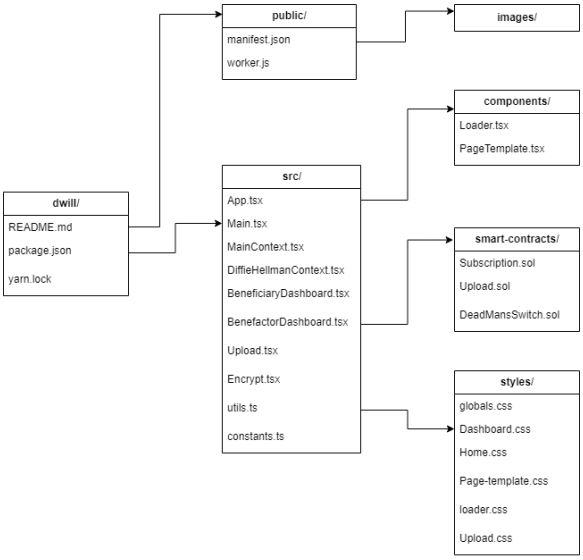

Figure 5: Overview of our source code hierarchy. Boxes represent the folders while arrows represent directory inclusion.

5.4 Notifications

In the initial stages of dWill's development, we implemented a system where users were prompted to input their email addresses and Discord usernames for the purpose of receiving notifications via

[1]The benefactor must compute their public key before uploading, encrypting or assigning files. Whereas, the beneficiaries must compute their public key immediately upon assignment as their public key is required by the benefactor to upload and assign encrypted files.

[2]The Upload.tsx, Loader.tsx and Loader.css are the only codes that were adapted from other sources. The rest of the code were completely written by the group members.

Tenderly. Tenderly, a Web3 gateway and development platform, offers real-time alerting services distinct from other conventional Web3 tools. Through Tenderly, users can configure alerts to be triggered by specific events within a smart contract, facilitating timely notifications.

However, as we progress with the integration of notification services within dWill, a critical consideration emerges regarding user data management. Storing user information such as email addresses and Discord usernames within a smart contract, while feasible, introduces a heightened level of reliance on dWill's infrastructure also leaning towards a 'centralised' approach. Given dWill's commitment to empowering users with comprehensive control over their data, this approach conflicts with our core objective.

To align with our commitment to user-centric data governance, we suggest for an alternative approach: encouraging users to directly set up alerts using Tenderly. By doing so, users retain full control over their data, leveraging Tenderly's robust notification capabilities without compromising their independence. This approach not only upholds our principle of user empowerment but also ensures a seamless and secure experience within dWill.

6 Testing

The development process of dWill involved a systematic approach to quality assurance. This bound several phases of testing to ensure the reliability and functionality of the application.

6.1 Unit testing

Firstly, unit testing was conducted on the smart contracts. This phase aimed to verify the correctness of individual contract functionalities. Refer to Section 10.2.1) for the relevant test suites. Note that in any case where an invalid account address was input, the front-end will produce a relevant message to the user.

6.2 System testing

Subsequently, system testing was carried out to assess the dWill's adherence to the specified requirements. Through this phase, the overall behavior and performance of the application were evaluated, ensuring its alignment with the functional expectations. Refer to Section 10.2.2) for the relevant test suites.

6.3 Beta testing

Lastly, beta testing was carried out, involving engagement with a group of 12 blockchain users. This phase served to gather invaluable feedback on usability, performance, and user experience. Insights obtained from beta testing were fundamental in refining the application and addressing any identified issues or areas for improvement.

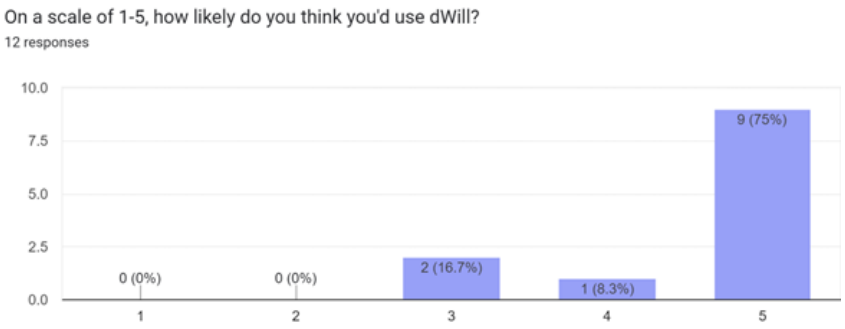

Figure 6: This figure presents some of the results from the beta testing survey.

6.3.1 Feedback Analysis

The following beta testing feedback provides valuable insights for refining the application to better meet user expectations and ensure a smoother user experience. Refer Figure 5 for general overview.

- **Likelihood of Usage**
  Users highlighted the app's security features and potential for secure file transfer. They also noted the simplicity and user-friendliness of the interface. The application was recognised for the digitisation and ease compared to traditional methods. Interest was expressed for future use but concerns were raised for when beneficiaries could be children.

- **Liked Features**
  Testers praised the clean and sleek UI design and appreciated the simplicity as well as real-world utility. Positive remarks were made for the dark mode, fonts, and navigation.

- **Areas for Improvement**
  Additional features like storing cryptocurrencies were suggested. There were concerns about gas fee payments. A user suggested for guides so the functionality of the application could be made more comprehensive.

- **Performance Issues**
  Loading animation occasionally was not present when required.

Collectively, these testing initiatives reflect a comprehensive quality assurance process, reinforcing the reliability and efficacy of the developed application.

7 Project Management

Our project adopted the **Agile** development model, adopting the Scrum methodology to ensure an effective approach to software development.

7.1 Mini-scrum Implementation

Scrum is a widely used Agile project management framework that ensure flexibility, team collaboration, and frequent feedback through iterative development cycles. Mini-scrum is ideal for small startups project with a smaller team. Since this perfectly aligns with our group's structure and our project's goals, we adopted the usage of Mini-scrum methodology. This section discusses about how our group utilised Mini-scrum, including role assignment among team members and sprints.

7.1.1 Role assignment

- **Carmel Natasha Barnabas:**
  Carmel was assigned the role of Product Owner who represented the stakeholders' interests, managing the product backlog and ensuring the application precisely aligns with users' requirements. She was in charge of delegating tasks during sprint reviews.During sprint planning session, Carmel discussed what tasks went well, what didn't, and decided what to work on in the upcoming sprint. Carmel took the lead in organising sprint planning meeting, and ensured that these meetings were productive and focused. During the Open Day, while presenting our project to industry representatives and examiners, Carmel's ability to articulate the product's value, answer questions and gather feedback was crucial to align our project with market opportunities. In addition to being the Product Owner, she also partook in major software development tasks and research as well. She researched and experimented with various file storage systems. Her proposed application of Pinata and IPFS was used in our final product.

- **Anshana Manoharan:**
  Anshana was assigned the role of Scrum Master. She ensured the group followed Agile practices and solved hurdles during the software development process, in terms of technical roadblocks, team dynamics, inefficient workflows, etc. As the Scrum Master, she played a vital role. For instance, she ensured that all discussions from meetings were recorded. This helped track the group progress and review past decisions efficiently. Besides this, the Scrum Master's central role is to identify and resolve obstacles faced by team members. She always comes out with solutions to resolve issues faced, ensuring our team environment fosters collaboration and efficiency. In addition to this role, she also was in charge of drafting out diagrams during the initial stages of development. Her involvement in developing the software was crucial because she implemented the Dead Man's Switch Mechanism and and initally proposed using the Diffie Hellman Key Exchange algorithm as an alternative to handing decryption keys physically. This algorithm was deemed extremely vital to our application because it was utilised for encrypting the IPFS hashes upon storing in the smart contract. She also experimented with implementing gas-less transactions and upon her research, she proposed using the thirdweb SDK, which proved to provide easy integration of wallets. She also proposed and oversaw the use of collaborative tools like Microsoft Planner.

- **Lai Ken Siang:**
  Ken Siang was assigned as Technical Research Lead and Tester. Initially, our group struggled with the integration of blockchain technology, a new and complex field for all of us. To kickstart the development process, he experimented with implementing a React app with integration with Metamask cryptowallets and also developed a draft dead man's switch smart contract. His prototype involved a live countdown mechanism, a notification system using EmailJS and draft page flows, where the countdown was used in our final product. He was also involved in testing the application by performing unit testing on all the smart contracts. This ensured that our application was able to carry out the basic functionalities.

- **Adyan Dean bin Wafdi Kamil:**
  Adyan was assigned as Technical Research Lead and Tester. Since Ken Siang's approach to notifications were a centralised one, Adyan proposed using Tenderly's alert system which allowed us to design custom prompts and notifications. He also contributed by coding the retrieval of files from IPFS and downloading of the files. In addition to being the Technical Research Lead, he also was involved in testing the application by performing system testing after the final prototype was created. He ensured that our application met the practical needs and expectations of our users.

7.1.2 Sprints

- **Sprint Backlog:** The sprint backlog is a list of tasks identified to be completed during the sprint. Our group utilized tools such as Microsoft Planner to manage sprint backlog in an Agile or Scrum framework. Figure 5 shows the built-in charts in Microsoft Planner where team members can update the status of tasks as they progress and add comments to communicate issues or updates. This keeps the entire team informed about task progress and any potential blockers.

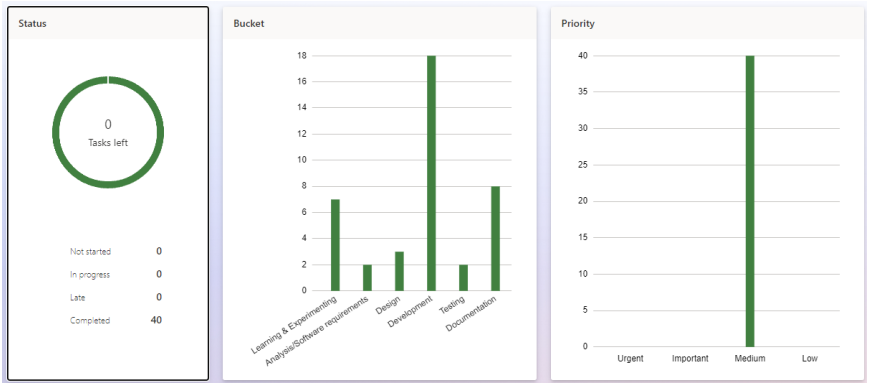

Figure 7: Overview of tasks progress across different categories

- **Sprint Planning:** In Sprint Planning of our group, we set clear and well-defined goals that were achievable. For example, Figure 6 shows the product backlog completed in first Sprint of our group. During the initial sprint of our group, we focus on researching and experimenting on blockchain technology and explore various methods to integrate them into our application. We recognized early the complexity behind the blockchain technology thus structured our sprint to prioritize learning and experimentation. By the end of the sprint, our group had successfully completed all planned tasks, as shown in Figure 6. The completion of all tasks in our first sprint provided valuable lessons that were carried forward into subsequent sprints.

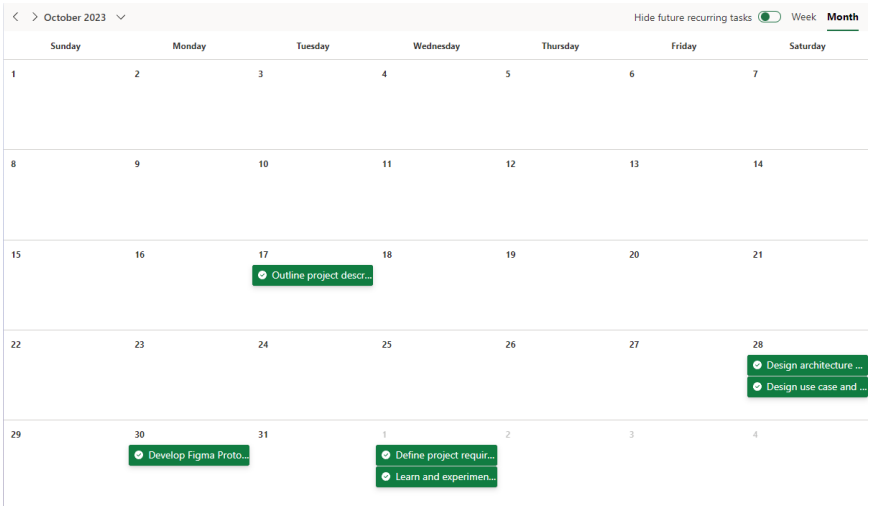

Figure 8: Overview of product backlog completed in first Sprint

- **Sprint Review:** Our Sprint Review sessions is conducted with our supervisor as checkpoints for our project's process to gather insights, suggestions, and feedback from supervisor to ensure our project is focus on the right track. Based on our meeting minutes11.3, we presented the work that has been completed, and demonstrated new features added to our application to our supervisor. The primary goal of the sessions is to bring valuable external perspective to the project. Figure 7 shows features on the Microsoft Planner where team members will marked off the task if the tasks assigned is completed. This feature is an effective tool for reporting our current status and for strategic planning discussions. The review of completed and incomplete tasks during Sprint Review helped us to make decisions about process adjustments needed to improve efficiency in next sprint.

Title	Assignment	Start Date	Due Date	Bucket	Progress	Priority
~~Experimenting with thirdweb SDK~~	Anshana Manohar.		12/11/2023	Learning & Experi	Completed	Medium
~~Find suitable APIs~~			12/5/2023	Learning & Experi	Completed	Medium
~~Design use case and activity diagrams~~	Anshana Manohar.		10/28/2023	Design	Completed	Medium
~~Design architecture diagram~~	Anshana Manohar.		10/28/2023	Design	Completed	Medium
~~Finalise dead mans switch, registration and subscripti...~~	Anshana Manohar.		1/25/2024	Development	Completed	Medium
~~Interim report editing~~			12/14/2023	Documentation	Completed	Medium
~~Interim report drafting~~			12/1/2023	Documentation	Completed	Medium
~~Download of files from IPFS using hash~~	Adyan Dean Bin W		3/27/2024	Development	Completed	Medium
~~Experimenting with tenderly notifications~~			12/12/2023	Learning & Experi	Completed	Medium
~~Experiment with integrating Metamask with React mo...~~			11/25/2023	Learning & Experi	Completed	Medium
~~Experiment with signer for encryption and decryption~~	Carmel Natasha Ba		4/2/2024	Learning & Experi	Completed	Medium
~~Learn and experiment with blockchain technologies~~			11/1/2023	Learning & Experi	Completed	Medium
~~Implement key exchange~~	Anshana Manohar.		4/6/2024	Development	Completed	Medium
~~Produce presentation content~~			4/7/2024	Documentation	Completed	Medium
~~Integrate live countdown~~			4/8/2024	Development	Completed	Medium
~~Outline project description~~			10/17/2023	Analysis/Software	Completed	Medium
~~Integrate back-end and front-end countdown functio...~~			4/5/2024	Development	Completed	Medium
~~Implement test dead man switch contract~~	Lai Ken Siang		11/30/2023	Development	Completed	Medium
~~Implement test subscription front-end~~	Lai Ken Siang		3/8/2024	Development	Completed	Medium

Figure 9: Overview of all completed product backlog

7.2 Iterative development

Following Agile principles, we adopted iterative development, breaking down our project into small, manageable iterations lasting one to four weeks. Each iteration yielded a working piece of software, allowing the scrum master and product owner to review and evaluate progress regularly.

7.3 Version Control

While developing dWill, version control played a crucial role in ensuring efficient collaboration and transparency of the project.

- **Github:** We adopted Github, a version control system familiar to all of us, to seamlesly develop our application. GitHub served as the primary platform for hosting the dWill project repository, providing a centralised location where the team could access the latest codebase and submit their contributions.
- **Branching:** A structured branching approach was put into place to efficiently oversee the development workflow. While feature branches were made to implement new features or fix problems, the master branch kept the application's stable version up to date. With this method, the team could work on discrete features without interfering with the main codebase. The team each branch out to add or fix features.
- **Pull Requests:** The contribution workflow relied on pull requests to propose changes to the codebase. All pull requests were subjected to code reviews by the contributors prior to merging into the master branch. This process ensured that proposed changes adhered to coding standards and addressed the intended objectives.
- **Merging:** The merge process followed a cautious approach to maintain code quality and stability. Merging approved pull requests into the master branch was limited to those that had passed all tests and required approvals. Furthermore, regular communication facilitated dispute resolution and guaranteed the smooth implementation of changes amongst contributors.
- **Commits:** In light of the app's modifications, each commit message that was sent had significance as well. This method made it easier to communicate changes and made version history tracking easier.

GitHub's version control system was essential to the development of dWill since it promoted teamwork, code integrity, and project stability. The project maintained an organised and structured development lifecycle by following established practices like branching, pull requests, and commits.

7.4 Reflection of Collaboration

As mentioned prior, our team had very minimal knowledge in the blockchain space, let alone creating a decentralized application. However, through consistent communication, dedication and active participation from members, we developed progress relatively well. The key factor in all of this was our means of communication, completed through formal meetings with our supervisor as well as informal meetings through online and offline means. As such, with our focus and targets set per meeting, we were able to steadily advance at a pace that allowed us to meet our main criteria as specified in Section 2.5.

With our roles set, Carmel acting as the leader, tasks were delegated amongst the four members in

our group; the team worked hand in hand in developing different aspects of the application, each of us delving into completely new territory such as smart contract development through Solidity, working with frameworks such as thirdweb or experimenting with IPFS gateways etc. With Anshana and Ken Siang working on smart contracts, Adyan experimenting with notification triggers and IPFS file retrieval, Carmel on the user interface, we accomplished such a great feat in a limited amount of time. Through the research we had completed, resources shared amongst ourselves, the team was able to integrate and create an application that satisfied our supervisor. Even though we had several hurdles, due to our inexperience in this specific field, we persevered and conducted our own experiments with new technologies to find what was suitable for our project.

7.4.1 Improvements

Initially, due to a lack of clarity on the project's scope, we faced challenges in organising the team into distinct front-end and back-end roles. This was due to the fact our back-end consisted of very intricate sections which we severely underestimated. We now recognise that segregating the group into specialists for each area would have simplified the development process significantly though, it would have been hard to know this in the early stages of development when we knew virtually nothing about blockchain.

Creating dWill had its fair share of ups and downs, with challenges that many of us were not prepared for as our mindset going into this was shaped by our current knowledge of Web2. Thus, being able to adapt to our new roles and mindset of retaining decentralization within our application, it proved that our team was cohesive and motivated. The components that make up our application are definitely convoluted though with the research we conducted ourselves and through utilizing our problem thinking and communication skills, we were able to highlight the most vital aspects of our application, resulting in a successful outcome.

8 Project Outcome

8.1 Expected Outcomes

By the end of the project, we aimed to fulfill all our requirements to ensure the basic functions were available for our demonstrations; this included features such as subscribing to dWill, uploading and retrieving files and most importantly, the countdown otherwise known as the dead man's switch. Alongside this, we wanted to add features such as:

1. **Key Exchange:** In early stages of development, we envisioned the method of allowing beneficiaries access by having the benefactor simply give them a decryption key. However, after much consideration, we decided against this as we wanted a more streamline process of retrieving assets. We decided we wanted to integrate the key exchange into dWill. Our main motivation of switching from automating this is for an added layer of security however, with the fear of private keys being visible to outsiders, this task proved to be a challenge.

2. **Hierarchy System:** In addition to this, we wanted to add a hierarchy system where beneficiaries would claim assets depending on the level assigned (i.e. if Beneficiary A was assigned 80% of the assets while Beneficiary B was assigned 20%, they would gain access to only that amount of assets). Since dWill is meant to essentially mimic traditional wills as they are done in real life, the added feature of having priority or a hierarchy with our beneficiaries would make the most sense.

3. **Gasless Transactions:** Alongside this, with every contract interaction that is made in our application, a gas amount is deducted from the user's wallet to pay for each interaction. We wanted to find a method to mitigate this additional cost as we had initially planned for all users to subscribe to our service and pay an amount yearly to gain access to the app's functionalities. With these gas transactions added, it meant that the user would be paying extra which is not ideal.

8.2 Achievements and Unmet Goals

Over the course of the year, we were able to achieve all the aforementioned requirements as we had initially expected, which was a success in and of itself. As for our bonus features that we wanted, their status differed:

1. **Key Exchange:** This integration was successful, as discussed in Section 5.2.
2. **Hierarchy System:** Due to time constraints, we were unable to integrate this and additionally, we found it difficult to partition the files. A question that we constantly asked was "What if a particular file requires another file but are split up during the partition?", thus without an answer to this, it hindered our progress in achieving the hierarchy system. However, our alternative to this was aside from assigning a global beneficiary, the benefactor has the ability to assign beneficiaries to specific files as well.
3. **Gas-less Transactions:** Our approach to accomplishing this was to create an OpenZeppelin relayer that can pay the gas cost. In this scheme, users signs messages (not transactions) containing information about a transaction they would like to execute. Due to the time constraints, we were not able to implement this feature.

9 Conclusion

Due to the nature of blockchain being very new to us, our team struggled in many areas to accomplish many of our requirements. This meant that we had to do additional research as well as completely shift our mindset to fully understand how the Web3 space worked. With this added knowledge, we were able to experiment with new technologies to implement our smart contracts which allowed us to connect to our authentication service (Metamask; through thirdweb) as well as connect to our storage platform (IPFS; through Pinata). While majority of these concepts are new, especially coding in Solidity as well as building a React application, we were able to achieve the essential features that we had agreed upon at the beginning of the semester. Thus, we deem our project a success.

10 Future Works

Our project at its core has great potential for the industry, as stated by many judges and peers. As part of feedback and the general notion of our application, we see many aspects that could further be improved to make our application more accessible.

1. **Non-restrictive audience:** One of the recurring questions we received was "why blockchain?", followed by "must users be a wallet holder?". Since majority of the user base currently is more familiar with Web2 methods of accessing data, the concept of creating a wallet and investing

in a new form of currency made several of our visitors turn their heads during our demonstration. Thus, if there were a way to implement our application while retaining decentralization, it would be more appealing to the public.
Additionally, since there is a learning curve in understanding how blockchain interactions work, being able to hide most of it from the users (i.e. removing our Metamask contract interaction popups), is definitely another improvement to user experience as well.

2. **Validation and automation:** Another improvement that we would definitely work on is the validation of wallet addresses. Currently, a user simply has to manually input someone's public wallet address as to assign them as the beneficiary (See Figure 15) or to log in as beneficiary, they must input the benefactor's address (See Figure 16). While for demonstration purposes this is fine, one of the key pieces of feedback received was how difficult this made the user experience as the application assumes that the user has the addresses at their disposal.
Realistically, it would be much more easier if there were suggested options rather than having the users type in addresses themselves; it not only automates the processes but it also streamlines the whole flow of having to look up one's account and paste it in. Additionally, our current process of inputting addresses have no form of validation, meaning the input field will simply accept anything. With blockchain explorers such as Polyscan, we are able to see a myriad of addresses that currently exist. This means implementing the validation based on existing accounts is a definite possible task to work upon.
Another concern that was brought up was, "what if the beneficiaries were children?". There can be potential use cases where children could potentially be beneficiaries, and in that cases, they need to be able to easily use the application. Hence, implementing validation based on existing accounts could be a possible solution to this.

11 Appendix

11.1 User Manual

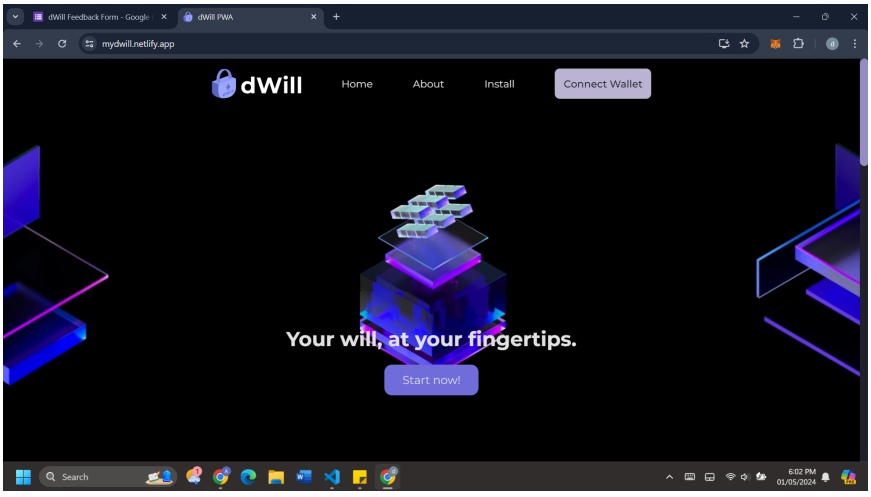

Figure 10: This figure shows dWill's landing page

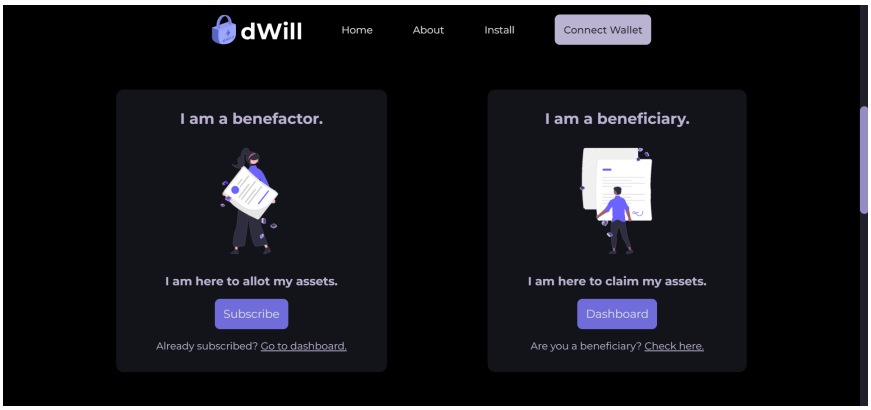

Figure 11: This figure shows the page after clicking the *Start Now* button on the landing page

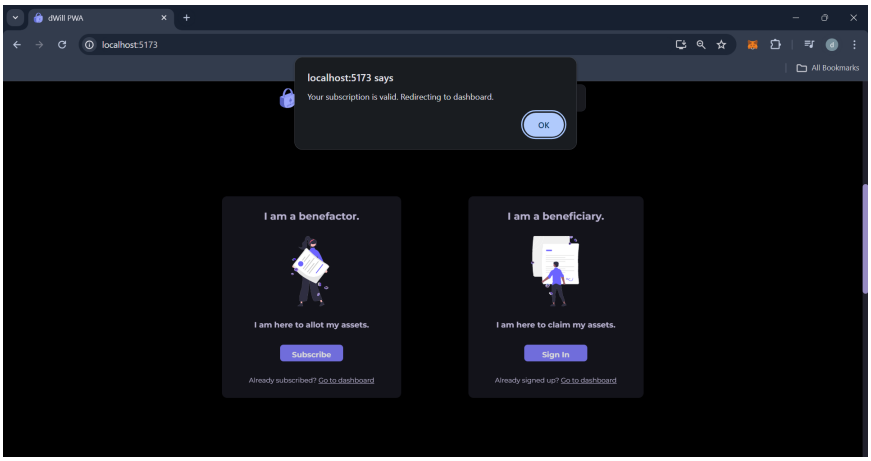

Figure 12: This figure shows the page after clicking the *Subscribe* button and paying the subscription and gas fees.

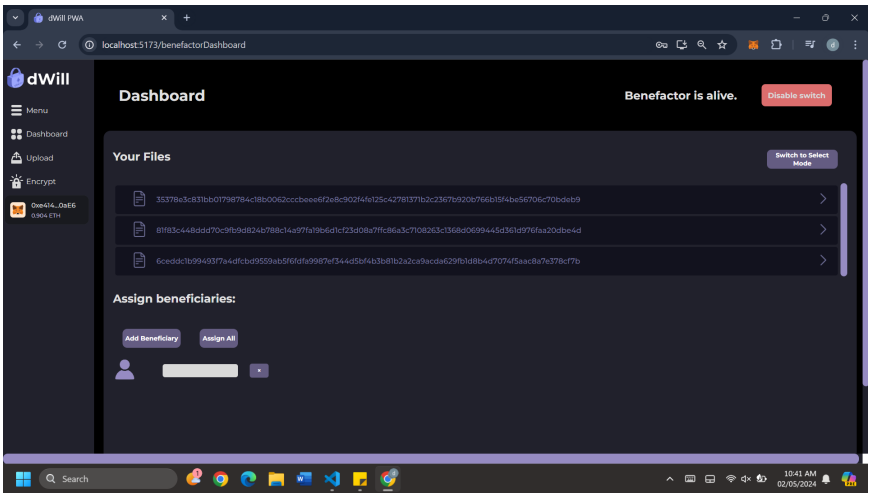

Figure 13: This figure shows the benefactor's dashboard where they can assign beneficiaries, generate their public key and disable their switch.

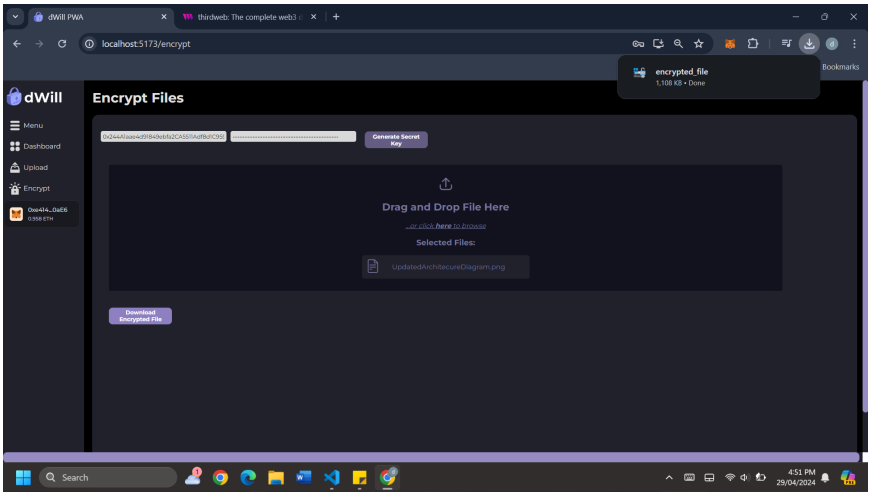

Figure 14: This figure shows the encrypt page where benefactors can encrypt and download encrypted files.

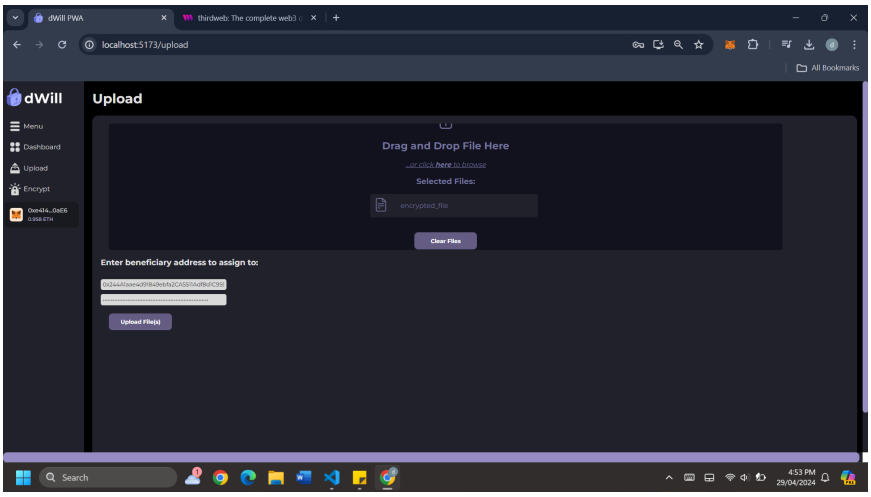

Figure 15: This figure shows the upload page where benefactors can upload and assign files to beneficiaries.

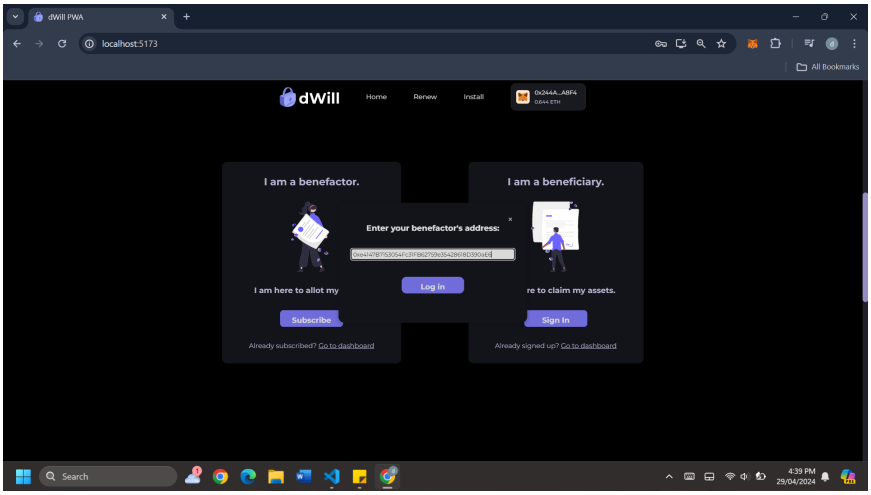

Figure 16: This figure shows how a beneficiary must login by entering their benefactor's address

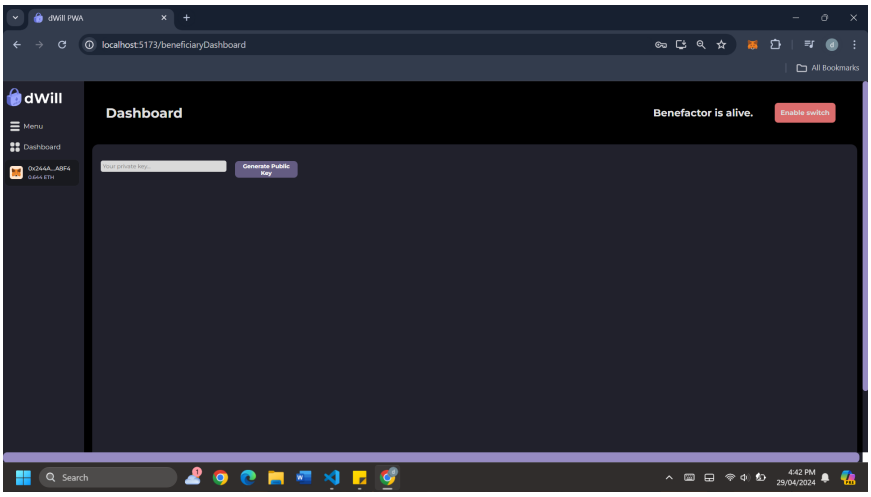

Figure 17: This figure shows how the beneficiary dashboard where they can generate their public key, enable their benefactor's switch and where their files will appear for download upon their benefactor's incapacitance.

11.2 Test Suites

11.2.1 Unit Testing Suites

This section includes the unit testing suites for the 3 smart contracts used in our application.

Table 1: This suite describes all the test cases involved in testing dead man's switch smart contract.

Module	**Dead Man's Switch Smart Contract**				
Tested by	Lai Ken Siang and Anshana Manoharan				
Test Date	27/04/2024				
Pre-conditions	Import the contract to a project in the Online Remix Solidity IDE				
ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
1	Compile and deploy smart contract	1. Compile the contract 2. Deploy the contract	Smart contract successfully deployed	Smart contract successfully deployed	Pass
2	Initialise benefactor's dead man's switch using *setBenefactor* function	1. Connect with an address 2. Click *setBenefactor* button	User is confirmed as benefactor	User is confirmed as benefactor	Pass
3	Assign beneficiaries using *addBeneficiaries* function with valid benefactor address	1. Connect with valid benefactor address 2. Enter beneficiaries addresses 3. Click the *transact* button	Assignment successful	Assignment successful	Pass
4	Assign beneficiaries using *addBeneficiaries* function using invalid benefactor address	1. Connect with invalid benefactor address 2. Enter beneficiaries addresses 3. Click the *transact* button	Benefactor does not exist	Benefactor does not exist	Pass
5	Verify an existing beneficiary using *isBeneficiary* function	1. Enter valid benefactor and existing beneficiary address 2. Click the *call* button	True	True	Pass

Table 2: This table is a continuation of the previous suite.

ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
6	Verify a non-existing beneficiary using *isBeneficiary* function	1. Enter valid benefactor and a non-existing address 2. Click the *call* button	False	False	Pass
7	Verify any beneficiary using *isBeneficiary* function with an invalid benefactor	1. Enter an invalid benefactor and any beneficiary address 2. Click the *call* button	Benefactor does not exist	Benefactor does not exist	Pass
8	Add benefactor public key using *addBenefactorPublicKey* function to a valid benefactor	1. Enter valid benefactor address 2. Enter public key 3. Click the *transact* button	Public key added successfully	Public key added successfully	Pass
9	Add benefactor public key using *addBenefactorPublicKey* function to an invalid benefactor	1. Enter invalid benefactor address 2. Enter public key 3. Click the *transact* button	Benefactor does not exist	Benefactor does not exist	Pass
10	Add beneficiary public key using *addBeneficiaryPublicKey* function with valid benefactor and valid beneficiary	1. Enter valid beneficiary and valid benefactor address 2. Enter public key 3. Click the *transact* button	Public key added successfully	Public key added successfully	Pass

Table 3: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
11	Add beneficiary public key using *addBeneficiaryPublicKey* function with invalid benefactor	1. Enter any address for beneficiary and invalid benefactor address 2. Enter public key 3. Click the *transact* button	Beneficiary/Benefactor not found	Beneficiary/Benefactor not found	Pass
12	Add beneficiary public key using *addBeneficiaryPublicKey* function with valid benefactor and non-existing beneficiary	1. Enter non-existing beneficiary and valid benefactor address 2. Enter public key 3. Click the *transact* button	Beneficiary/Benefactor not found	Beneficiary/Benefactor not found	Pass
13	Get benefactor public key using *getBenefactorPublicKey* function with valid benefactor and assigned beneficiary address	1. Connect with valid beneficiary address 2. Enter valid benefactor address 3. Click the *call* button	Function returns benefactor public key	Function returns benefactor public key	Pass
14	Get benefactor public key using *getBenefactorPublicKey* function with invalid benefactor address and any beneficiary address	1. Connect with any address 2. Enter invalid benefactor address 3. Click the *call* button	Function returns ""	Function returns benefactor ""	Pass
15	Get benefactor public key using *getBenefactorPublicKey* function with valid benefactor address and invalid beneficiary address	1. Connect with invalid beneficiary address 2. Enter valid benefactor address 3. Click the *call* button	Function returns ""	Function returns benefactor ""	Pass

Table 4: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
16	Get beneficiary public key using *getBeneficiaryPublicKey* function with valid benefactor and beneficiary address	1. Enter valid beneficiary and valid benefactor address 2. Click the *call* button	Function returns beneficiary public key	Function returns beneficiary public key	Pass
17	Get beneficiary public key using *getBeneficiaryPublicKey* function with invalid benefactor address and any beneficiary address	1. Enter invalid benefactor address and any beneficiary address 2. Click the *call* button	Function returns ""	Function returns benefactor ""	Pass
18	Get beneficiary public key using *getBeneficiaryPublicKey* function with valid benefactor address and invalid beneficiary address	1. Enter valid benefactor address and invalid beneficiary address 2. Click the *call* button	Function returns ""	Function returns benefactor ""	Pass
19	Assign an unassigned IPFS CID to a valid beneficiary using *addIpfsCIDs* function	1. Connect with valid benefactor account 2. Enter valid beneficiary address and unassigned CID 3. Click the *transact* button	IPFS CIDs assigned successfully	IPFS CIDs assigned successfully	Pass
20	Assign an already assigned IPFS CID to a valid beneficiary using *addIpfsCIDs* function	1. Connect with valid benefactor account 2. Enter valid beneficiary address and an already assigned CID 3. Click the *transact* button	CID already assigned to this beneficiary	CID already assigned to this beneficiary	Pass

Table 5: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
21	Assign an unassigned IPFS CID to an invalid beneficiary using *addIpfsCIDs* function	1. Connect with valid benefactor account 2. Enter beneficiary address and unassigned CID 3. Click the *transact* button	Beneficiary not found	Beneficiary not found	Pass
22	Assign an unassigned IPFS CID to any beneficiary as an invalid benefactor using *addIpfsCIDs* function	1. Connect with invalid benefactor address 2. Enter any address as beneficiary and unassigned CID 3. Click the *transact* button	Beneficiary not found	Beneficiary not found	Pass
23	Remove an assigned IPFS CID from an existing beneficiary using *removesIpfsCIDs* function	1. Connect to valid benefactor account 2. Enter an existing beneficiary address and an assigned IPFS CID 3. Click the *transact* button	IPFS CID removed successfully	IPFS CID removed successfully	Pass
24	Remove any IPFS CID from a non-existing beneficiary using *removesIpfsCIDs* function	1. Connect to valid benefactor account 2. Enter non-existing beneficiary address and any IPFS CID 3. Click the *transact* button	Beneficiary does not exist	Beneficiary does not exist	Pass
25	Remove any IPFS CID from any beneficiary as invalid benefactor using *removesIpfsCIDs* function	1. Connect to invalid benefactor account 2. Enter any beneficiary address and any IPFS CID 3. Click the *transact* button	Beneficiary does not exist	Beneficiary does not exist	Pass

Table 6: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
26	Remove an unassigned IPFS CID from an existing beneficiary using *removesIpfsCIDs* function	1. Connect to valid benefactor account 2. Enter existing beneficiary address and an unassigned IPFS CID 3. Click the *transact* button	CID not assigned to this beneficiary	CID not assigned to this beneficiary	Pass
27	Remove assigned beneficiary using *removeBeneficiary* function as valid benefactor	1. Connect to valid benefactor account 2. Enter assigned beneficiary address 3. Click the *transact* button	Beneficiary removed successfully	Beneficiary does not exist	Pass
28	Remove un-assigned beneficiary using *removeBeneficiary* function as valid benefactor	1. Connect to valid benefactor account 2. Enter un-assigned beneficiary address 3. Click the *transact* button	Beneficiary not found	Beneficiary not found	Pass
29	Remove any beneficiary address using *removeBeneficiary* function as invalid benefactor	1. Connect to invalid benefactor account 2. Enter any beneficiary address 3. Click the *transact* button	Beneficiary not found	Beneficiary not found	Pass
30	Remove valid benefactor using *removeBenefactor* function	1. Connect to valid benefactor account 2. Enter a valid benefactor address 3. Click the *transact* button	Benefactor removed successfully	Benefactor removed successfully	Pass

Table 7: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
31	Remove an invalid benefactor using *removeBenefactor* function	1. Connect to an invalid benefactor account 2. Enter an invalid benefactor address 3. Click the *transact* button	Benefactor not registered	Benefactor not registered	Pass
32	Enable the dead mans switch using *enableSwitch* function with a valid benefactor and beneficiary address	1. Connect to a valid beneficiary account 2. Enter an valid benefactor address 3. Click the *transact* button	Dead mans switch enabled	Dead mans switch enabled	Pass
33	Enable the dead mans switch using *enableSwitch* function with an invalid benefactor and any beneficiary address	1. Connect to any beneficiary account 2. Enter an invalid benefactor address 3. Click the *transact* button	Not a valid beneficiary	Not a valid beneficiary	Pass
34	Enable the dead mans switch using *enableSwitch* function with a valid benefactor and an invalid beneficiary address	1. Connect to an invalid beneficiary account 2. Enter a valid benefactor address 3. Click the *transact* button	Not a valid beneficiary	Not a valid beneficiary	Pass
35	Respond to the switch with *respondToSwitch* function as a valid benefactor when the switch is on and	1. Connect to a valid benefactor address 2. Click the *respondToSwitch* button when the switch is on and the countdown is not over	Benefactor is alive	Benefactor is alive	Pass
36	Respond to the switch with *respondToSwitch* function as an invalid benefactor	1. Connect to an invalid benefactor address 2. Click the *respondToSwitch* button when the switch is on	Benefactor does not exist	Benefactor does not exist	Pass

Table 8: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
36	Respond to the switch with *respondToSwitch* function as an invalid benefactor	1. Connect to an invalid benefactor address 2. Click the *respondToSwitch* button when the switch is on	Benefactor does not exist	Benefactor does not exist	Pass
37	Respond to the switch with *respondToSwitch* function as a valid benefactor when the countdown is over	1. Connect to a valid benefactor address 2. Click the *respondToSwitch* button when the switch is on	Benefactor is dead	Benefactor is dead	Pass
38	Get switch status with valid benefactor using *getSwitchStatus* function when the switch is enabled	1. Enter a valid benefactor address 2. Click the *transact* button when the switch is on	True	True	Pass
39	Get switch status with valid benefactor using *getSwitchStatus* function when the switch is disabled	1. Enter a valid benefactor address 2. Click the *transact* button when the switch is off	False	False	Pass
41	Check remaining countdown time with *getRemainingCountdownTime* function as a valid benefactor when the switch is enabled	1. Enter a valid benefactor address 2. Click the *transact* button when switch status is true	Returns the remaining countdown time in seconds	Returns the remaining countdown time in seconds	Pass

Table 9: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
41	Check remaining countdown time with *getRemainingCountdownTime* function as a valid benefactor when the switch is disabled	1. Enter a valid benefactor address 2. Click the *transact* button when switch status is false.	Returns 0	Returns 0	Pass
42	Get alive status with valid benefactor using *checkAliveStatusFunction* function when the switch is not triggered	1. Enter an valid benefactor address 2. Click the *transact* button	True	True	Pass
43	Get alive status with valid benefactor using *checkAliveStatusFunction* function when the switch was not disabled and the countdown is over	1. Enter an valid benefactor address 2. Click the *transact* button	False	False	Pass
44	Get CIDs as valid beneficiaries using *getCIDs* function when the benefactor isAlive is true	1. Enter a valid benefactor and beneficiary address 2. Click the *call* button	No access to CIDs	No access to CIDs	Pass
45	Get CIDs as valid beneficiaries using *getCIDs* function when the benefactor isAlive is false	1. Enter a valid benefactor and beneficiary address 2. Click the *call* button	Returns the array of encrypted hashes	Returns the array of encrypted hashes	Pass

Table 10: This table is a continuation of the previous suite.

ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
46	Get beneficiary details using *getBeneficiaryData* function as a valid beneficiary	1. Enter a valid beneficiary and benefactor address 2. Click the *transact* button	Returns beneficiary details	Returns beneficiary details	Pass
47	Get benefactor details using *getBenefactorData* function as a valid benefactor	1. Enter a valid benefactor address 2. Click the *transact* button	Returns benefactor details	Returns benefactor details	Pass

Table 11: This suite describes all the test cases involved in testing subscription smart contract.

Module	**Subscription Smart Contract**				
Tested by	Lai Ken Siang				
Test Date	28/04/2024				
Pre-conditions	Import the contract to a project in the Online Remix Solidity IDE				
ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
1	Compile and deploy smart contract	1. Compile the contract 2. Deploy the contract	Smart contract successfully deployed	Smart contract successfully deployed	Pass
2	Get subscription status using *checkSubscriptonStatus* function	1. Enter a user's address 2. Click *checkSubscriptonStatus* button	Returns subscription status	User is confirmed as benefactor	Pass
3	Subscribe a user using *subscribe* function	1. Enter a user's address 2. Click the *subscribe* button	Returns subscription status	Returns subscription status	Pass
4	Unsubscribe a subscribed user using *unsubscribe* function	1. Enter a user's address 2. Click the *unsubscribe* button	Returns subscription status	Returns subscription status	Pass
4	Unsubscribe a unsubscribed user using *unsubscribe* function	1. Enter a user's address 2. Click the *unsubscribe* button	Returns "Not subscribed"	Returns "Not subscribed"	Pass

Table 12: This suite describes all the test cases involved in testing upload smart contract.

Module	**Upload Smart Contract**				
Tested by	Lai Ken Siang				
Test Date	28/04/2024				
Pre-conditions	Import the contract to a project in the Online Remix Solidity IDE				
ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
1	Compile and deploy smart contract	1. Compile the contract 2. Deploy the contract	Smart contract successfully deployed	Smart contract successfully deployed	Pass
2	Add URL to users data using *add* function	1. Enter a user's address 2. Enter URL 3. Click *transact* button	Data uploaded successfully	Data uploaded successfully	Pass
3	Allow user access to data using *allow* function	1. Enter a user's address 2. Click the *transact* button	Access granted and update access list	Access granted and update access list	Pass
4	Disallow a user access using *disallow* function	1. Enter a user's address 2. Click the *transact* button	Revoke access and update access list	Revoke access and update access list	Pass
5	Get URLs that can be accessed by specified user using *shareAcess* function	1. Enter a user's address 2. Click *shareAccess* button	Function return access list and user's access status	Function return access list and user's access status	Pass
6	Return URL of data mapped with user using *display* function	1. Enter a user's address 2. Click *transact* button	Function return URL of data mapped with user	Function return URL of data mapped with user	Pass

11.2.2 System Testing Suite

This section includes the system testing suite that tested whether the requirements were fulfilled.

Table 13: This suite describes all the test cases involved in testing the developed system.

Module	**dWill Application**				
Tested by	Adyan Dean bin Wafdi Kamil and Anshana Manoharan				
Test Date	28/04/2024				
Pre-conditions	Have a Metamask wallet connected to the Polygon Amoy testnet and a minimum of 2 MATIC.				
ID	**Test Case**	**Steps**	**Expected Result**	**Actual Result**	**Pass/ Fail**
1	Verify Metamask authentication by subscribing and logging in as benefactor	1. Connect wallet in dWill homepage 2. Click *"Start now"* 3. Click *"Subscribe"* 4. Pay the gas fees for the function calls in the subscription and dead mans switch contracts from the Metamask pop-up 5. Click alert's ok button	Redirect to benefactor dashboard	Redirect to benefactor dashboard	Pass
2	Generate benefactor's public key	1. Be on the benefactor dashboard page 2. Enter a private key 3. Click *"Generate public key"* button 4. Pay gas fee from Metamask pop-up	Button disappears and alert appears that public key is generated	Button disappears and alert appears that public key is generated	Pass
3	Assign beneficiaries	1. Be on the benefactor dashboard page 2. Enter beneficiary address 3. Click *"Add Beneficiary"* 4. Enter another beneficiary address 5. Click *"Assign All"* 6. Pay gas fee from Metamask pop-up	Alert says successfully added beneficiaries	Alert says successfully added beneficiaries	Pass

Table 14: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
4	Log in as beneficiary	1. Connect wallet in dWill homepage 2. Click *"Start now"* 3. Click *"Sign in"* 4. Enter benefactor address and click *"Log in"* 5. Pay the gas fees for the function calls in the dead mans switch contract from the Metamask pop-up	Redirect to beneficiary dashboard	Redirect to beneficiary dashboard	Pass
5	Generate beneficiary's public key	1. Be on the beneficiary dashboard page 2. Enter a private key 3. Click *"Generate public key"* button 4. Pay gas fee from Metamask pop-up	Button disappears and alert appears that public key is generated	Button disappears and alert appears that public key is generated	Pass
6	Encrypt a file	1. Be on the benefactor encrypt page 2. Enter beneficiary address and your private key 3. Click *"Generate secret key"* 4. Upload a file for encryption 5. Click *"Download encrypted file"*	Encrypted file is downloaded automatically	Encrypted file is downloaded automatically	Pass

Table 15: This table is a continuation of the previous suite.

ID	Test Case	Steps	Expected Result	Actual Result	Pass/ Fail
7	Upload and assign a file	1. Be on the benefactor upload page 2. Enter a private key and beneficiary to assign to 3. Upload an encrypted file 4. Click *"Upload File(s)"* button 5. Pay gas fee from Metamask pop-up	Alert appears that file was successfully pinned	Alert appears that file was successfully pinned	Pass
8	Enable the benefactor's dead mans switch	1. Be on the beneficiary dashboard page 2. Click *"Enable switch"* 3. Pay gas fee from Metamask pop-up	Alert appears that switch is enabled successfully	Alert appears that switch is enabled successfully	Pass
9	Disable the benefactor's dead mans switch	1. Be on the benefactor dashboard page 2. Click *"Disable switch"* 3. Pay gas fee from Metamask pop-up	Alert appears that switch is disabled successfully	Alert appears that switch is disabled successfully	Pass

11.3 Meeting Minutes

Table 16: Meeting minutes for the first formal meeting.

Meeting type	Formal		
Meeting date	3rd October 2023	**Meeting duration**	1 hour
Minutes prepared by			Anshana Manoharan
Meeting objective			
Brainstorm project idea and get concrete understanding of the objectives and aims of the project.			
Attendees			
Dr Nabil, Carmel Natasha Barnabas, Adyan Dean bin Wafdi, Anshana Manoharan, Lai Ken Siang			
Tasks done prior the meeting			
None since this is the first meeting.			
Topic	**Discussion**		
Choosing a Project idea	Dr Nabil briefed the group on different potential project ideas; application to track and add sCPD points, LinkedIn verifiability feature where professional and educational institutions verify a user's achievements and a feature which automatically generates a resume, digital dead man's switch implemented on the blockchain. After thorough discussion, the group chose to work on the dead man's switch using blockchain technology.		
Additional Notes	1. Data can be stored on an existing web3 file storage system like IPFS. 2. Benefactor uploads data to the app. 3. The app encrypts and stores the data into IPFS. 4. The ideology where the beneficiary has the key to the benefactor's "house" but does not know its "location". 5. Benefactor physically hands over the decryption key to the beneficiary. 6. If beneficiary suspects the benefactor is dead, the beneficiary must trigger the countdown. 7. If the benefactor is alive, the benefactor must switch off the countdown, else assumed dead. 8. Beneficiary is only directed to location of data after the benefactor is confirmed to be dead.		
Post-meeting objective			
Research on blockchain technology, web3 development. Figure out how to implement the smart contracts and familiarize with programming languages to be used. Understand the main infrastructure of the app to be built.			

Table 17: Meeting minutes for the second formal meeting

Meeting type	Formal		
Meeting date	11th October 2023	**Meeting duration**	1 hour
Minutes prepared by			Anshana Manoharan
Meeting objective			
Clear doubts from previous informal meeting.			
Attendees			
Dr Nabil, Carmel Natasha Barnabas, Adyan Dean bin Wafdi, Anshana Manoharan, Lai Ken Siang			
Tasks done prior the meeting			
Searched up useful resources to be used for the implementation of the application.			
Topic	**Discussion**		
Authentication	Instead of using biometrics, Dr Nabil suggested to use MetaMask wallets.		
Account merging	Accounts will not be merged, instead benefactor account should be deleted after the grace period is completed.		
Blacklisting	No blacklisting, as priority is only given to the current benefactor's wishes.		
Additional Notes	1. Either use 6 months or 12 months subscription. 2. IPFS allows the location access to beneficiary after benefactor dies. This is to be done by the smart contract. 3. Better to use NodeJS and JavaScript for back-end. 4. Beneficiary will decide when to trigger countdown, there will be no reminders for this. 5. After benefactor has passed, there will only be a transfer of ownership, no need to transfer any data files.		
Post-meeting objective			
Start drafting the project description and start technical research.			

Table 18: Meeting minutes for the third formal meeting

Meeting type	Formal		
Meeting date	22nd November 2023	**Meeting duration**	1 hour
Minutes prepared by			Anshana Manoharan
Meeting objective			
Discuss current progress of the project.			
Attendees			
Dr Nabil, Carmel Natasha Barnabas, Adyan Dean bin Wafdi, Anshana Manoharan, Lai Ken Siang			
Tasks done prior the meeting			
Began writing smart contracts on Solidity, and experimented with mobile app development using Ionic and React Native. Experimenting of different methods to connect MetaMask with mobile app, web3.storage connection, and payment method for subscription.			
Topic	**Discussion**		
Handover of decryption key	Required to be confirmed by members and Dr Nabil: both parties need to confirm the handover process, only then, the dead man's switch will be enabled. Concluded that since the beneficiary is a trustee, handover of the key will happen securely. This may be used as a nice-to-have feature.		
Confirm updated use case diagram	Remove beneficiary subscribe use case. Only benefactor needs to subscribe because they are the one setting the dead man's switch.		
MetaMask redirection issue	Dr Nabil suggested to connect Infura with an API Use the steps in Quicknode article "Connection with MetaMask wallet", and dev.to article "Connecting MetaMask with Flutter".		
Additional Notes	1. There is no requirement to make a mobile app but if the mobile app can provide notifications real time it can be advantageous. 2. Will focus on making the app just to turn off the countdown and for notifications. 3. Research about openzepplin - how to make erc20 token? Can use it as the currency for transactions (buying storage, etc)		
Post-meeting objective			
Update the use case diagram, create the activity diagram and flowchart. Work on how to make the MetaMask connect to the app when redirecting back to the call. Experiment with third web and deploy a test smart contract.			

Table 19: Meeting minutes for the fourth formal meeting

Meeting type	Formal		
Meeting date	06th February 2024	**Meeting duration**	1 hour
Minutes prepared by			Anshana Manoharan
Meeting objective			
Discuss current progress of the project.			
Attendees			
Dr Nabil, Carmel Natasha Barnabas, Adyan Dean bin Wafdi, Anshana Manoharan, Lai Ken Siang			
Tasks done prior the meeting			
Experimented with Flutter, React Native with MetaMask, completed activity and use case diagrams for documentation. Programmed website for logging in with MetaMask and redirecting to the user's dashboard.			
Topic	**Discussion**		
Diffie Hellman Key Exchange Algorithm	Dr Nabil permitted group to use this method since the decryption key can be safely generated rather than transmitted.		
Additional Notes	None		
Post-meeting objective			
Focus on integrating all separate components. Check thirdweb documentation because since the group is already using thirdweb, the deploy tool can be used instead of using hardhat.			

Table 20: Meeting minutes for the fifth formal meeting

Meeting type	Formal		
Meeting date	21st March 2024	**Meeting duration**	1 hour
Minutes prepared by			Anshana Manoharan
Meeting objective			
Update progress of assigned tasks. Discuss poster layout.			
Attendees			
Dr Nabil, Carmel Natasha Barnabas, Adyan Dean bin Wafdi, Anshana Manoharan, Lai Ken Siang			
Tasks done prior the meeting			
File encryption and download, dead mans switch functionality, downloading of files from gateway, live countdown.			
Topic	**Discussion**		
Feedback on updates	Gateway link was publicly accessible on etherscan, need to find a way to make it hidden. Dr Nabil proposed to transition to hardhat to deploy the smart contracts instead of using Remix IDE for deployment.		
Poster	Points that can be included: decentralisation and distributed storage, everything is automated, users having complete control over their data, future directions for the project, system evaluation. How can it benefit the industry?		
Feedback of Demo	Dr Nabil noted that the group was progressing at a good pace.		
Additional Notes	For encryption of the IPFS hash, there is a way to sign (check the video Dr Nabil sent) Research if Diffie Hellman Key Exchange can be used to encrypt and decrypt CIDs.		
Post-meeting objective			
Add Diffie Hellman to the smart contract. Fix the upload issue. Finish downloading of files functionality. Add files assignment to beneficiaries. Integrate all components.			

References

Ivanov, D. and Pashkov, P. (2023), Trust architecture in apparel supply chains migrating to web3 protocols, *in* '2023 Dynamics of Systems, Mechanisms and Machines (Dynamics)', IEEE, pp. 1–4.

Macii, E. and Poncino, M. (2014), 'Design of dead man's switch: A review', *Journal of Assistive Technology* **8**(2), 85–92.

Merre, R. (2020), 'Safety beyond the grave with chainlink and NGRAVE', *ResearchGate* **8**.

Pereira, F. H. S., Prates, R. O., Maciel, C. and Pereira, V. C. (2017), 'Combining configurable interaction anticipation challenges and volitional aspects in the analysis of digital posthumous communication systems', *Journal on Interactive Systems* **8**(2), 1.