College of Engineering Pune
(An Autonomous Institute of the Govt. of Maharashtra)

Software Engineering Mini Project II

Third Year Computer Engineering

ShardShare – Consensus-based secret sharing

Under the guidance of

Prof. Rohini Y. Sarode

Hrishikesh Athalye 111803154

Nachiket Pethe 141903012

Table of Contents

1. Abstract

The management of credentials to classified information is a very crucial problem. Secret information like passwords of root accounts, bank account numbers and other important secrets often resides with only one person and this centralisation of secrets makes it very difficult to recover these secrets in the event the person holding the credentials is absent. As the amount of credentials an individual needs to manage is growing, a solution to safely share these credentials is needed.

There are various problems with currently available solutions. It is common practice for only one person to know a secret credential, in order to keep it safe. But this leads to high risk as information is concentrated with a single person and prone to loss. This is a *single point-of-failure.*

Many secrets are such that they should only be accessed by a group of people and not a single person in the absence of the owner. For e.g. - a person's will is to be opened only if all members party to the will are present. Therefore, any scheme for sharing credentials should also be consensus based i.e.; once the secret is shared with a group of people, the secret should not be accessible by any one person if all others don't consent to it.

Some existing solutions to the problem of credential management are like these - One solution people use is to store their secret keys on their computer or in a journal. But this approach is not fault proof in the event that the computer breaks down or if the journal is lost. Some solutions propose storing such information on the cloud, which is again not a very secure approach given the amount of data leaks that happen every now and then. Giving a copy of the key to some people is also a solution but it does not protect against unauthorized access since the person comes to know the entire key. Hence, decentralising the key is needed.

Approaches that come close to decentralising the key do it by splitting the key and distributing one part to each person in the group. This looks decentralised but actually is as fault prone as centralised approaches since, if even one person is unavailable from the group the entire key is lost since all people are needed for recovery. Hence, an algorithmic solution to credential sharing that can address all these problems is needed.

2. Introduction

Our software – ShardShare, is an attempt to address and solve all of the problems mentioned above. The algorithmic solution that we propose to use i.e., Shamir's Secret Sharing Scheme, is perfect for our software since it is free of all of the above problems. The algorithm is already in use in specialised software particularly those pertaining to managing cryptocurrency wallet passwords. We plan to develop a web app and a mobile device interface that will make this algorithm available on demand to anyone who wants to share any kind of textual secret among a group of people in order to decentralise it and make it easy to recover in the event of any unforeseen incident.

We see our application being put to use in any situation where a password needs to be remembered and where using "Forgot Password" may not be an option, as is the case with cryptocurrency wallet passwords. It can also be put into use to enforce consensus among people who have access to a secret. An example of this would be the password of a will that needs to be opened only if a certain number of people are consenting to it being opened. In situations like these our application can ensure that unless k out of n people agree to recover a secret, the secret can't be recovered. This prevents, at least to a certain extent, against unauthorized access.

3. SRS

3.1 External Interface Requirements

Name	Date	Reason For Changes	Version
SRS	28/02/22		1.0

3.2 Introduction

3.2.1 Purpose

This project's primary purpose is to solve various problems that arise when sharing sensitive secrets like passcodes, identification numbers, and credentials to sensitive information. It aims to enable users to manage sensitive secret information by being able to – securely lock, distribute parts of locked secret information, and recombine and unlock it based on the consensus of a defined number of people. In essence, it aims to implement a tamper-proof and consensus-based mechanism to share secret information.

This document presents a detailed description of the project. It will explain the software's purpose and features, the software's interfaces, what the software will do, and the constraints under which it must operate. This document is intended for users of the software and also potential developers.

3.2.2 Document Conventions

This document was created based on the IEEE template for System Requirement Specification Documents. The section and subsection headings are in bold text, followed by the relevant information. All hyperlinks are in underlined blue text.

3.2.3 Intended Audience and Reading Suggestions

This document should be helpful to users to get a brief overview of the features the project provides and developers/testers who would wish to contribute to the project in any way.

The project's intended audience is any individual who wishes to share his/her secret credentials like bank account numbers, passwords of accounts, etc with some trusted people or organisations who wish to decentralise access to a password-protected resource such as a root account.

3.2.4 Product Scope

The scope of this project spans any area where consensus-based sharing of secrets is important. It is useful to individuals/organisations who wish to share secret credentials with a number of trusted people to ensure that the secret remains recoverable even if the owner isn't present. This applies to many scenarios, including legal documents like a will to passwords of root accounts in organizations which should remain accessible in emergency situations even if the admin is not present. Being decentralised, this way to share secret information comes at a safety advantage compared to most other schemes.

3.3 Overall Description

3.3.1 Product Perspective

This is a new and self-contained product aiming to provide start-to-end functionality with respect to credential and secret sharing. It can also act as a supplementary product to currently available authentication systems by adding the functionality of credential sharing to them and making them fault-proof.

The project aims to solve the following existing problems:

- ***Centralisation of secrets*** – It is common practice for only one person to know a secret credential because it is seen as the only secure way to guarantee that unauthorized access will not occur. But the centralization of secrets in this way leads to high risk as information is concentrated with a single person and prone to lose.
- ***Single Point-of-Failure*** – Due to the centralised nature of the way we currently hold secrets, i.e., with one person, unfortunate events like if the person holding the credentials dies or is unavailable for a long period of time, access to crucial information is hampered. This is a single-point-of-failure. Therefore, more often than not, credentials need to be shared with people.
- ***No Consensus*** – Even if credentials are shared, many credentials are such that they should only be accessed by a group of people and not a single person in the absence of the owner. For example, a person's will is to be opened only if all members party to the will are present. And a similar case can be presented for classified business documents that are only supposed to be opened in the presence of all or some critical board members but not by any one person alone. Many secret sharing schemes are not consensus-based i.e.; once the secret is shared with a group of people, the secret can be accessed by any person even if all others don't consent to it.

3.3.2 Product Functions

The major functionality that the product will provide include:

1. ***User authentication*** **-** User account creation and login
2. ***Accepting secret keys*** **-** Accepting a secret credential from the sharer
3. ***Sharding and sharing*** **-** Splitting (Sharding) the credential into parts according to Shamir's secret sharing algorithm and sharing credential shards with a list of trusted people.
4. ***Consensus-based key recovery*** **-** A user with whom a key shard is shared will be able to request for key recovery and will be able to recover the key only if a fixed-size subset of people out of those provided by the key sharer approve of it.
5. ***Resharing and Tracking*** **-** The owner of the key will be able to revoke an existing key and re-share different key parts. He will also be able to track all keys shared by him till now.

3.3.3 User Classes and Characteristics

End Users:

The application is developed for a general group of users, giving them equal rights and a similar set of functions. People who hold sensitive information and secrets will find our application’s feature list most useful and valuable, though the application will be accessible to any person willing to decentralize a secret.

The application will also make the process of holding secrets easier for people with whom a secret has been shared since they will easily be able to manage various key parts shared with them.

3.3.4 Operating Environment

User Operating Environment:

- Any browser supporting **HTTP/HTTPS**

Developer Operating Environment:

- **Node.js, MongoDB, React.js**

Operating System:

- Any operating system like Ubuntu/Windows capable of running Node.js

3.3.5 Design and Implementation Constraints

Hardware Constraints **-** The algorithm that this project uses namely, Shamir’s secret sharing scheme is a mathematically intensive algorithm that may require significant computing power if very large inputs are given. But since this is rarely the case with passwords or other normal text-based secrets, this doesn't affect the application significantly.

Software Constraints - Since the application is built using JavaScript and JavaScript based libraries which include React.js and Express, any platform on which it has to be tested by developers must have a JavaScript runtime environment like Node.js.

3.3.6 User Documentation

The application will be provided with a README file containing detailed explanations and steps to install and run. It will also specify dependencies and versions of the technologies used. The documentation will be made available to users from the landing page of the application.

3.3.7 Assumptions and Dependencies

Assumptions:

- The person decentralising the secret is honest and will not compromise the secret in any situation.
- The participants of the secret are rational, credible to handle sensitive information responsibly.
- The default way of sending the shard, i.e., Email, is a safe way to share the shard.
- There is no third party intercepting the data that goes out of the application.

3.4 External Interface Requirements

3.4.1 User Interfaces

The main components of the user interface can be listed as follows:

- ***Login Screen*** - This will be the first screen the user is greeted with as soon as the web app is accessed. This is where existing users can log in to their account or can be directed to another screen to create a new account.
- ***User Dashboard*** - This will be the next thing the user sees after logging in. It will act as a welcome screen as well as contain the various options the application provides.
- ***Navbar*** - A top navbar or sidebar will guide the user through the various options that the application provides and will navigate the user to one of the following views - Share Key view, Shared With You view, Shared By You view, Recovery Requests view.
- ***Other Miscellaneous UI Components*** - We may need to include some components that are not stated here if deemed critical to the functioning of the application.

3.4.2 Hardware Interfaces

- **HI-1:** The main hardware component that our application interfaces with is the **database**. The application uses a hosted MongoDB Atlas database to ensure remote accessibility.
- **HI-2: Processor** 1GHz or faster
- **HI-3: RAM** 1GB (32bit) or 2GB (64bit)

3.4.3 Software Interfaces

- ***Runtimes***: The Node.js runtime environment is central to the functioning of our application. We have used Node.js as the primary way to run JavaScript code.
- ***Packages and Libraries***: Libraries including React.js and Express.js are essential for the functioning of the frontend and backend respectively. Our application would also rely on NPM packages for certain features. But, all of these dependencies would be bundled with the application as one complete package.
- ***APIs***: Our software uses the database by abstracting the database functionality through a software API written in Express.js. All major frontend interfaces mentioned in section 3.1 make use of this API to transfer data to the database.
- **OS:** For testers and developers, any OS capable of running Node.js is also a required software interface. Preferably, Ubuntu 18.04 and higher.

3.4.4 Communications Interfaces

- Our application will interface with an email sending service for the purpose of sharing credential parts with a user pool via email. This will employ a service typically using a communications standard like the SMTP protocol to send emails.
- Users of our application will also need to have a Web Browser to access the web app and use the application.

3.5 System Features

3.5.1 User Accounts

Description and Priority

Users should be able to create an account and login into the application. This will be done using either OAuth or using a JWT based login/signup interface.

Stimulus-response sequences

Stimulus: User clicks on the login link on the Landing Page

Response: Login page is displayed

Stimulus: User clicks on a link at the bottom of the login page which asks if he is a first-time user

Response: The user is guided to a registration page where he can register

Stimulus: If already registered, the user enters his credentials on the login page and clicks sign in or the user chooses to login with Google.

Response: User is redirected to the Dashboard if login is successful, else, the user is shown an error message.

Functional requirements

- REQ-1: Webpage where the user can enter his credentials and login.
- REQ-2: Webpage where the user can enter details and create an account
- REQ-3: Email verification to ensure the validity of the user.
- REQ-4: In case of forgetting the password, user should be in a position to recover it.

3.5.2 Home Page / Dashboard

Description and Priority

Users should be able to navigate through all main features of the application. This will be done via a dashboard page listing all the main features of the application and will be the first thing the user sees after login. It makes the application easy to navigate for the user.

From here the user will have access to the following interfaces -

1. Share a new key
2. View keys shared by user
3. View keys shared with user
4. View all recovery requests
5. Profile
6. Logout

Stimulus-response sequences

Stimulus: User clicks on “Share new key” menu item.

Response: User is guided to the “Share new key” page.

Stimulus: User clicks on “Keys shared with user” menu item.

Response: User is guided to the “Keys shared with user” page.

Stimulus: User clicks on “Keys shared by user” menu item.

Response: User is guided to the “Keys shared by user” page.

Stimulus: User clicks on “Profile” menu item.

Response: User is guided to a settings page.

Stimulus: User clicks on “Logout”.

Response: User is logged out of the application.

Functional requirements

REQ-1: Page where links to above mentioned pages are available. A logout mechanism will log the user out from the application and redirect him/her back to the login page.

3.5.3 Sharing A New Key

Description and Priority

Users should be able share a new key by sending credential parts to other users.

Stimulus-response sequences

Stimulus: Users enter their secret key. They then give a list of the people that will receive the key shards and the threshold number of shards required to recover the key. They click “Share”.

Response: The system sends the list of people and the threshold value to the backend. The backend logic splits the key into secure parts and an emailing system sends an email with the shards to the desired people. If everything succeeds, the user is shown with a success message, else a failure message.

Functional requirements

REQ-1: A webpage with an intuitive UI allowing the user to create a new secret key and share it with the participants.

REQ-2: Backend functionality for splitting the key securely using Shamir’s Secret Sharing Scheme and emailing functionality to send the key parts to the desired users.

3.5.4 Keys Shared By User

Description and Priority

Users should be able to access keys shared by them and have the option to change and reshare the key.

Stimulus-response sequences

Stimulus: Users will click on a page to view the keys shared by them.

Response: Users will be able to share all the keys shared by them. Each listing will also show an option to “Reshare” that key again.

Stimulus: Users click on “Reshare”.

Response: Users will be guided through a form where they can select receivers again and share the key parts again.

Functional requirements

REQ-1: Backend functionality to retrieve keys shared by each user from the database.

REQ-2: Backend functionality to reshare key parts if resharing is requested.

3.5.5 Keys Shared With User

Description and Priority

Users should be able to access keys shared with them.

Stimulus-response sequences

Stimulus: Users will click on a page to view the keys shared with them.

Response: Users will be able to share all the keys shared with them. Each listing will also show an option to “Request Recovery” for the key.

Stimulus: Users click on “Request Recovery”.

Response: If all other users who hold key parts are sent a recovery notification successfully, it will show success else, it will show failure and ask the user to try again.

Functional requirements

REQ-1: Backend functionality to retrieve keys shared with each user from the database.

REQ-2: Backend functionality to send an in-app or email-based notification to all users with whom the key was shared originally if “Request Recovery” is clicked by the user.

3.5.6 Recovery Requests

Description and Priority

Users should be able to view if the recovery of any key that they have a part of is requested and be able to grant/revoke access (i.e., be able to say if they consent to the recovery or not).

Stimulus-response sequences

Stimulus: Users will click on a page to view if they have any recovery requests pending.

Response: Users will be able to share all the keys that were shared with them and have a recovery request pending on them. They will also see 2 buttons in each listing, one to “Grant” access and other to “Revoke”.

Stimulus: User clicks on “Grant”.

Response: User will be required to enter his key shard.

Stimulus: User clicks on “Revoke”.

Response: User will be required to enter his key shard.

Functional requirements

REQ-1: Backend and database functionality to keep track of which keys are being requested to recover.

REQ-2: Functionality to implement “Grant” and “Revoke”.

3.6 Other Nonfunctional Requirements

3.6.1 Performance Requirements

The performance of the application is highly dependent on the size of the input as well as network latency. Very large inputs are likely to take more time and slow network speeds can slow down the process of sending emails which is an important step in our application.

Although for all practical use cases, our application should run without any considerable performance or reliability issues.

3.6.2 Safety Requirements

The product will have to confer to the native data privacy and information standards in the regions where it will be put into use.

3.6.3 Security Requirements

Users will have 2 options to be authenticated, either through a username and password approach or using OAuth providers like Gmail.

We only store user account information on our server and a history of the key names shared. A key or any part of it is never stored on the server and is truly decentralised. However, since the mechanism of sharing the key is via email, it is the receiver's responsibility to ensure that the key shard is safe with him once it is shared with him and that no other party gains access to it.

3.6.4 Software Quality Attributes

The software will serve the intended purpose of providing a platform to manage and share secrets and is bound to work for all users that meet the hardware and software compatibility specifications.

1. ***Correctness*** - The application employs a fool-proof concept of cryptographic secret sharing which is proven to be mathematically correct for providing an (n, k) consensus-based secret sharing scheme that has been described throughout this document.
2. ***Reliability*** - The application is bound to not produce any errors unless there is an issue with the email provider or with the database.
3. ***Usability*** - The application will be easy enough to use for any first-time user.

3.6.5 Business Rules

Since the product will be released as a free and open-source software, any modification and redistribution of it will also have to be provided free of cost and with an open-source licence.

Business rules regarding software in the country where the product will be put into use will need to be adhered to.

This software is an independent entity and is not affiliated with or does not seek to promote any other software.

Appendix A: Glossary

1. **Consensus-Based:** A consensus-based system refers to one where something happens only after approval from all or some members in a group of people.
2. **(n, k) scheme:** A consensus scheme which needs the approval of only k out of n people.
3. **Key:** A secret that needs to be protected from unauthorized access. For e.g. - A credential.
4. **Shard:** A key part, produced after a key is given to Shamir's Secret Sharing algorithm.
5. **Shamir's Secret Sharing Scheme:** A cryptographic scheme described in the paper "How to Share A Secret" by Adi Shamir, linked in the references section.
6. **JWT:** A data encoding scheme, useful in user authorization. (https://jwt.io/)
7. **OAuth:** A user authorization scheme, used by many authentication providers. (https://oauth.net/)

4. Literature Review

Research Related Papers:

Year of Paper	Title of the paper	Journal/conference details	Methodology used	Proposed idea	Advantages/ achieved objectives in paper	Disadvantages/ Limitations
1979	**Safeguarding Cryptographic Keys**	International Workshop on Managing Requirements Knowledge (MARK)	Mathematical modelling using Hyperplane Geometry	The secret is split into n parts based on an approach based on hyperplane geometry. Two nonparallel lines in the same plane intersect at exactly one point. Three nonparallel planes in space intersect at exactly one point. More generally, any *n* nonparallel (n −1) dimensional hyperplanes intersect at a specific point. The secret may be encoded as any single coordinate of the point of intersection. The secret can be recovered if any k parts are available.	Highly effective and secure. Each part carries the same amount of information. Avoids single point of failure while also dealing with the problem of consensus. Most ideal scheme among the schemes that were proposed very early on in this field.	Computationally expensive. Complex to implement. Many high dimensional mathematical calculations involved. Similar schemes with better running times exist. Each key part is k times larger than the original secret, where k is the threshold number of people.

1979	**How To Share A Secret**	Communications of the ACM	Mathematical modelling using Polynomial Interpolation	Polynomial Interpolation – Find k points which uniquely describe a k+1-order polynomial such that the y intercept is the secret. Take any n points on the polynomial as a basis to distribute parts to n people. Even if k points are available, the polynomial can be described and the secret can be recovered.	Shifts approach from planes to polynomials potentially speeding up the algorithm. Not very complex mathematically, while also remaining secure. Avoids single-point-of failure	Consumes decent amount of computing power but less than the previous scheme. Not the most ideal method but good. A bit old, better methods have come up recently. Each key part is only as big as the original secret.
2012	**Security Limitations of Using Secret Sharing for Data Outsourcing**	IFIP Annual Conference on Data and Applications Security and Privacy	Highlights security issues when employing the algorithm	The paper shows that if shares are kept on a database, using a specific type of attack, they can be easily obtained by an attacker. It shows that while it may be convenient to save the shares on a database, it may not be safe to do so.	This algorithm proves that storing shares on a database is susceptible to attack. We use this paper’s results to decide if our application should store shares on the database or not.	It does not show possible alternatives of how the shares can be saved.
2014	**Performance Analysis of Various Secret Sharing Techniques**	International Journal of Science and Research	Comparative analysis using experiments	This paper compares between all the existing secret sharing algorithms to determine which one has the best performance with respect to both security and running time.	The objective achieved by this paper is that it provides an understanding with regards to which secret sharing scheme can be used for a practical implementation. It confirms our choice of going with Shamir’s Secret Sharing Scheme which was described in the second paper in this literature survey.	The paper does not provide much insight into any novel algorithms that might exist for this purpose.

2014	**Asynchronous Secret Reconstruction and Its Application to the Threshold Cryptography**	International Journal of Communications, Network and System Sciences	Correcting flaws in existing schemes	The objective of reading this paper was to understand the flaws with our chosen implementation i.e., Shamir's Scheme. In this paper, it is shown that when there are more than *t* users participating and shares are released asynchronously in the secret reconstruction, an attacker can always release his share last. In such a way, after knowing *t* valid shares of legitimate shareholders, the attacker can obtain the secret and can successfully impersonate to be a legitimate shareholder without being detected. A simple modification of Shamir's scheme is proposed to fix this problem.	The paper successfully highlights an important flaw with the scheme. It shows that the flaw can be corrected without much overhead. Knowing this flaw, we can find a way in which our application can avoid it.	Though a modification to the algorithm is proposed, other ways in which this flaw can be prevented aren't highlighted which may not require modification to the algorithm itself.
2014	**Secret Splitting Schemes: A Review**	International Journal of Computer Science and Mobile Computing	Comparative Analysis	The paper presents a comparative analysis of various secret sharing schemes and possible limitations of each scheme.	This paper provides a brief on all schemes and helps confirm the final choice for the scheme. It also shows some novel schemes which helped in identifying if any good alternatives were being left out.	Implementation details aren't provided and it only provides an overview.
2020	**Linear Secret Sharing Schemes**	Applied Cryptography and Network Security Conference (ACNS)	Trivial Secret Sharing	Split the key into n parts. Distribute each part to each person. Key can be recovered by collecting all parts of the key.	Very simple to implement. Not much computational effort required to split and recombine the key yet achieves basic consensus.	The proposed method is too simple and does not provide much security. Each part of the key is not same and may provide more information than the other. Leads to single point of failure.

Implementation Related Research Papers:

1999	**A Future-Adaptable Password Scheme**	Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference	Password Hashing	Using block ciphers with a hash function to come up with a powerful password hashing scheme to implement a secure one-way hash function. Using two techniques, Eksblowfish, a block cipher with a purposefully expensive key schedule, and bcrypt, a related hash function it shows how secure password-based systems can be implemented.	Password is not in plain text so can't be breached by anyone. The algorithm is such that it will be computationally very expensive to reverse the hash function and can safeguard the password from attackers.	Only research perspective explained, implementation details lacking.
2017	**Security Flows in OAuth 2.0 Framework: A Case Study**	International Conference on Computer Safety, Reliability, and Security	Comprehensive study of advantages of OAuth based systems	The paper shows the advantages of using OAuth in applications for the purpose of authorization, user verification and authentication.	The paper provides a comprehensive review of all features as well as flaws when implementing OAuth in an application.	The paper does not provide an idea about when OAuth should be preferred over other authentication techniques.
2020	**Comprehensive Analysis of React-Redux Development Framework**	International Journal of Creative Research Thoughts (IRJET)	In-depth analysis	Analysis of React.js for frontend development and Redux as a state management library.	Shows how React.js is by the best technology to incorporate the agile development of front end. Shows the benefits of using React-Redux and the benefits it provides in user experience.	Does not explain about possible use cases and does not provide specific examples.

Conclusion:

This literature review explores various secret sharing schemes from easy to complex ones in an attempt to find a scheme with the best balance of running time and security. Following the literature review, it can be concluded that the scheme described in the second paper i.e., Shamir's Secret Sharing Scheme would be the perfect scheme for our application since it provides a balance of both good security and reasonable running time, both of which are crucial to our application.

Papers 3, 4, 5 and 7 provide an insight into the performance of these schemes and the various advantages as well as flaws that come with each scheme. These papers reaffirm our choice of algorithm as it is seen to perform well given reasonable computing power, it also has only minor flaws which are easily correctible if the application is designed properly.

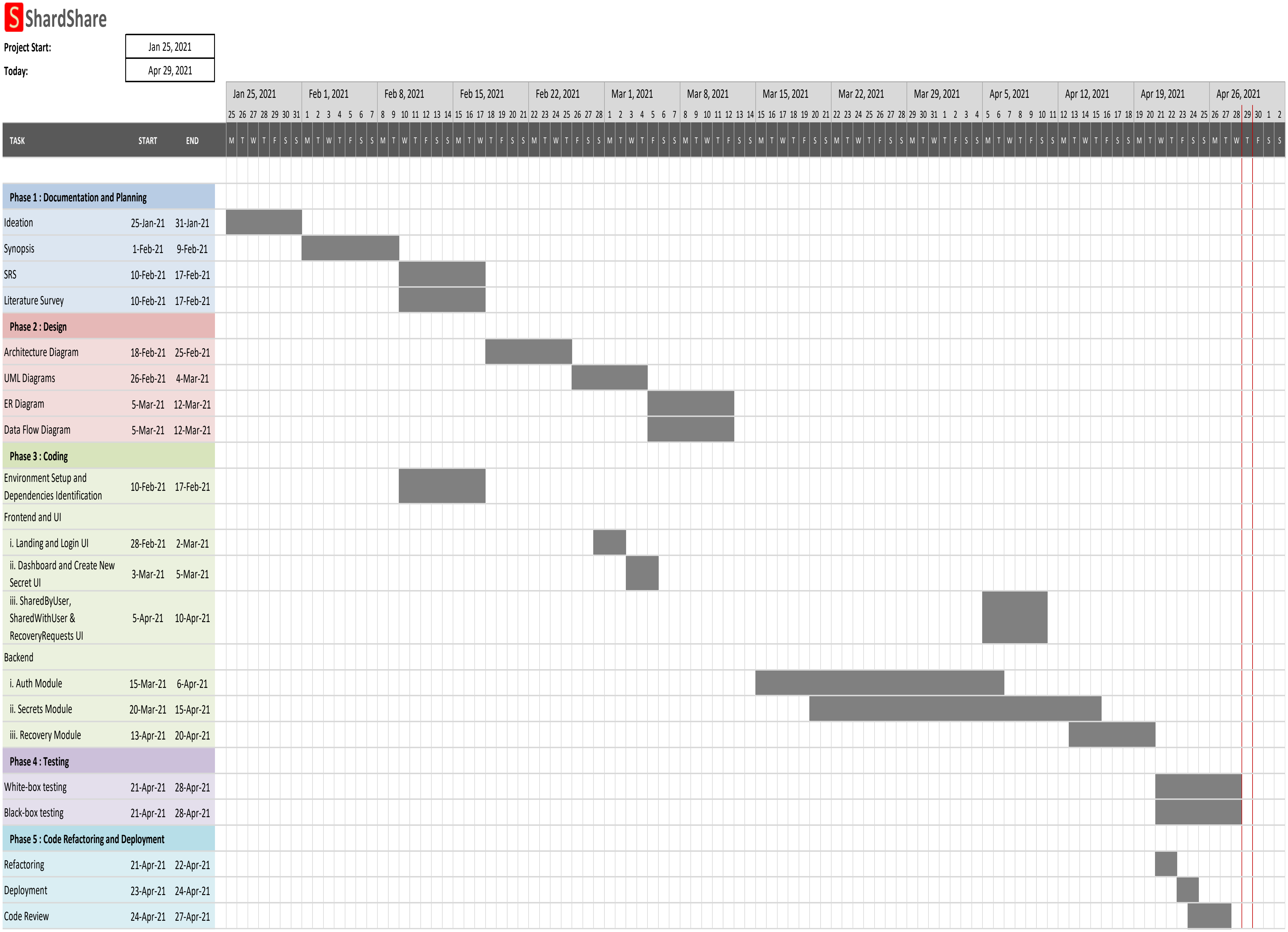
Paper 6 explores some other schemes to ensure that no good alternatives have been left out. Finally, papers 8, 9 and 10 provide some insight about the suitability of the tech stack like React.js and the supplementary protocols like bcrypt and OAuth that we plan to use in this project.

5. Proposed Statement

To implement a decentralised, consensus-based secret sharing application in order to overcome and provide a solution to various challenges with secret sharing. The application will be developed keeping the current use cases of credential sharing and their limitations into consideration and the data stored on the server will be kept to a minimum to provide a truly decentralised service.

6. Planning

Gantt Chart:



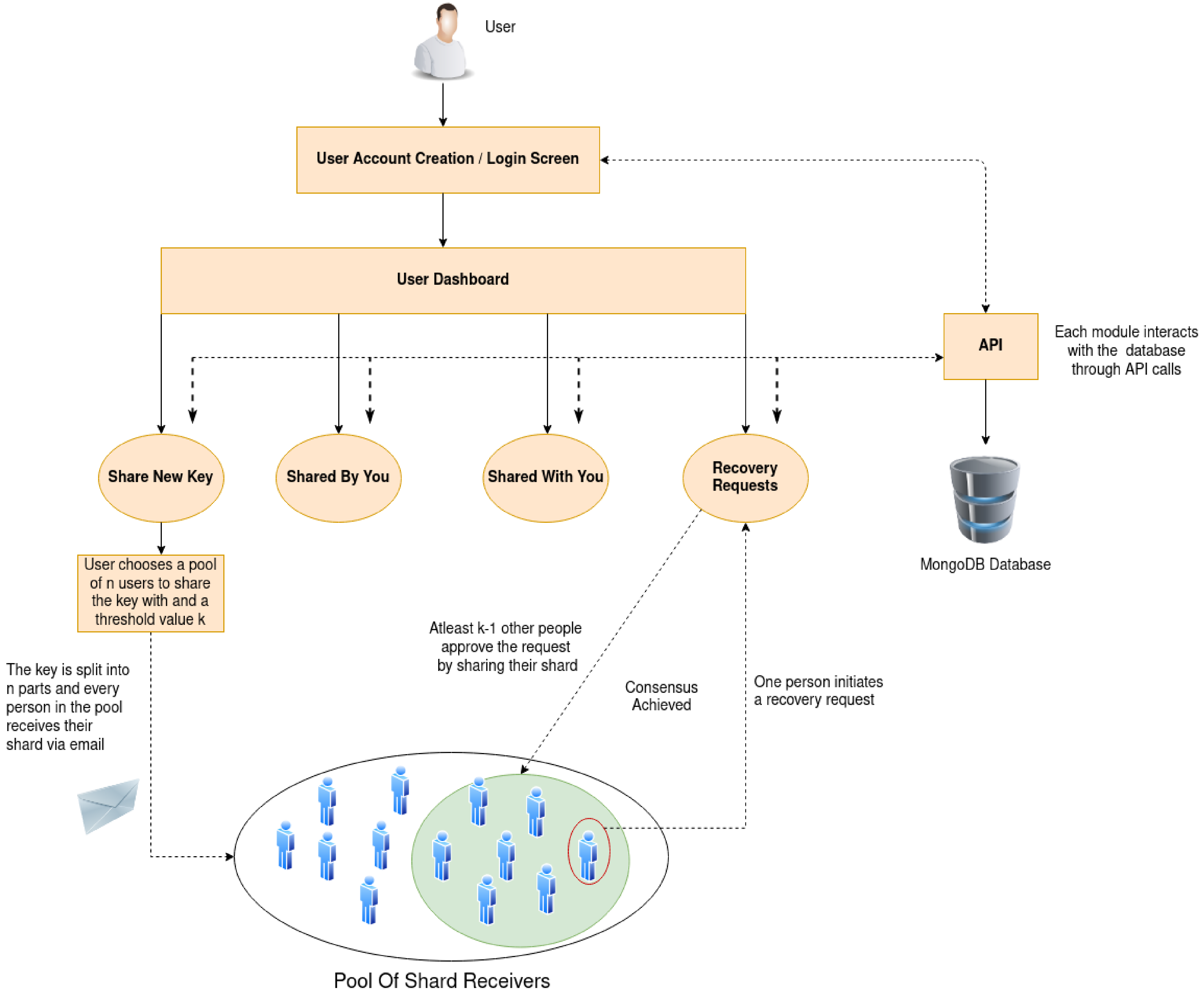
ShardShare

Project Start:	Jan 25, 2021
Today:	Apr 29, 2021

TASK	START	END
Phase 1 : Documentation and Planning		
Ideation	25-Jan-21	31-Jan-21
Synopsis	1-Feb-21	9-Feb-21
SRS	10-Feb-21	17-Feb-21
Literature Survey	10-Feb-21	17-Feb-21
Phase 2 : Design		
Architecture Diagram	18-Feb-21	25-Feb-21
UML Diagrams	26-Feb-21	4-Mar-21
ER Diagram	5-Mar-21	12-Mar-21
Data Flow Diagram	5-Mar-21	12-Mar-21
Phase 3 : Coding		
Environment Setup and Dependencies Identification	10-Feb-21	17-Feb-21
Frontend and UI		
i. Landing and Login UI	28-Feb-21	2-Mar-21
ii. Dashboard and Create New Secret UI	3-Mar-21	5-Mar-21
iii. SharedByUser, SharedWithUser & RecoveryRequests UI	5-Apr-21	10-Apr-21
Backend		
i. Auth Module	15-Mar-21	6-Apr-21
ii. Secrets Module	20-Mar-21	15-Apr-21
iii. Recovery Module	13-Apr-21	20-Apr-21
Phase 4 : Testing		
White-box testing	21-Apr-21	28-Apr-21
Black-box testing	21-Apr-21	28-Apr-21
Phase 5 : Code Refactoring and Deployment		
Refactoring	21-Apr-21	22-Apr-21
Deployment	23-Apr-21	24-Apr-21
Code Review	24-Apr-21	27-Apr-21

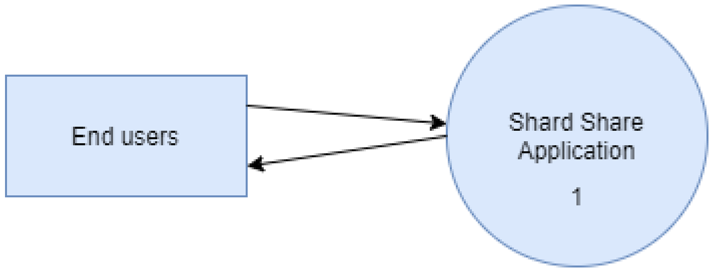
7. Design

7.1 Architecture Diagram:

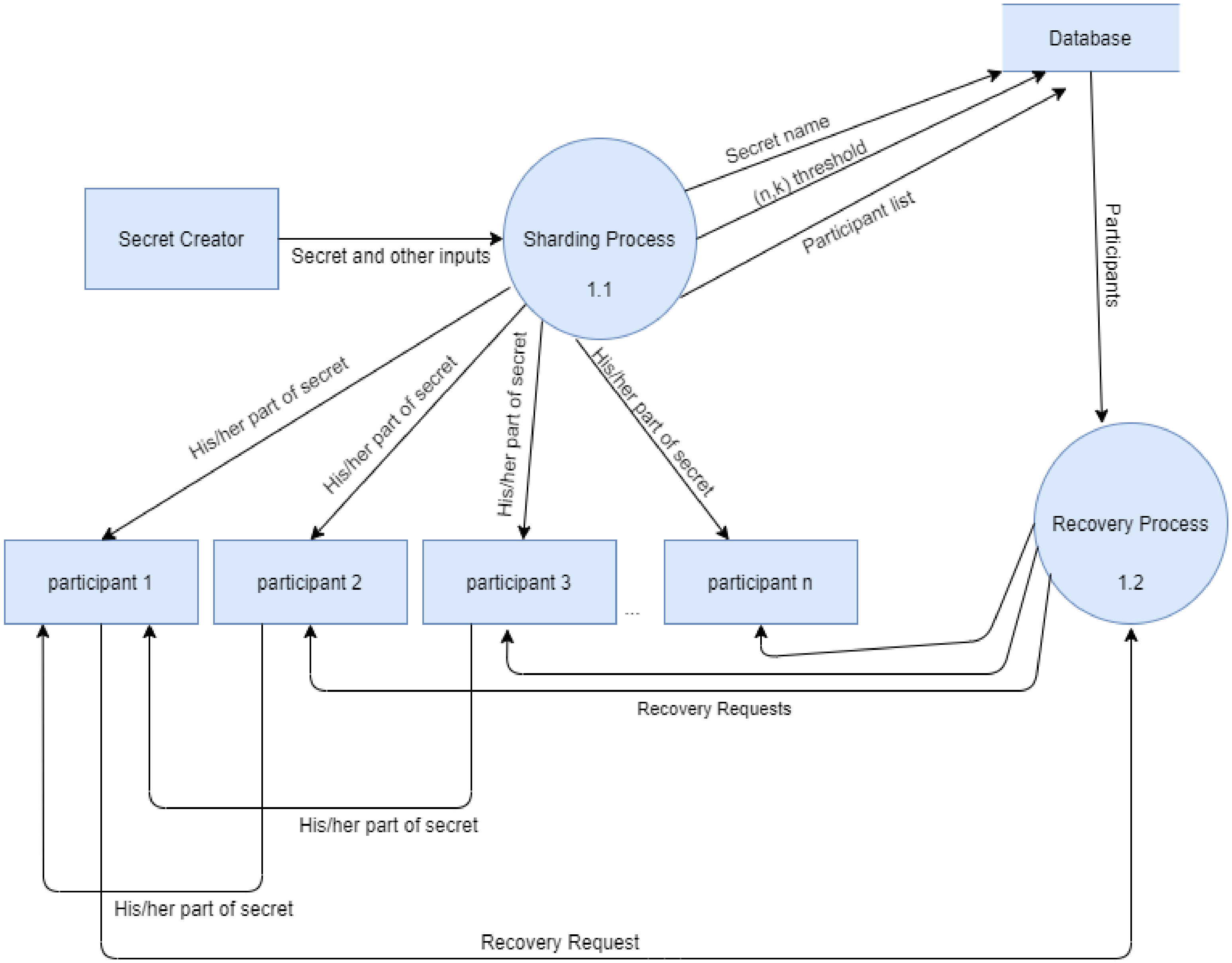

User
User Account Creation / Login Screen
User Dashboard
API
Each module interacts with the database through API calls
Share New Key
Shared By You
Shared With You
Recovery Requests
MongoDB Database
User chooses a pool of n users to share the key with and a threshold value k
The key is split into n parts and every person in the pool receives their shard via email
Atleast k-1 other people approve the request by sharing their shard
Consensus Achieved
One person initiates a recovery request
Pool Of Shard Receivers

7.2 Data Flow Diagrams:

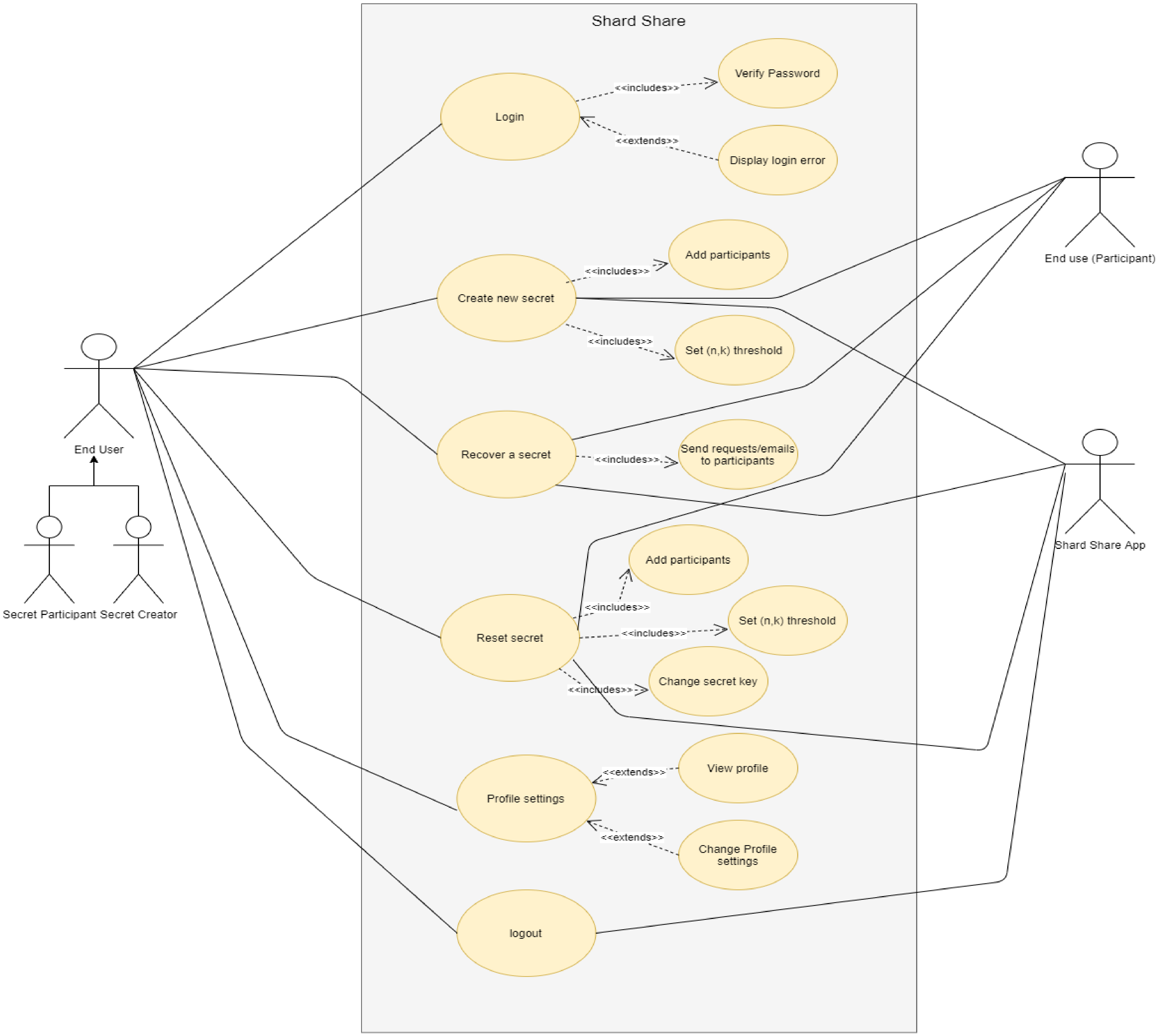
Level 0 DFD:



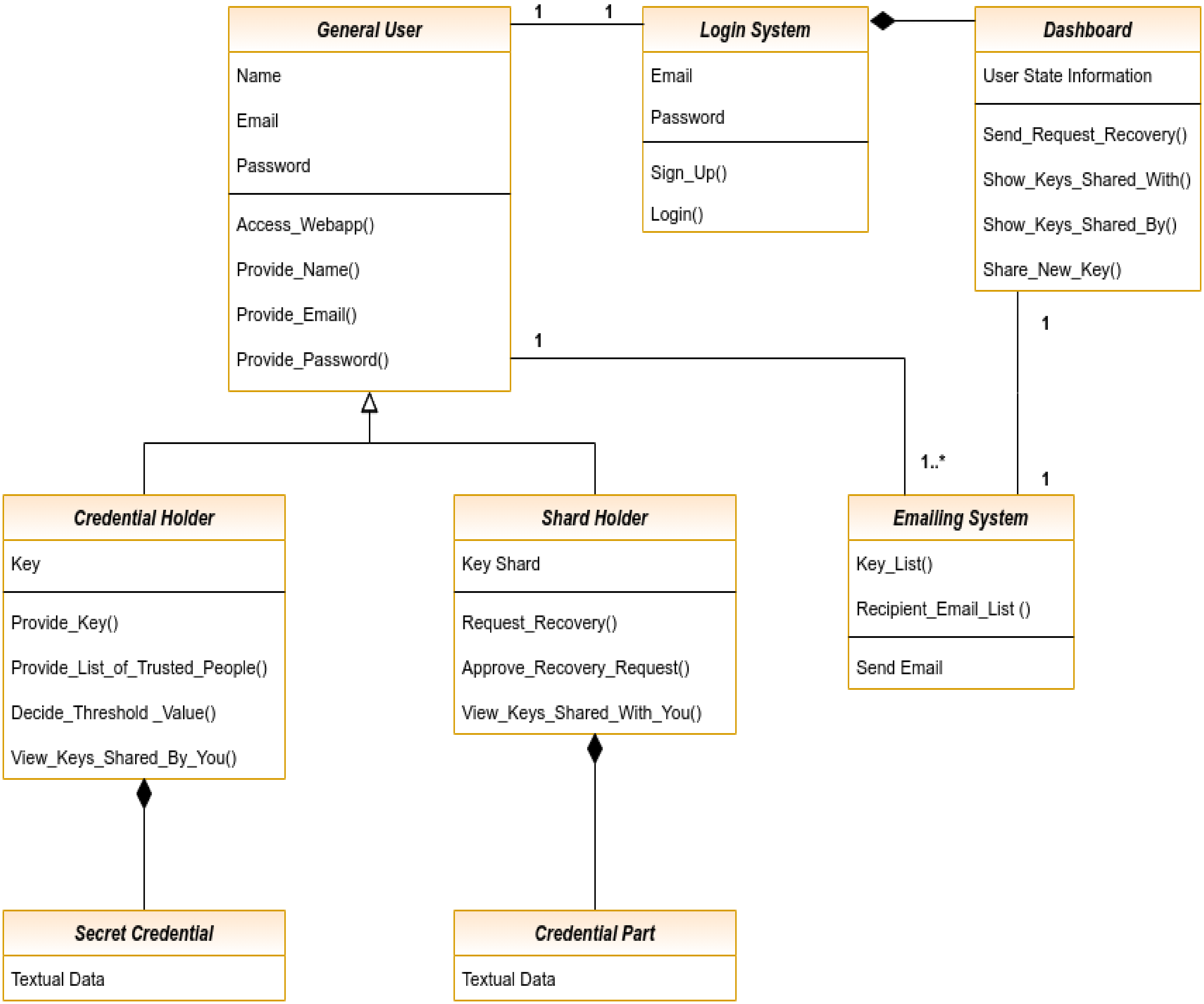
Level 1 DFD:



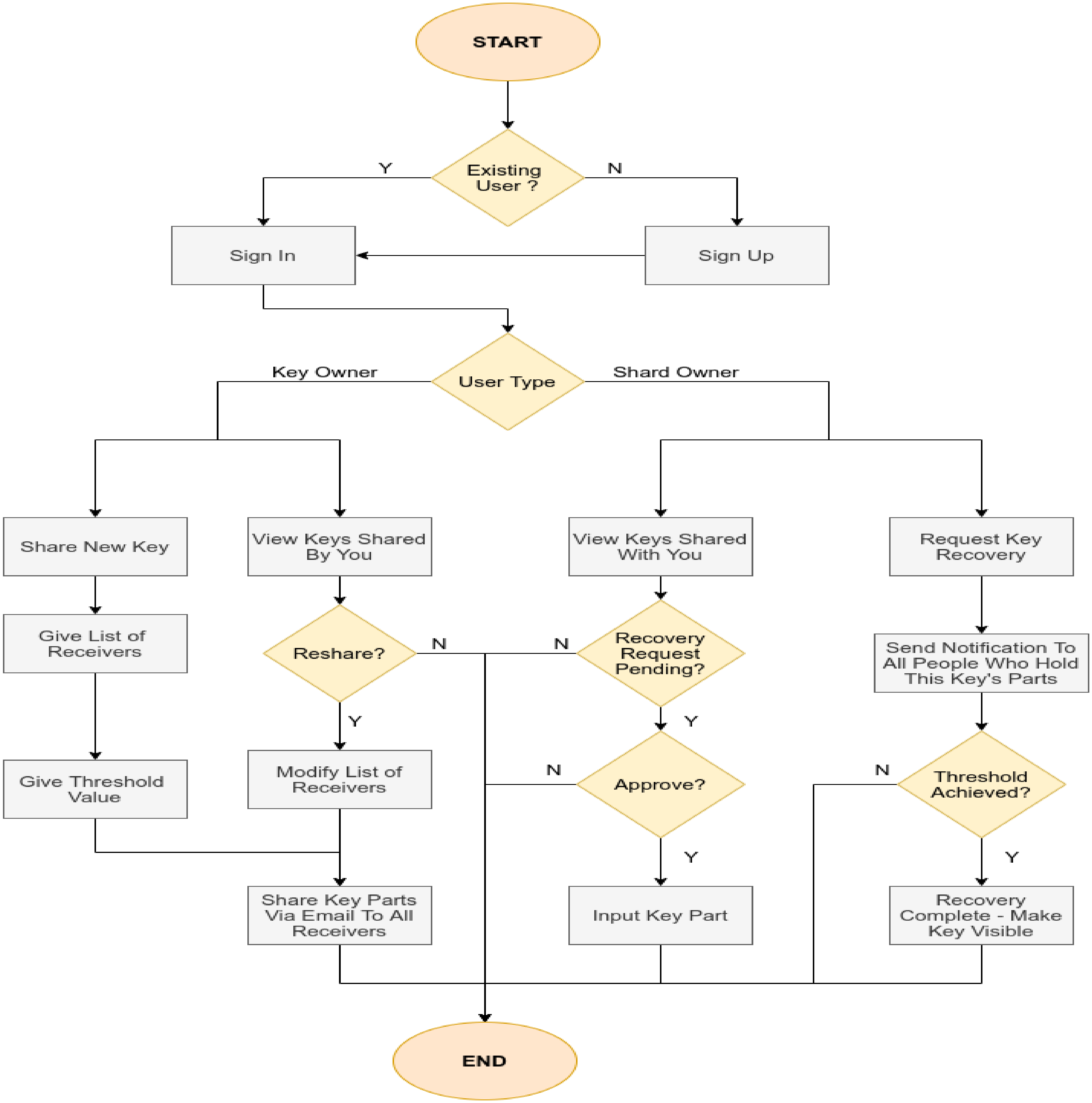
7.3 Use Case Diagram:



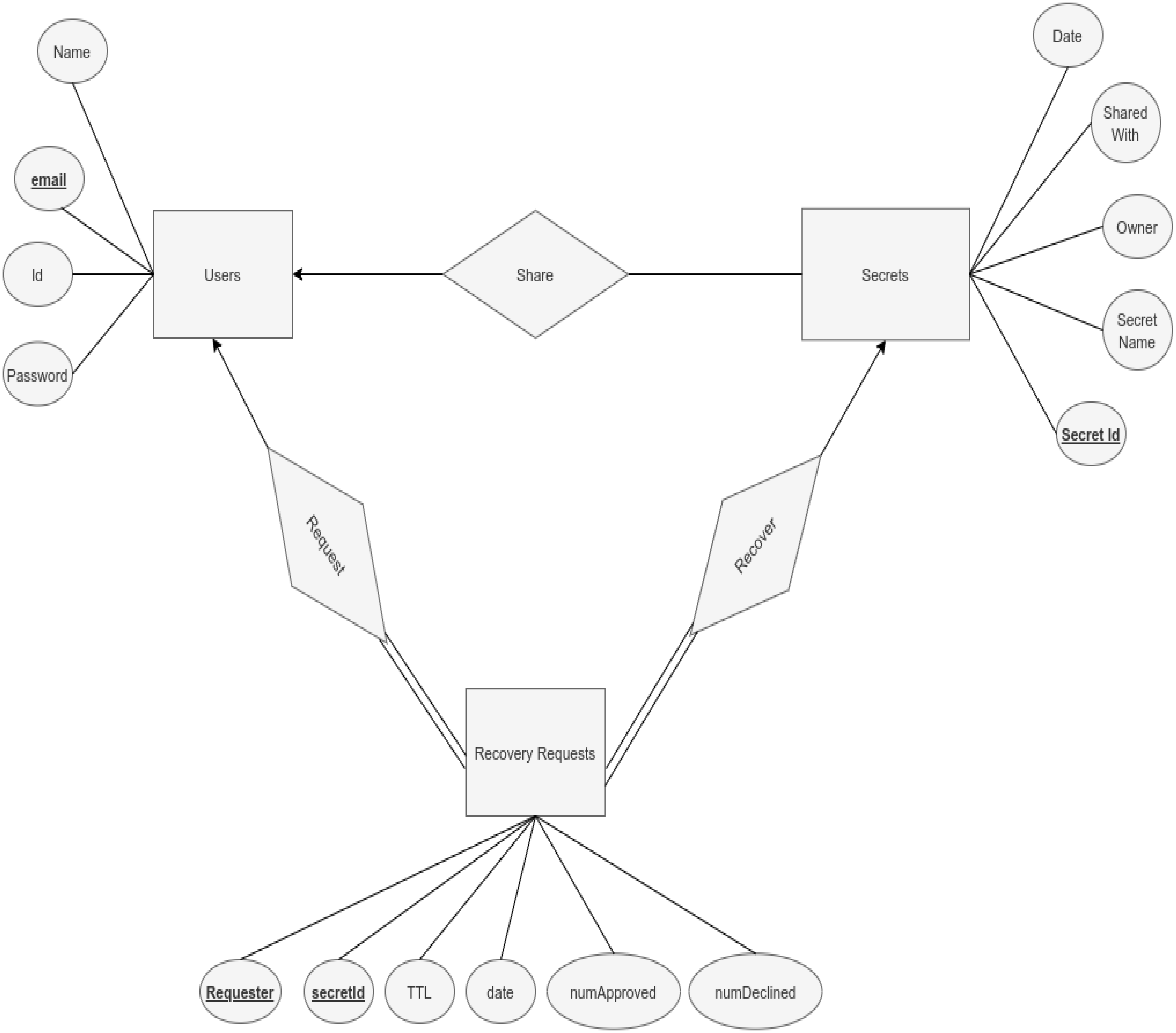
7.4 Class Diagram:



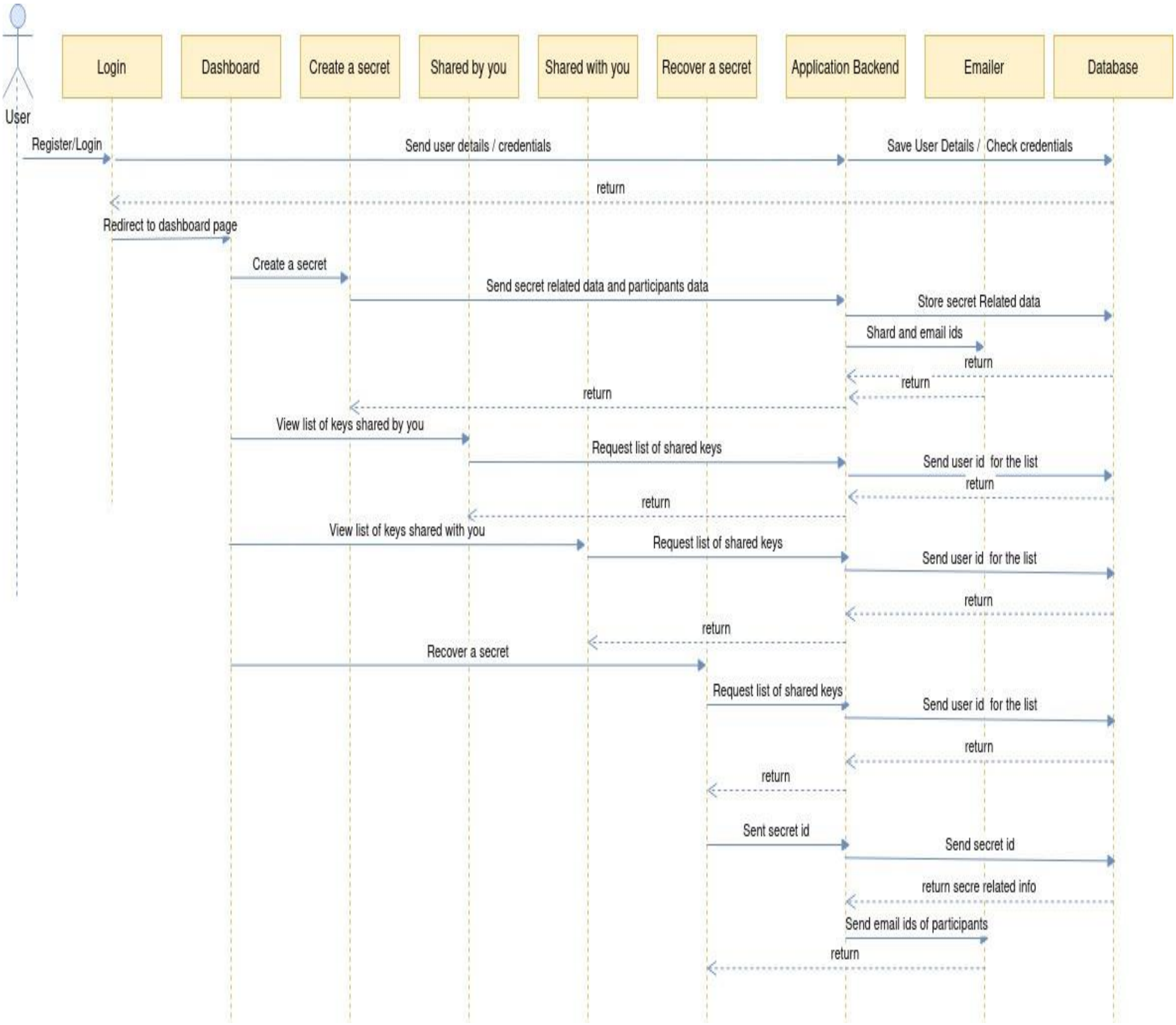
7.5 Activity Diagram:



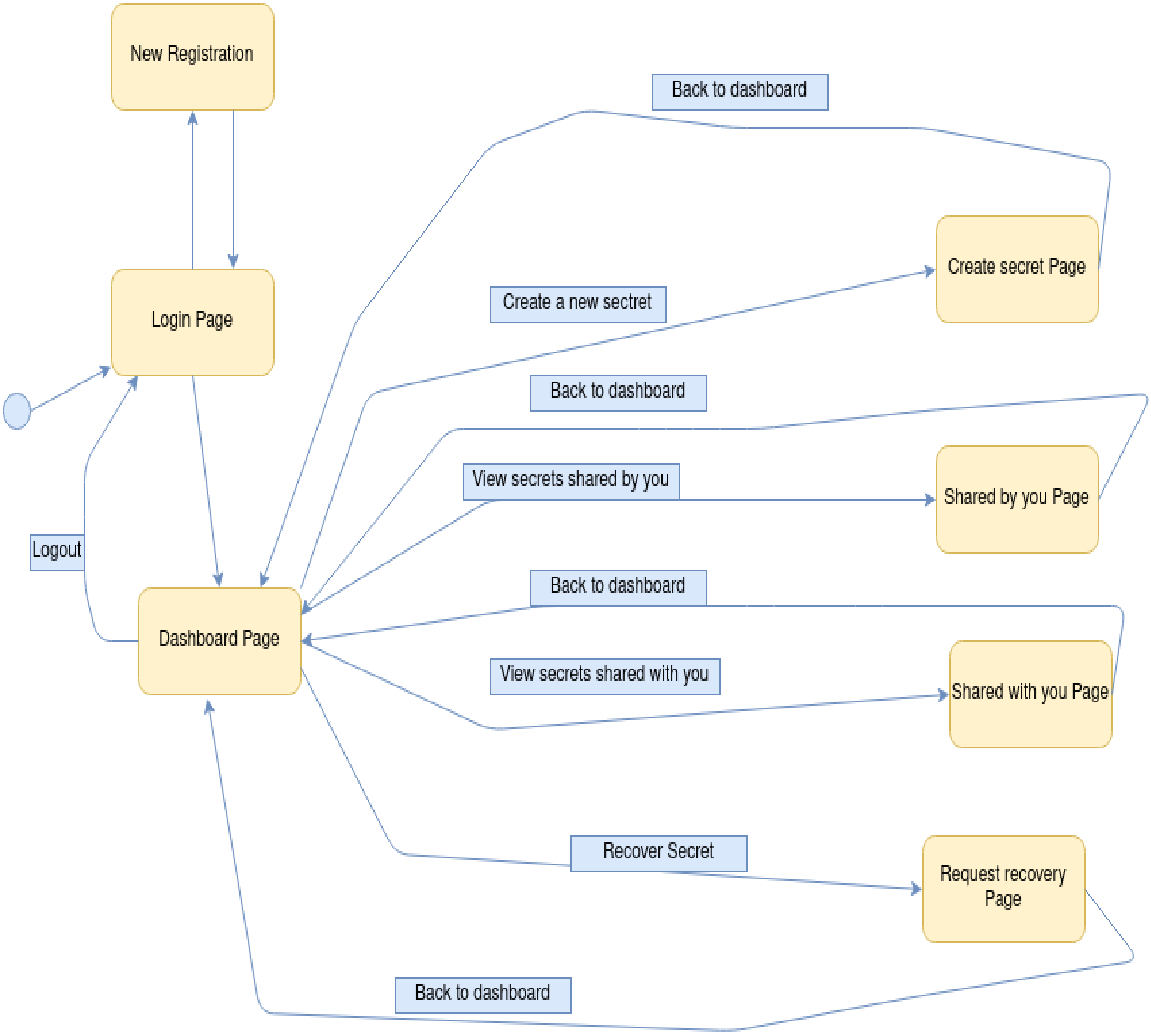
7.6 ER Diagram:



7.7 Sequence Diagram:


User
Login
Dashboard
Create a secret
Shared by you
Shared with you
Recover a secret
Application Backend
Emailer
Database
Register/Login
Send user details / credentials
Save User Details / Check credentials
return
Redirect to dashboard page
Create a secret
Send secret related data and participants data
Store secret Related data
Shard and email ids
return
return
return
View list of keys shared by you
Request list of shared keys
Send user id for the list
return
return
View list of keys shared with you
Request list of shared keys
Send user id for the list
return
return
Recover a secret
Request list of shared keys
Send user id for the list
return
return
Sent secret id
Send secret id
return secre related info
Send email ids of participants
return

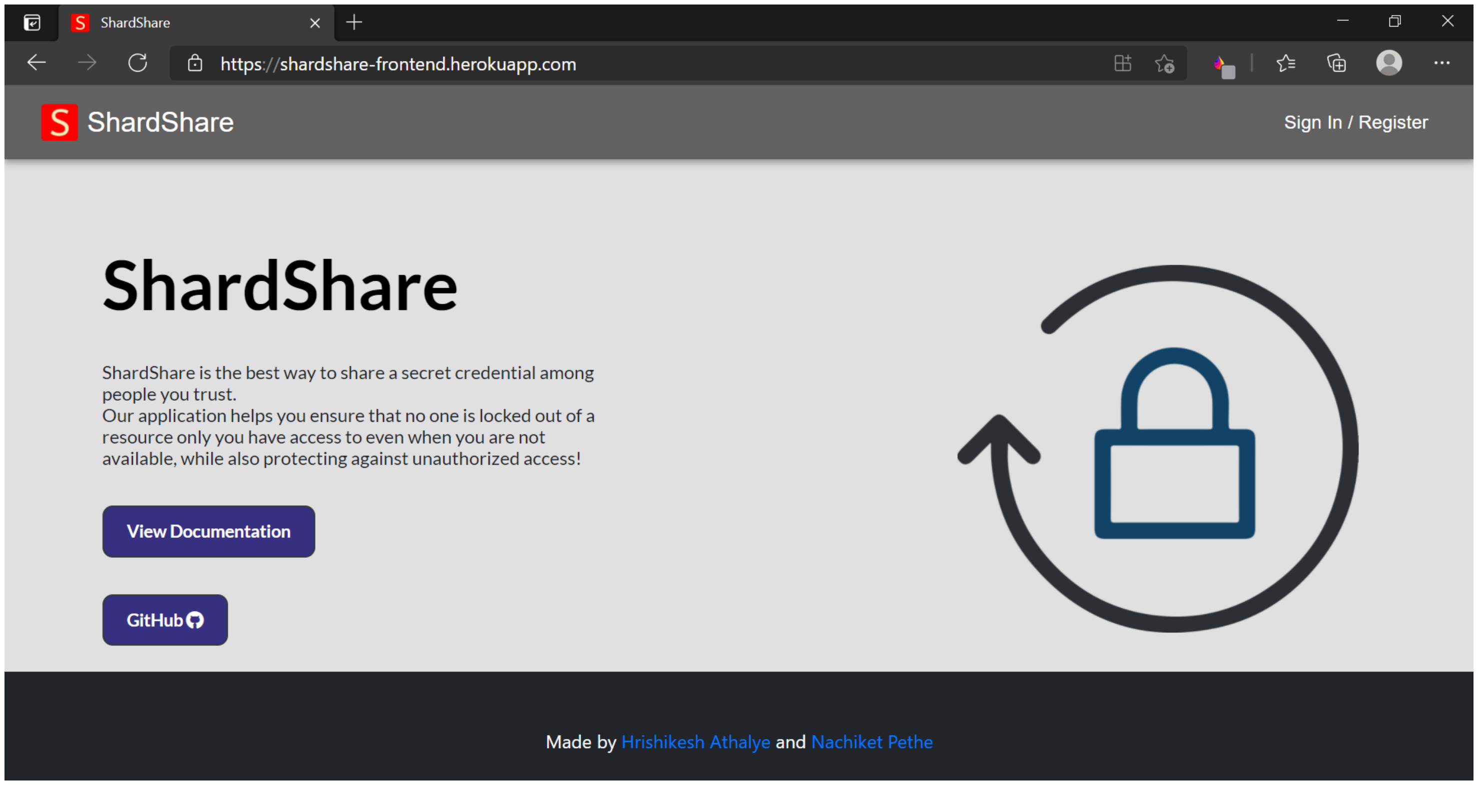
7.8 State Diagram:



8. Implementation

8.1 Web:

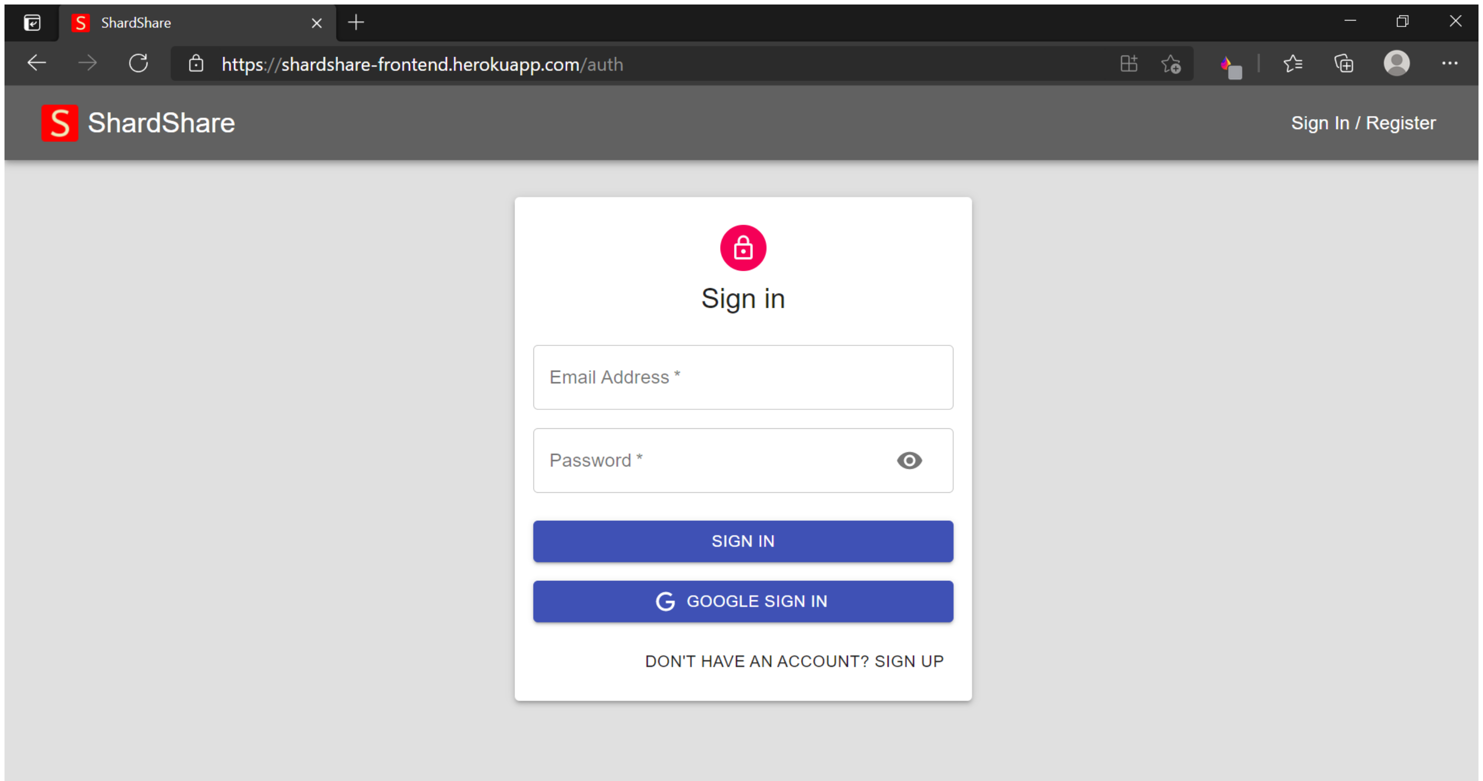
Landing Page



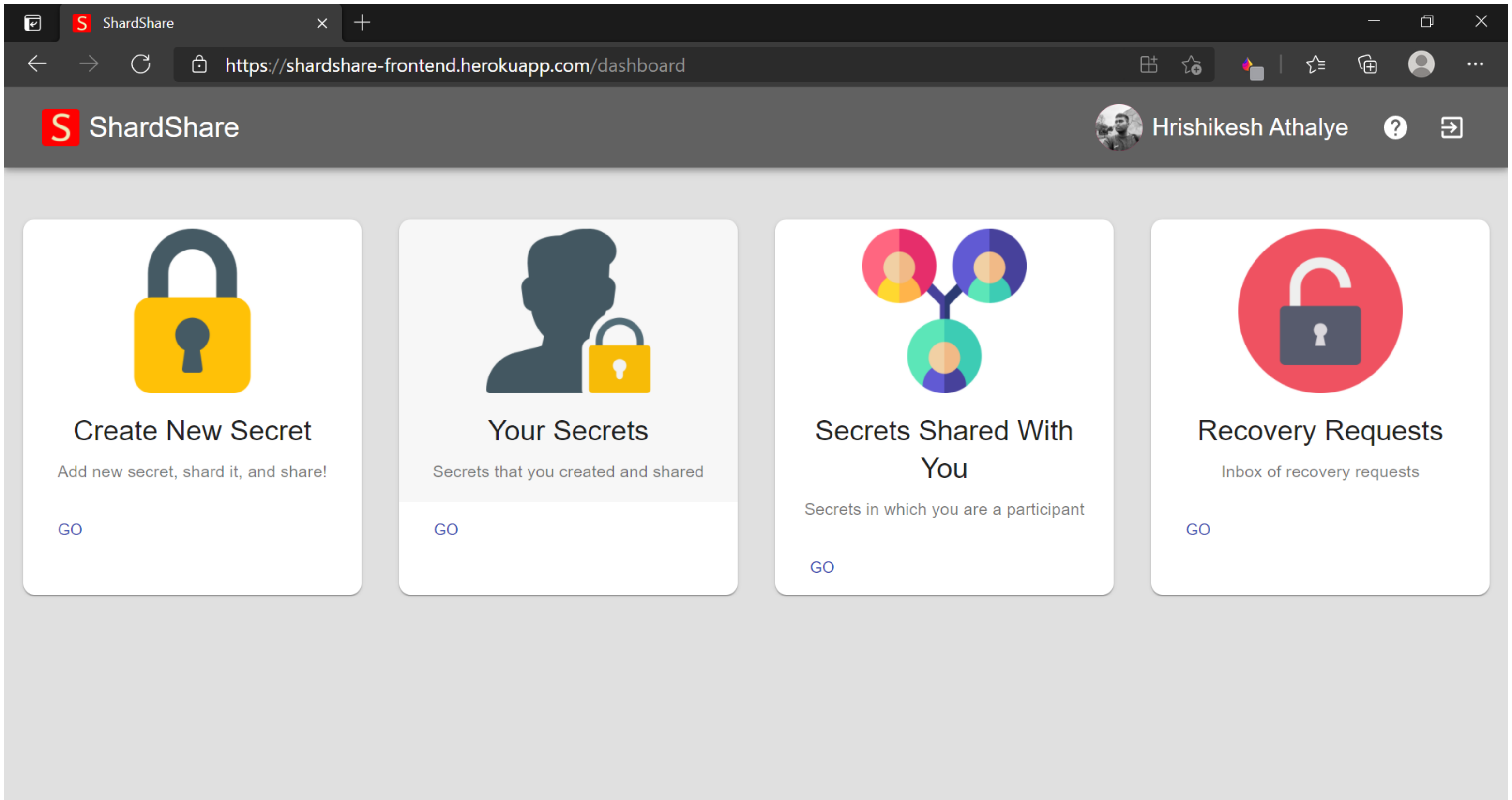
ShardShare is the best way to share a secret credential among people you trust.
Our application helps you ensure that no one is locked out of a resource only you have access to even when you are not available, while also protecting against unauthorized access!

Made by Hrishikesh Athalye and Nachiket Pethe

Login/Register



Dashboard



ShardShare   Hrishikesh Athalye

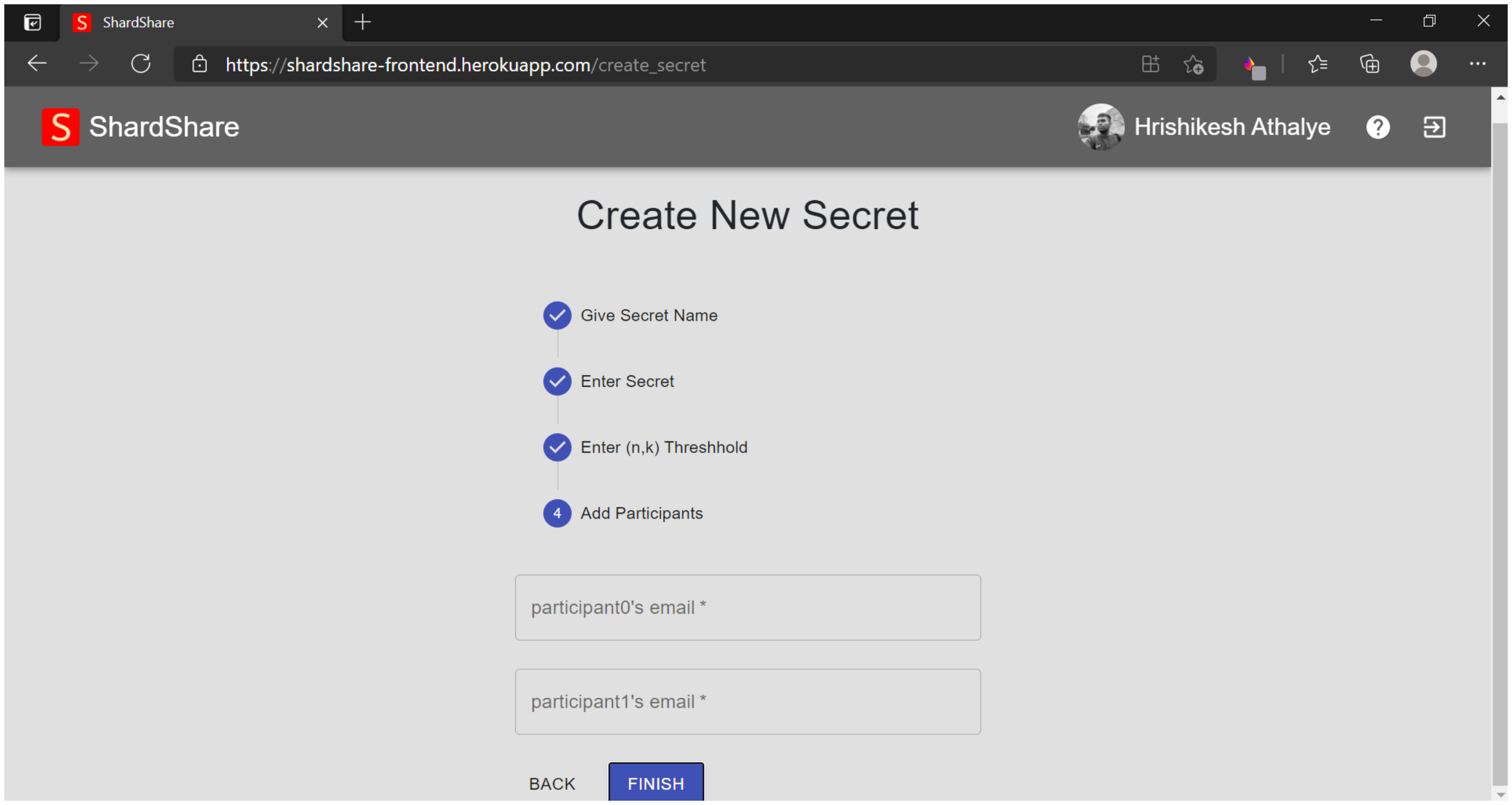
Create New Secret
Add new secret, shard it, and share!
GO

Your Secrets
Secrets that you created and shared
GO

Secrets Shared With You
Secrets in which you are a participant
GO

Recovery Requests
Inbox of recovery requests
GO

Create Secret



ShardShare   Hrishikesh Athalye

Create New Secret

✓ Give Secret Name

✓ Enter Secret

✓ Enter (n,k) Threshhold

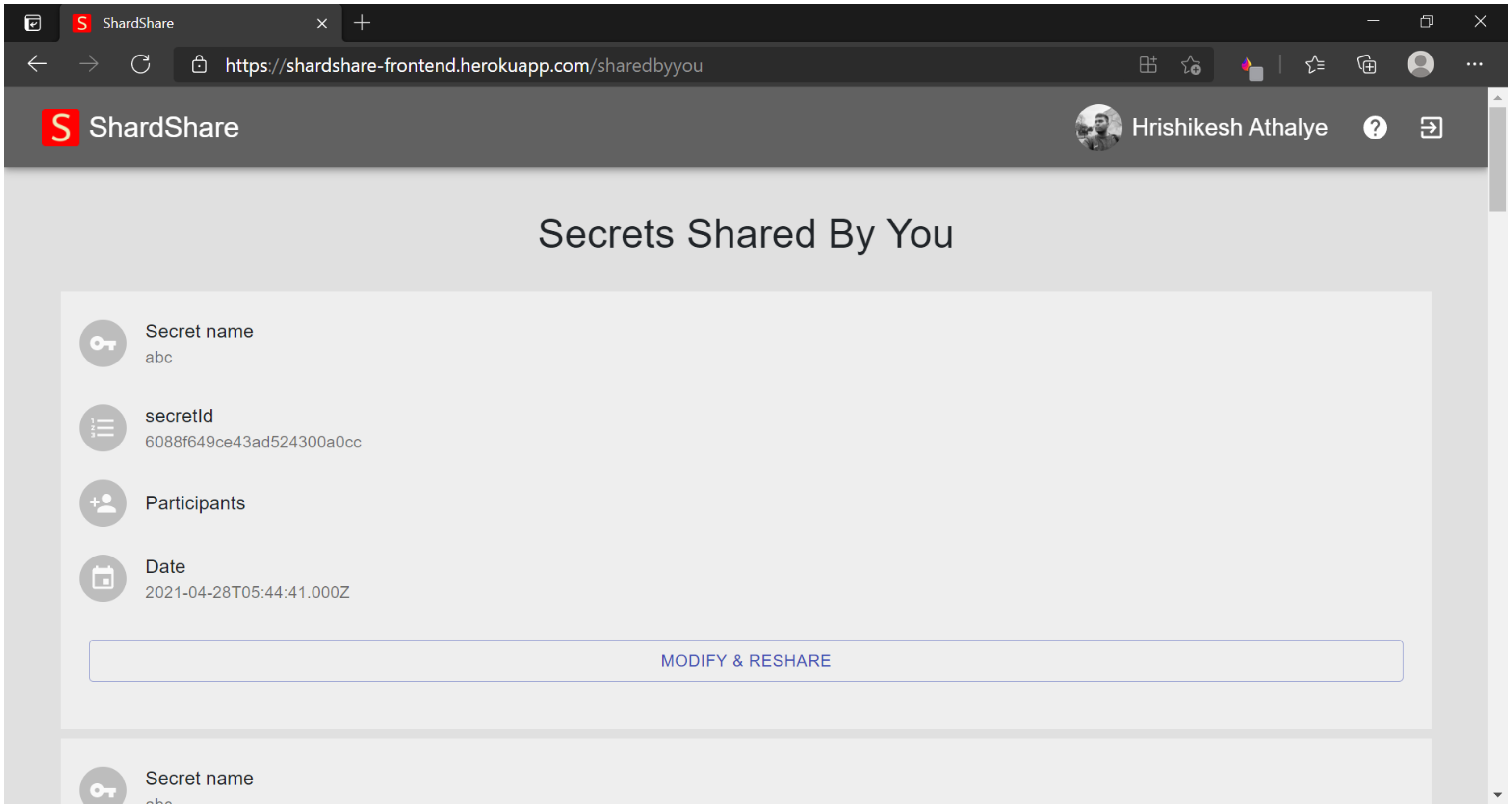
4 Add Participants

participant0's email *

participant1's email *

BACK   FINISH

Viewing keys shared by you



Secrets Shared By You

Secret name
abc

secretId
6088f649ce43ad524300a0cc

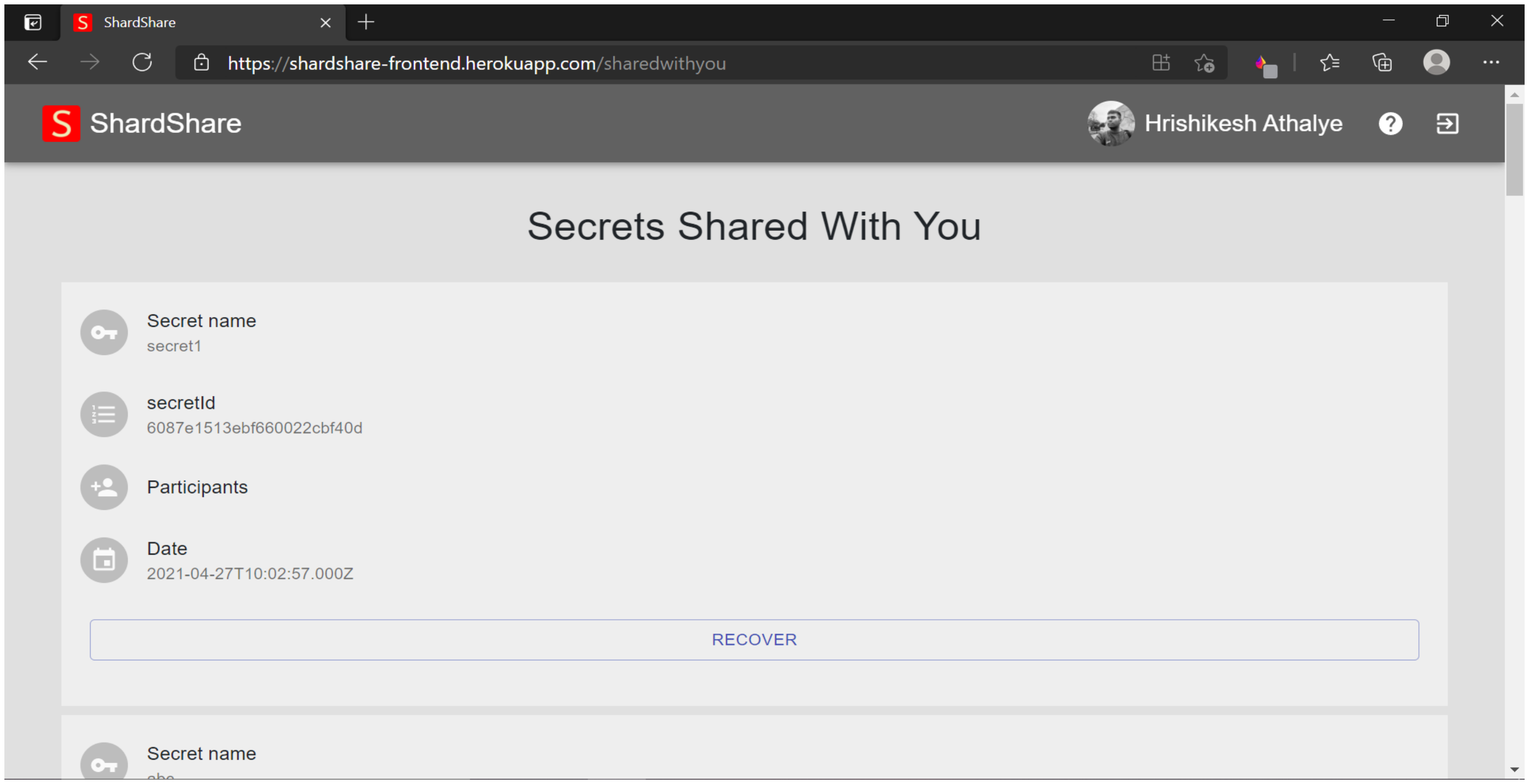
Participants

Date
2021-04-28T05:44:41.000Z

MODIFY & RESHARE

Secret name

Viewing keys shared with you



Secrets Shared With You

Secret name
secret1

secretId
6087e1513ebf660022cbf40d

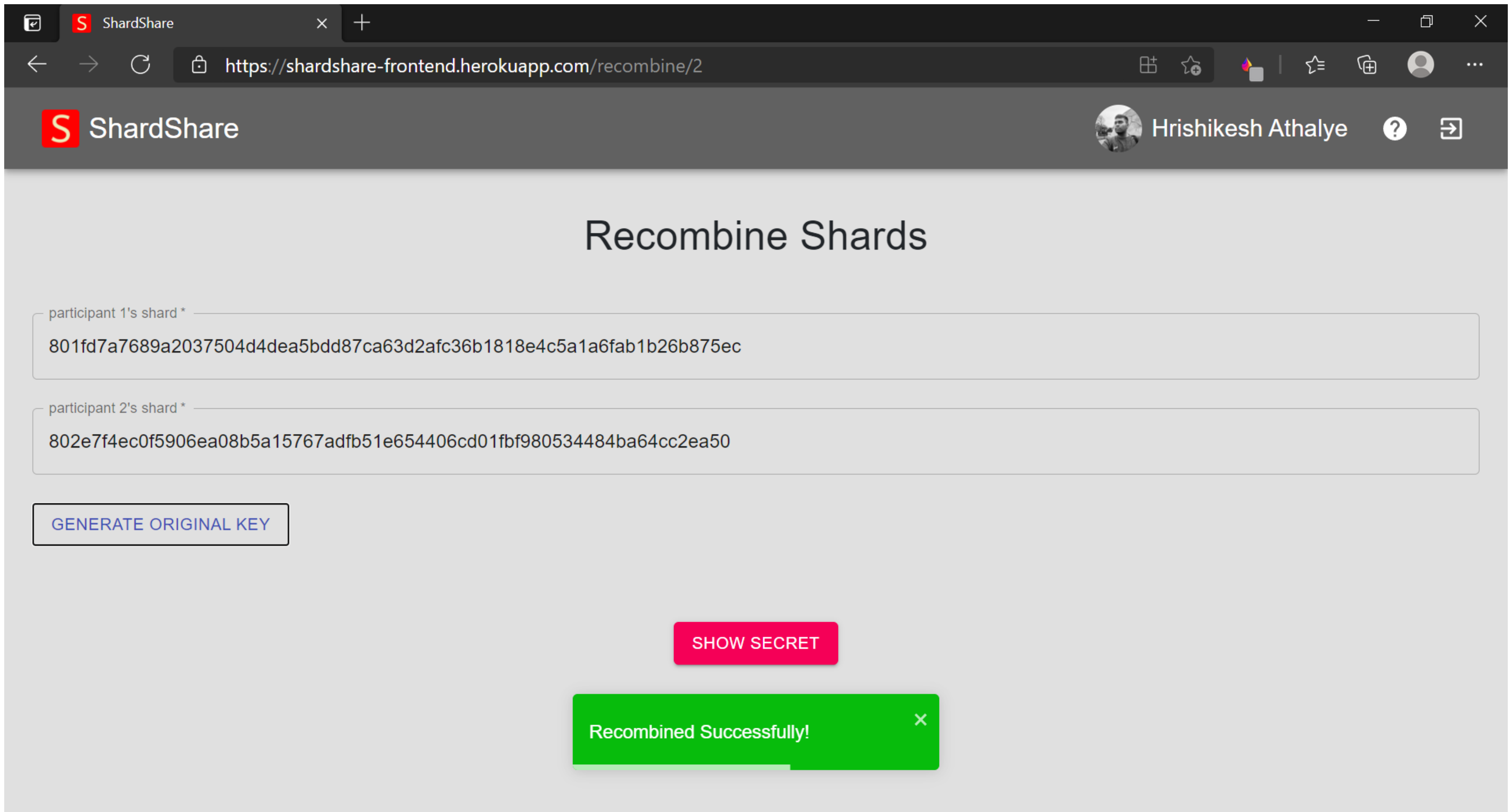
Participants

Date
2021-04-27T10:02:57.000Z

RECOVER

Secret name

Recovery process

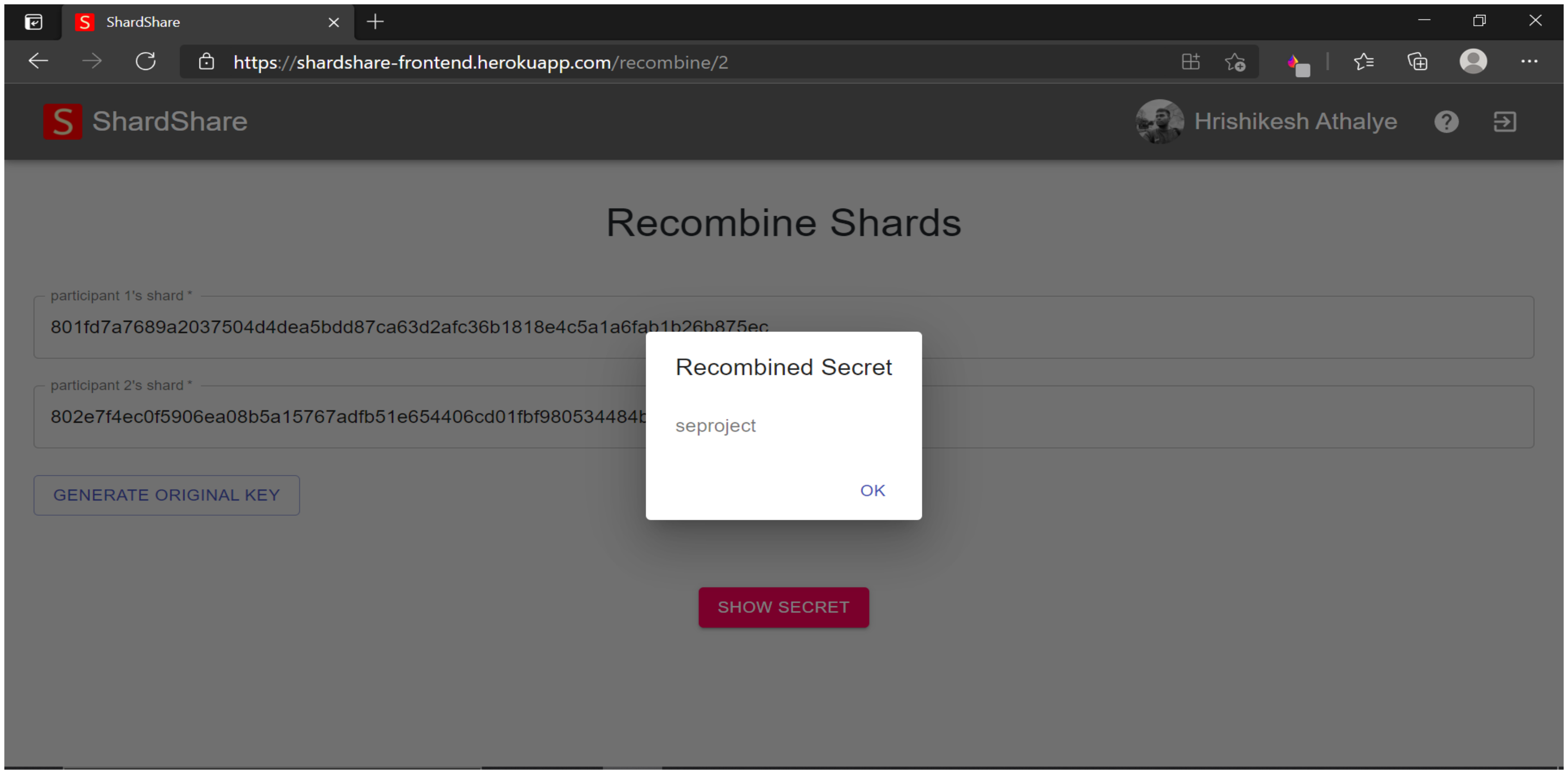


Recombine Shards

participant 1's shard *
801fd7a7689a2037504d4dea5bdd87ca63d2afc36b1818e4c5a1a6fab1b26b875ec

participant 2's shard *
802e7f4ec0f5906ea08b5a15767adfb51e654406cd01fbf980534484ba64cc2ea50

Recombined Successfully!



Recombine Shards

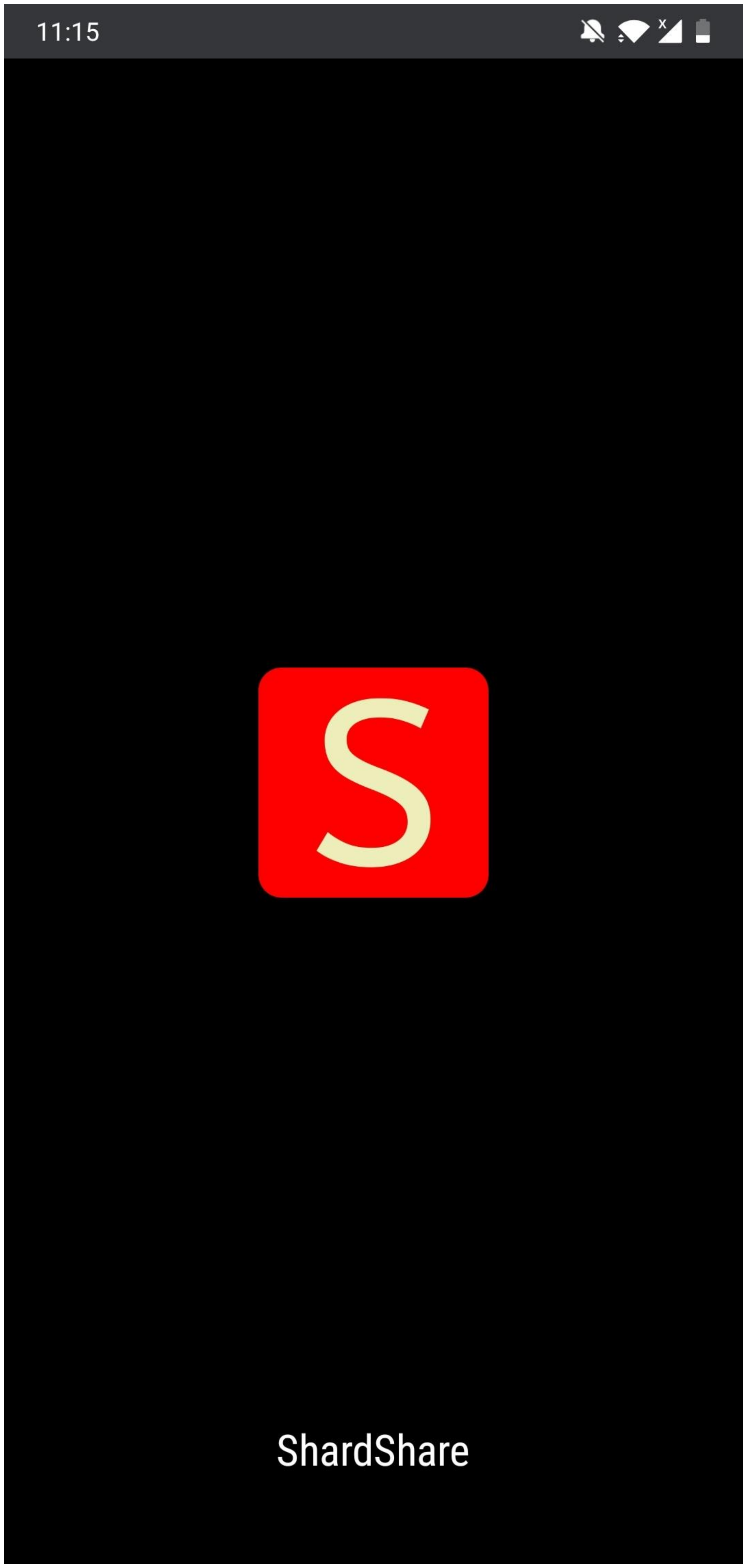
participant 1's shard *
801fd7a7689a2037504d4dea5bdd87ca63d2afc36b1818e4c5a1a6fab1b26b875ec

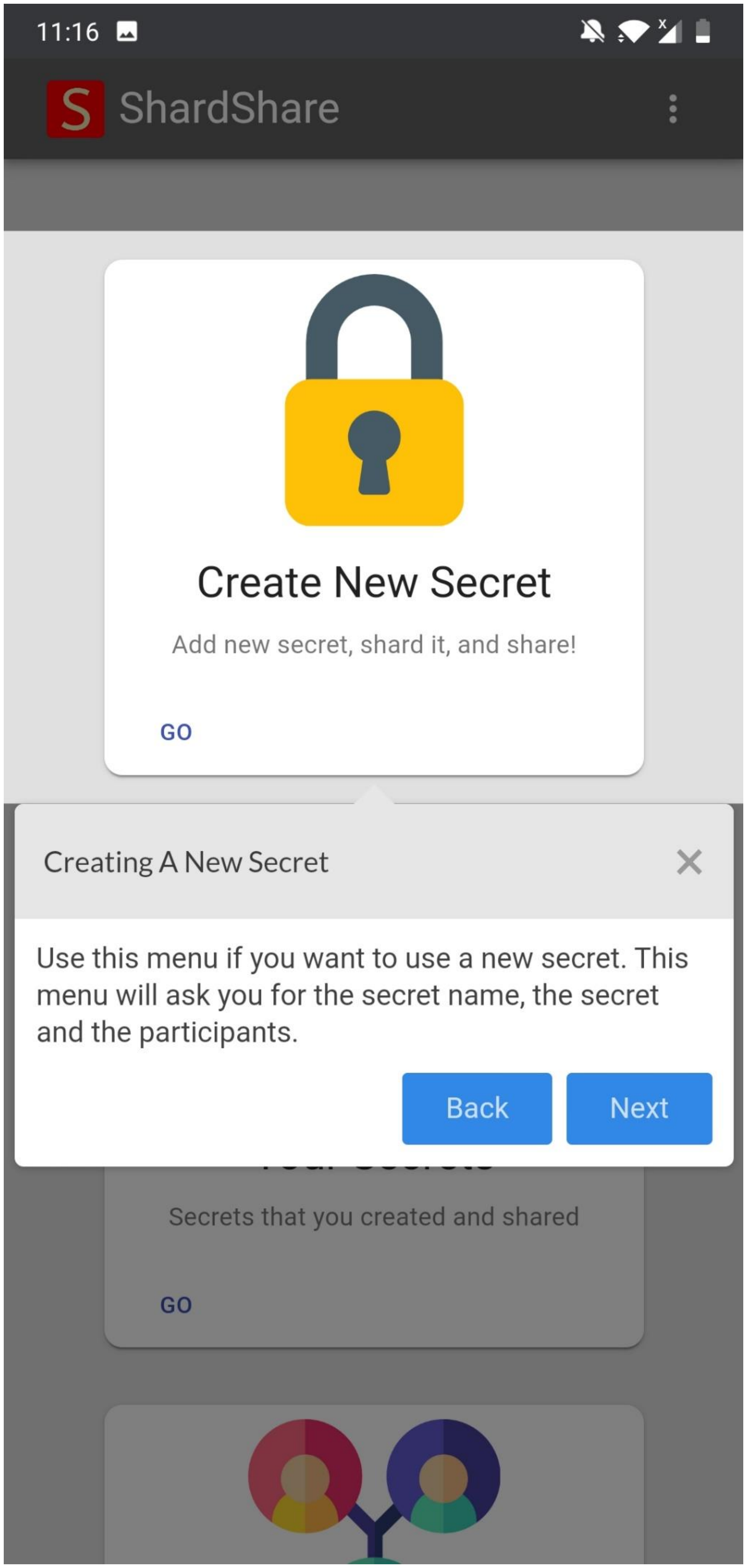
participant 2's shard *
802e7f4ec0f5906ea08b5a15767adfb51e654406cd01fbf980534484b

Recombined Secret

seproject

8.2 PWA on Android:





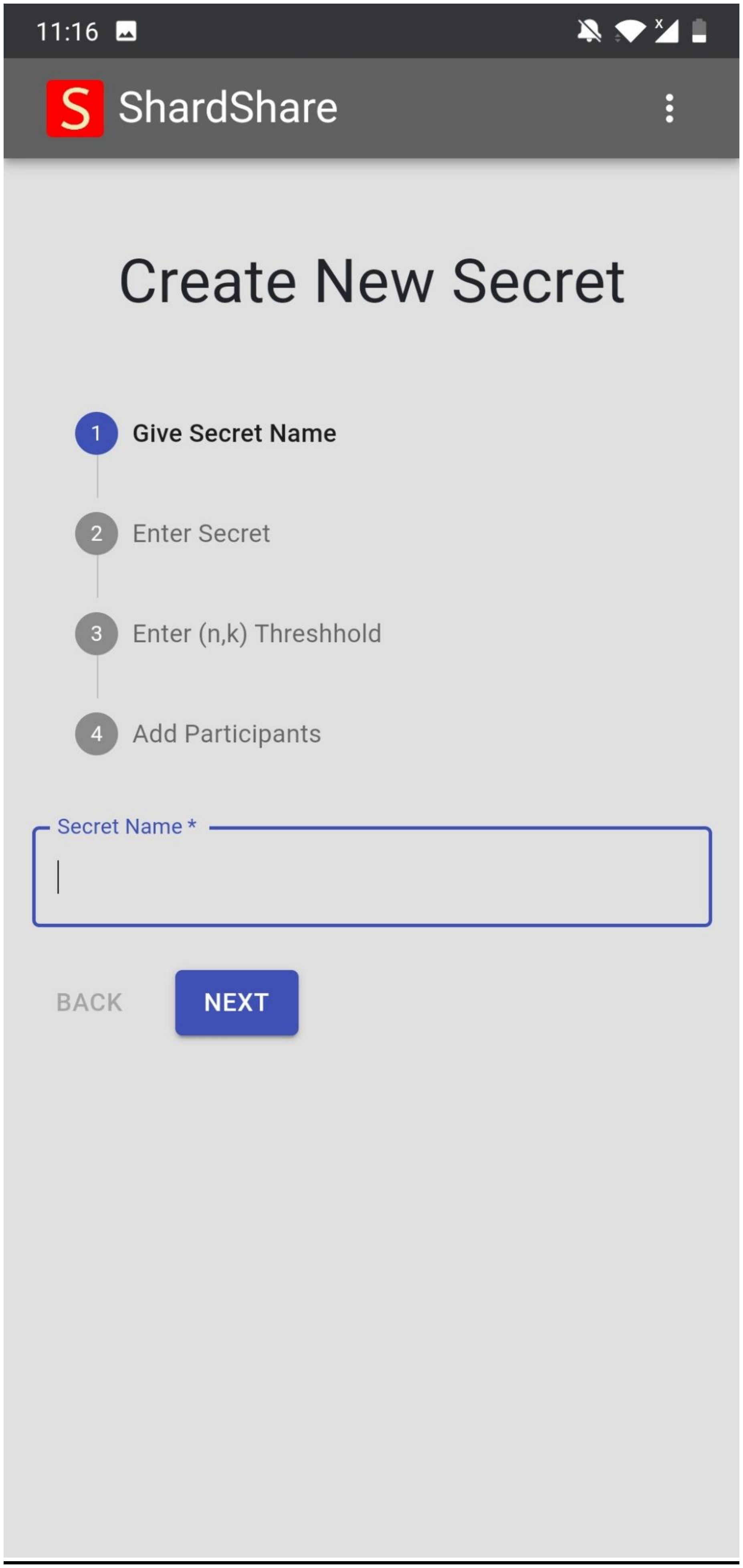
Create New Secret

Add new secret, shard it, and share!

Creating A New Secret

Use this menu if you want to use a new secret. This menu will ask you for the secret name, the secret and the participants.

Secrets that you created and shared



Create New Secret

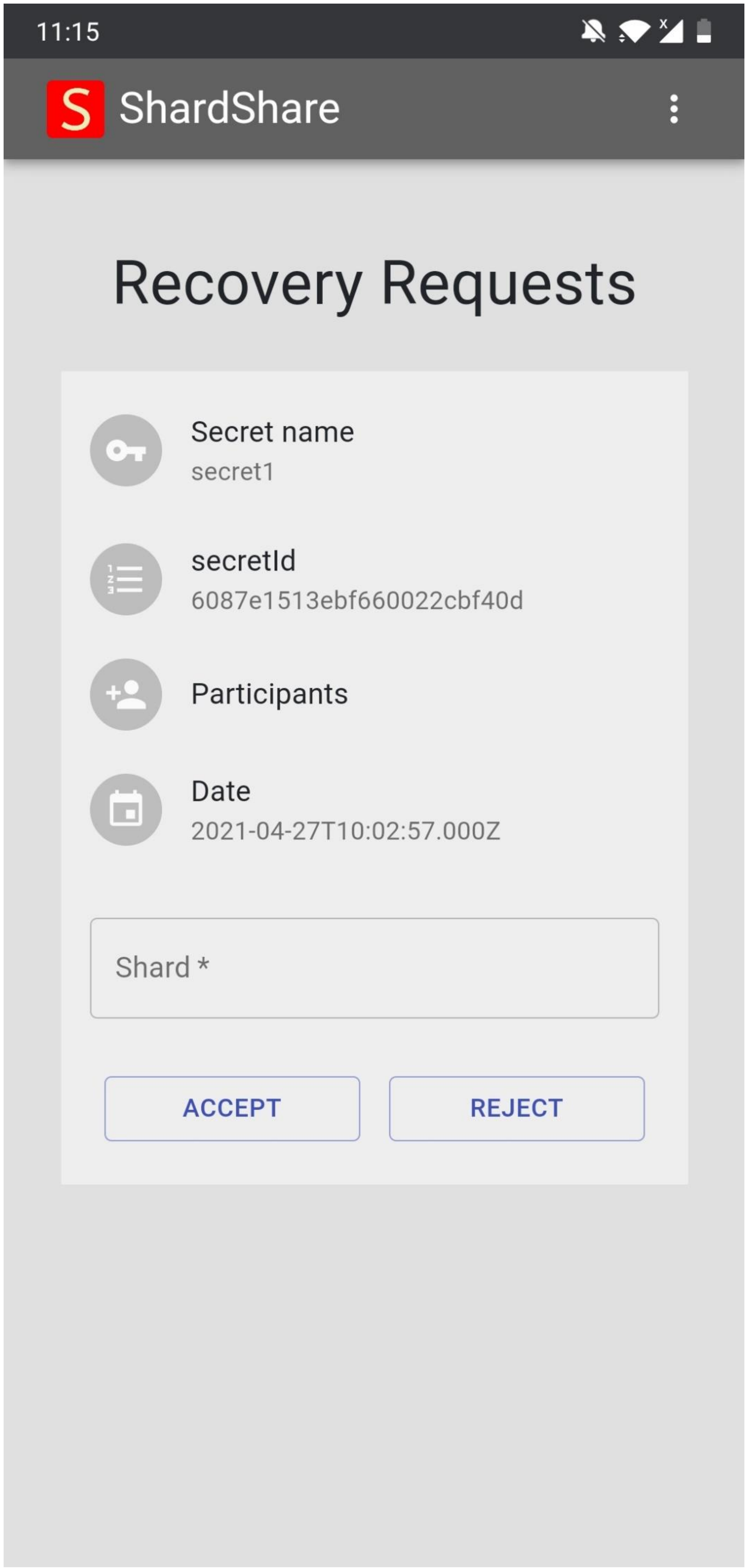
1 Give Secret Name

2 Enter Secret

3 Enter (n,k) Threshhold

4 Add Participants

Secret Name *



Recovery Requests

Secret name
secret1

secretId
6087e1513ebf660022cbf40d

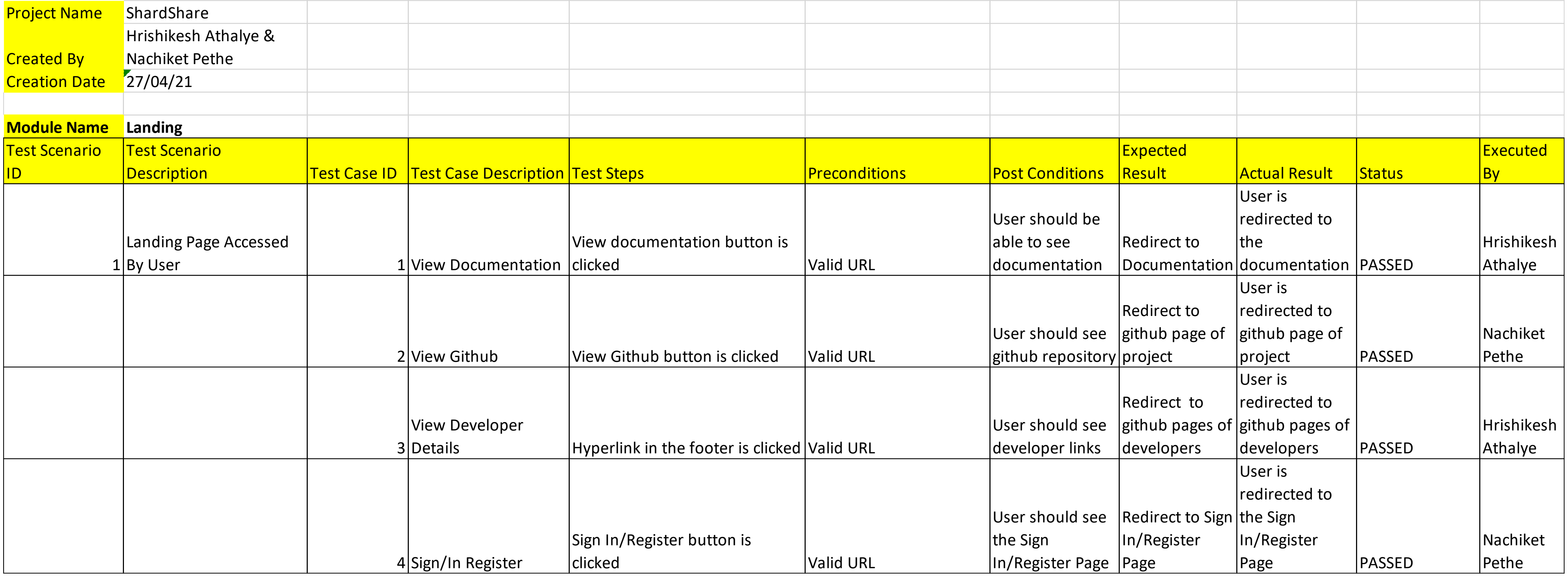
Participants

Date
2021-04-27T10:02:57.000Z

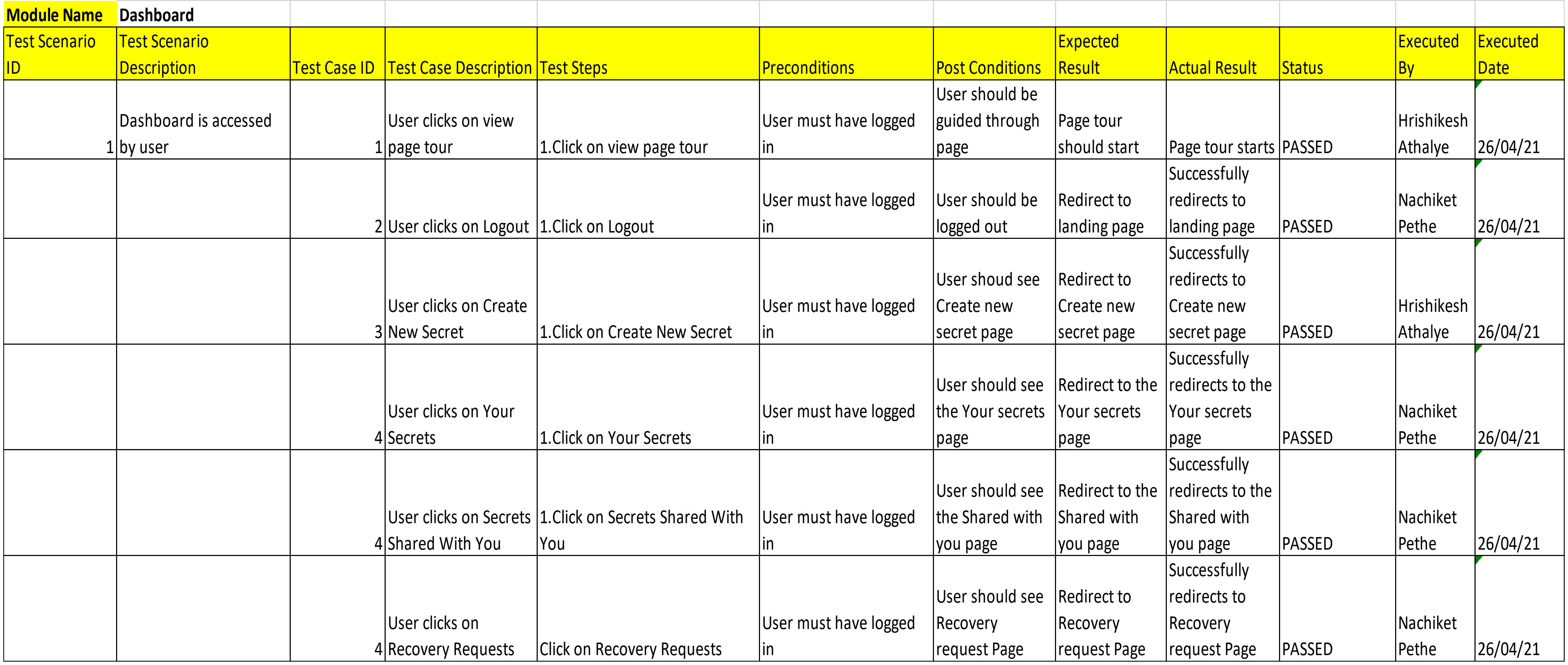
Shard *

9. Testing

9.1 Black Box Testing

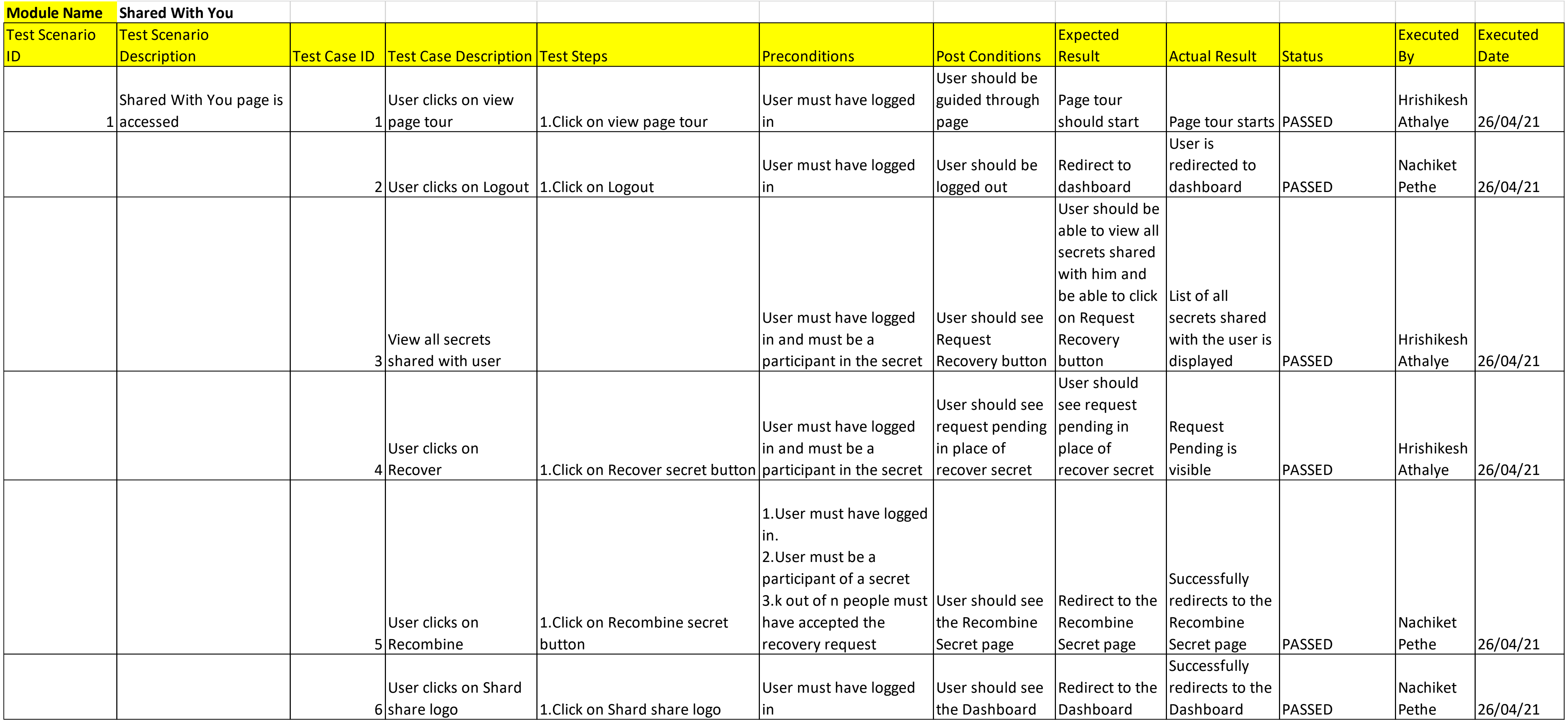


Project Name	ShardShare									
Created By	Hrishikesh Athalye & Nachiket Pethe									
Creation Date	27/04/21									
Module Name	**Landing**									
Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Preconditions	Post Conditions	Expected Result	Actual Result	Status	Executed By
1	Landing Page Accessed By User	1	View Documentation	View documentation button is clicked	Valid URL	User should be able to see documentation	Redirect to Documentation	User is redirected to the documentation	PASSED	Hrishikesh Athalye
		2	View Github	View Github button is clicked	Valid URL	User should see github repository	Redirect to github page of project	User is redirected to github page of project	PASSED	Nachiket Pethe
		3	View Developer Details	Hyperlink in the footer is clicked	Valid URL	User should see developer links	Redirect to github pages of developers	User is redirected to github pages of developers	PASSED	Hrishikesh Athalye
		4	Sign/In Register	Sign In/Register button is clicked	Valid URL	User should see the Sign In/Register Page	Redirect to Sign In/Register Page	User is redirected to the Sign In/Register Page	PASSED	Nachiket Pethe

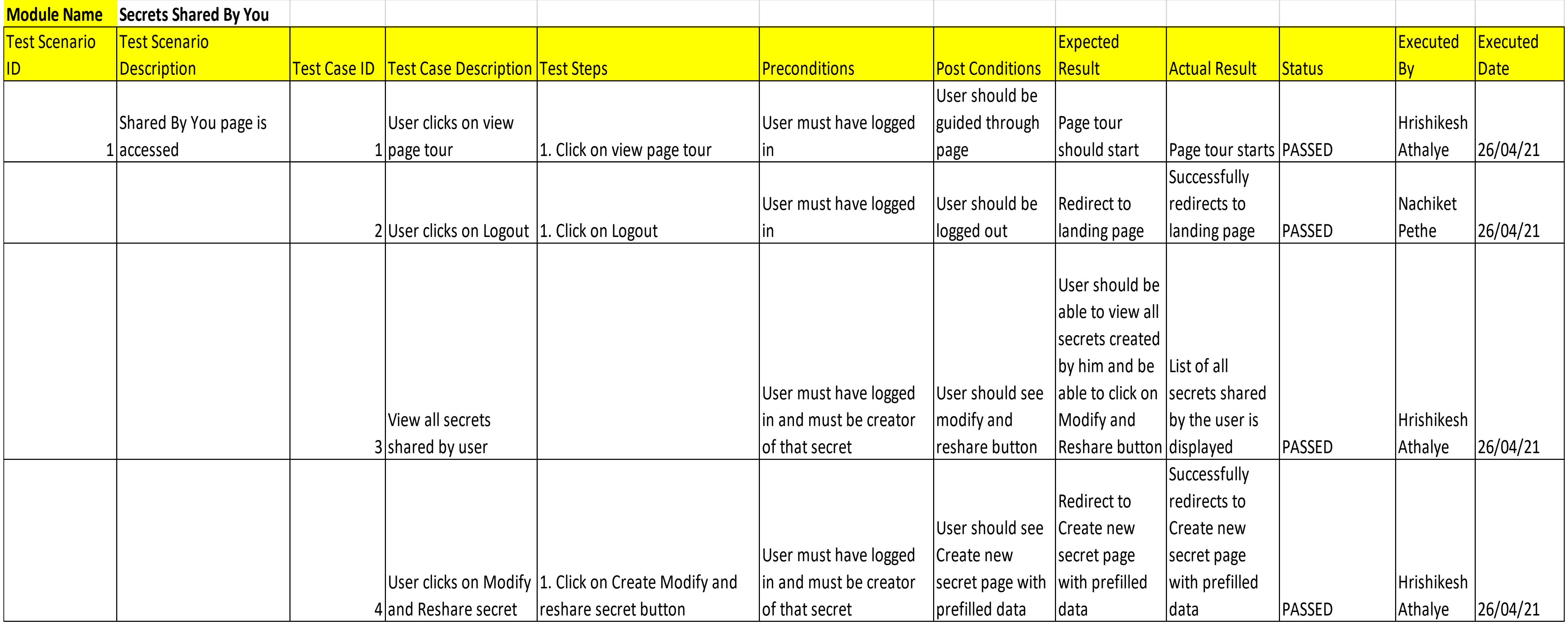


Module Name	**Dashboard**										
Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Preconditions	Post Conditions	Expected Result	Actual Result	Status	Executed By	Executed Date
1	Dashboard is accessed by user	1	User clicks on view page tour	1.Click on view page tour	User must have logged in	User should be guided through page	Page tour should start	Page tour starts	PASSED	Hrishikesh Athalye	26/04/21
		2	User clicks on Logout	1.Click on Logout	User must have logged in	User should be logged out	Redirect to landing page	Successfully redirects to landing page	PASSED	Nachiket Pethe	26/04/21
		3	User clicks on Create New Secret	1.Click on Create New Secret	User must have logged in	User shoud see Create new secret page	Redirect to Create new secret page	Successfully redirects to Create new secret page	PASSED	Hrishikesh Athalye	26/04/21
		4	User clicks on Your Secrets	1.Click on Your Secrets	User must have logged in	User should see the Your secrets page	Redirect to the Your secrets page	Successfully redirects to the Your secrets page	PASSED	Nachiket Pethe	26/04/21
		4	User clicks on Secrets Shared With You	1.Click on Secrets Shared With You	User must have logged in	User should see the Shared with you page	Redirect to the Shared with you page	Successfully redirects to the Shared with you page	PASSED	Nachiket Pethe	26/04/21
		4	User clicks on Recovery Requests	Click on Recovery Requests	User must have logged in	User should see Recovery request Page	Redirect to Recovery request Page	Successfully redirects to Recovery request Page	PASSED	Nachiket Pethe	26/04/21

Module Name	**Sign In/Register**											
Equivalence Classes	**Valid:** Valid email of the form username@domain.com and password having atleast 5 characters, one uppercase, one		**Invalid:** Incomplete email address or password that does not follow password rules									
Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Post Conditions	Expected Result	Actual Result	Status	Executed By	Executed Date
1	Sign In Page Accessed By User	1	Enter a valid email and valid password	1. Enter valid email 2. Enter valid password 3. Click on sign in	Valid URL Test Data	Email : email@a.com Password : A@1de	User should be able to see dashboard	User should be redirected to dashboard	User is redirected to the documentation	PASSED	Hrishikesh Athalye	26/04/21
		2	Enter a invalid email and valid password	1. Enter invalid email 2. Enter valid password 3. Click on sign in	Valid URL Test Data	Email : email@a Password : A@1de	User should get an error saying User Does Not Exist	Error message should be seen	Error message seen	PASSED	Nachiket Pethe	26/04/21
		3	Enter a valid email and invalid password	1. Enter valid email 2. Enter invalid password 3. Click on sign in	Valid URL Test Data	Email : email@a.com Password : abc	User should get an error saying Invalid Credentials	Error message should be seen	Error message seen	PASSED	Hrishikesh Athalye	26/04/21
		4	Login with Gmail	1. click on sign in with gmail button	User should have a gmail account		User should be able to see dashboard	User should be redirected to dashboard	Successfully redirected to dashboard	PASSED	Nachiket Pethe	26/04/21
2	Register Page Accessed By User	1	Register as a new user with all valid data	1. Click on Sign up button on sign in page 2. Enter First Name and Last Name 3. Enter valid Email Address 4. Enter valid and matching passwords with minimum 5 characters, one uppercase, one lowercase and one special character	Sign Up page should be visible	First Name : MyName Last Name : MySurname Email Address : first@last.com Password : A1b@e Repeat Password : A1b@e	User account should be created and user should see dashboard	User should be redirected to dashboard	Successfully redirected to dashboard	PASSED	Nachiket Pethe	26/04/21
		2	Register as a new user with invalid email	1. Click on Sign up button on sign in page 2. Enter First Name and Last Name 3. Enter Invalid Email Address 4. Enter matching passwords with minimum 5 characters, one uppercase, one lowercase and one special character	Sign Up page should be visible	First Name : MyName Last Name : MySurname Email Address : first@last Password : A1b@e Repeat Password : A1b@e	Invalid Email error should be displayed	Error message should be seen	Error message seen	PASSED	Hrishikesh Athalye	26/04/21
		3	Register as a new user with non matching passwords	1. Click on Sign up button on sign in page 2. Enter First Name and Last Name 3. Enter valid Email Address 4. Enter non matching passwords with minimum 5 characters, one uppercase, one lowercase and one special character	Sign Up page should be visible	First Name : MyName Last Name : MySurname Email Address : first@last.com Password : A1b@e Repeat Password : A2c#f	Passwords do not match error should be displayed	Error message should be seen	Error message seen	PASSED	Nachiket Pethe	26/04/21
		4	Register as a new user with matching but unsafe password	1. Click on Sign up button on sign in page 2. Enter First Name and Last Name 3. Enter Invalid Email Address 4. Enter matching passwords with minimum 5 characters, one uppercase, one lowercase and one special character	Sign Up page should be visible	First Name : MyName Last Name : MySurname Email Address : first@last.com Password : abcde Repeat Password : abcde	Password improvement error should be displayed	Error message should be seen	Error message seen	PASSED	Hrishikesh Athalye	26/04/21



Module Name	Shared With You										
Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Preconditions	Post Conditions	Expected Result	Actual Result	Status	Executed By	Executed Date
1	Shared With You page is accessed	1	User clicks on view page tour	1.Click on view page tour	User must have logged in	User should be guided through page	Page tour should start	Page tour starts	PASSED	Hrishikesh Athalye	26/04/21
		2	User clicks on Logout	1.Click on Logout	User must have logged in	User should be logged out	Redirect to dashboard	User is redirected to dashboard	PASSED	Nachiket Pethe	26/04/21
		3	View all secrets shared with user		User must have logged in and must be a participant in the secret	User should see Request Recovery button	User should be able to view all secrets shared with him and be able to click on Request Recovery button	List of all secrets shared with the user is displayed	PASSED	Hrishikesh Athalye	26/04/21
		4	User clicks on Recover	1.Click on Recover secret button	User must have logged in and must be a participant in the secret	User should see request pending in place of recover secret	User should see request pending in place of recover secret	Request Pending is visible	PASSED	Hrishikesh Athalye	26/04/21
		5	User clicks on Recombine	1.Click on Recombine secret button	1.User must have logged in. 2.User must be a participant of a secret 3.k out of n people must have accepted the recovery request	User should see the Recombine Secret page	Redirect to the Recombine Secret page	Successfully redirects to the Recombine Secret page	PASSED	Nachiket Pethe	26/04/21
		6	User clicks on Shard share logo	1.Click on Shard share logo	User must have logged in	User should see the Dashboard	Redirect to the Dashboard	Successfully redirects to the Dashboard	PASSED	Nachiket Pethe	26/04/21

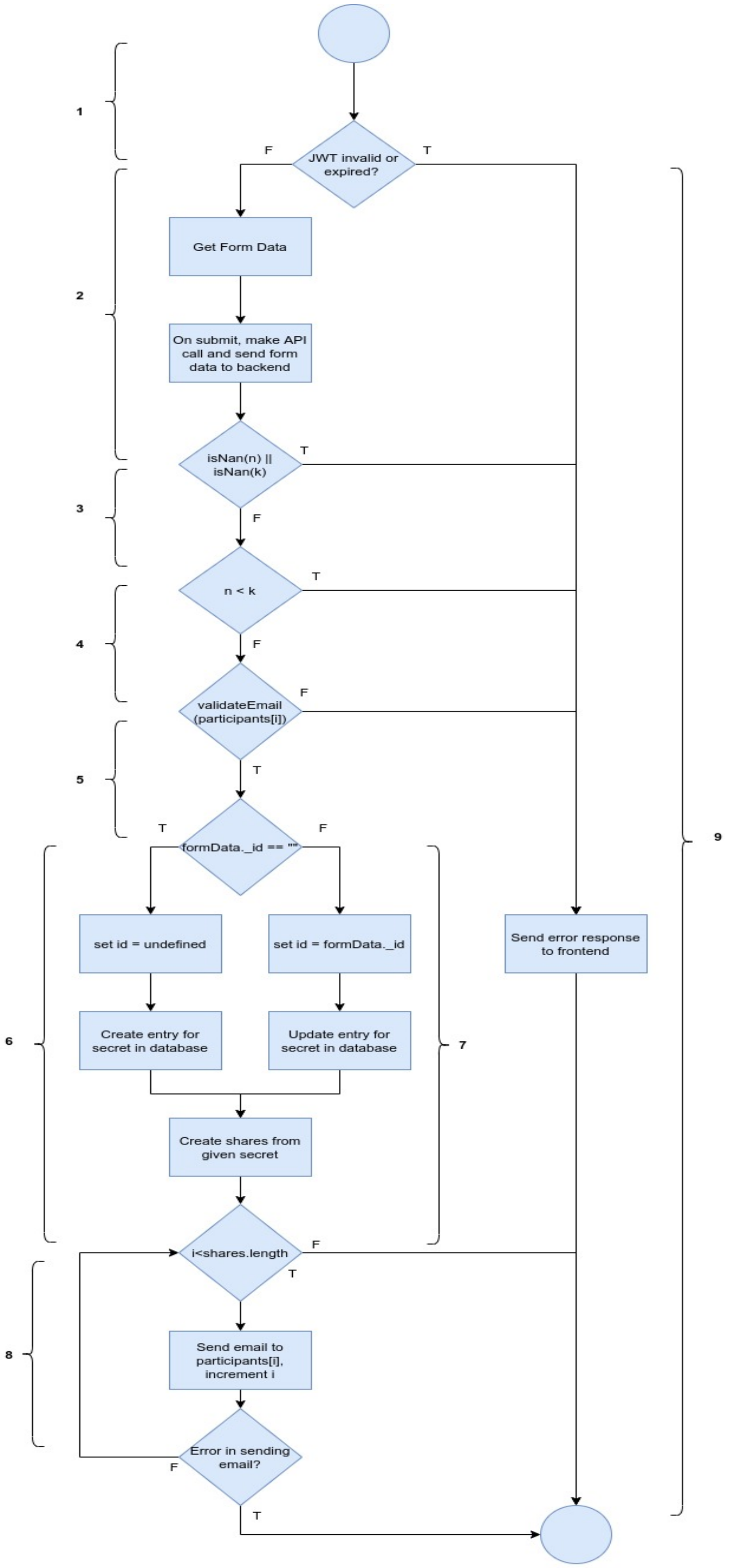


Module Name	Secrets Shared By You										
Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Preconditions	Post Conditions	Expected Result	Actual Result	Status	Executed By	Executed Date
1	Shared By You page is accessed	1	User clicks on view page tour	1. Click on view page tour	User must have logged in	User should be guided through page	Page tour should start	Page tour starts	PASSED	Hrishikesh Athalye	26/04/21
		2	User clicks on Logout	1. Click on Logout	User must have logged in	User should be logged out	Redirect to landing page	Successfully redirects to landing page	PASSED	Nachiket Pethe	26/04/21
		3	View all secrets shared by user		User must have logged in and must be creator of that secret	User should see modify and reshare button	User should be able to view all secrets created by him and be able to click on Modify and Reshare button	List of all secrets shared by the user is displayed	PASSED	Hrishikesh Athalye	26/04/21
		4	User clicks on Modify and Reshare secret	1. Click on Create Modify and reshare secret button	User must have logged in and must be creator of that secret	User should see Create new secret page with prefilled data	Redirect to Create new secret page with prefilled data	Successfully redirects to Create new secret page with prefilled data	PASSED	Hrishikesh Athalye	26/04/21

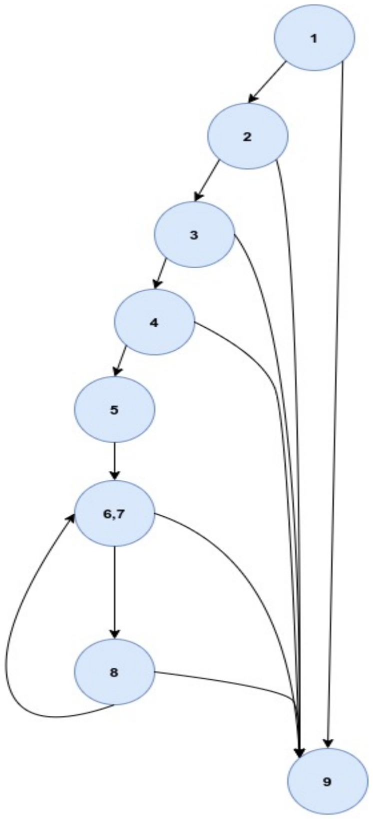
Module Name	**Create Secret**											
Equivalence Classes	**Valid**: Valid email of the form username@domain.com And n>0, k>0 And n and k not equal to 1 And n>=k			**Invalid:** n<0 or k <0 n=1 and k=1 Or n<k Or Atleast one invalid email								
Test Scenario ID	Test Scenario Description	Test Case ID		Test Steps	Test Data	Preconditions	Post Conditions	Expected Result	Actual Result	Status	Executed By	Executed Date
1	Create secret page is accessed by user	1	User clicks on view page tour	1.click on view page tour		User must be logged in	User should be guided through page	Page tour should start	Page tour starts	PASSED	Hrishikesh Athalye	26/04/21
		2	User clicks on Logout	1.click on Logout		User must be logged in	User should be logged out	Redirect to landing	Successfully redirected to landing	PASSED	Nachiket Pethe	26/04/21
		3	Share Secret with n=1, k=1	1. Enter secret name 2. Enter Secret 3. Enter n=1, k=1 4. Add participants	Secret Name: Secret1 Secret: abc N: 1 K: 1 Participants: hathalye7@gmail.com	User must be logged in	User should see an error as n=1 and k=1 is not permitted	An error message is expected	Error message seen	PASSED	Hrishikesh Athalye	26/04/21
		4	Share Secret with n<k	1. Enter secret name 2. Enter Secret 3. Enter n,k values such that n<k 4. Add participants	Secret Name: Secret1 Secret: abc N: 2 K: 3 Participants: hathalye7@gmail.com hathalye7@hotmail.com	User must be logged in	User should see an error message as n<k is not permitted	An error message is expected	Error message seen	PASSED	Hrishikesh Athalye	26/04/21
		5	Share secret which includes one invalid email	1. Enter secret name 2. Enter Secret 3. Enter n,k values 4. Add participants such that atleast one email is invalid	Secret Name: Secret1 Secret: abc N: 2 K: 2 Participants: hathalye7@gmail hathalye7@hotmail.com	User must be logged in	User should see an error message to check the emails	An error message is expected	Error message seen	PASSED	Nachiket Pethe	26/04/21
		6	Share secret with all valid fields	1. Enter secret name 2. Enter Secret 3. Enter n,k values 4. Add participants	Secret Name: Secret1 Secret: abc N: 2 K: 2 Participants: hathalye7@gmail.com hathalye7@hotmail.com	User must be logged in	User should see a success message and emails should be sent to all participants	Success message expected and email should be received	Emails were received successfully and success message was seen	PASSED	Nachiket Pethe	26/04/21

9.2 White Box Testing

Module Name	Share Secret


1
2
3
4
5
6
7
8
9
JWT invalid or expired?
F
T
Get Form Data
On submit, make API call and send form data to backend
isNan(n) \|\| isNan(k)
T
F
n < k
T
F
validateEmail (participants[i])
F
T
formData._id == ""
T
F
set id = undefined
set id = formData._id
Create entry for secret in database
Update entry for secret in database
Send error response to frontend
Create shares from given secret
i<shares.length
F
T
Send email to participants[i], increment i
Error in sending email?
F
T

Flowchart



Flow Graph

Number of predicate nodes = 7

Cyclomatic Complexity = 8

Basis Paths:

1. 1-9
2. 1-2-9
3. 1-2-3-9
4. 1-2-3-4-9
5. 1-2-3-4-5-6-9
6. 1-2-3-4-5-7-9
7. 1-2-3-4-5-6-8-9
8. 1-2-3-4-5-7-8-9

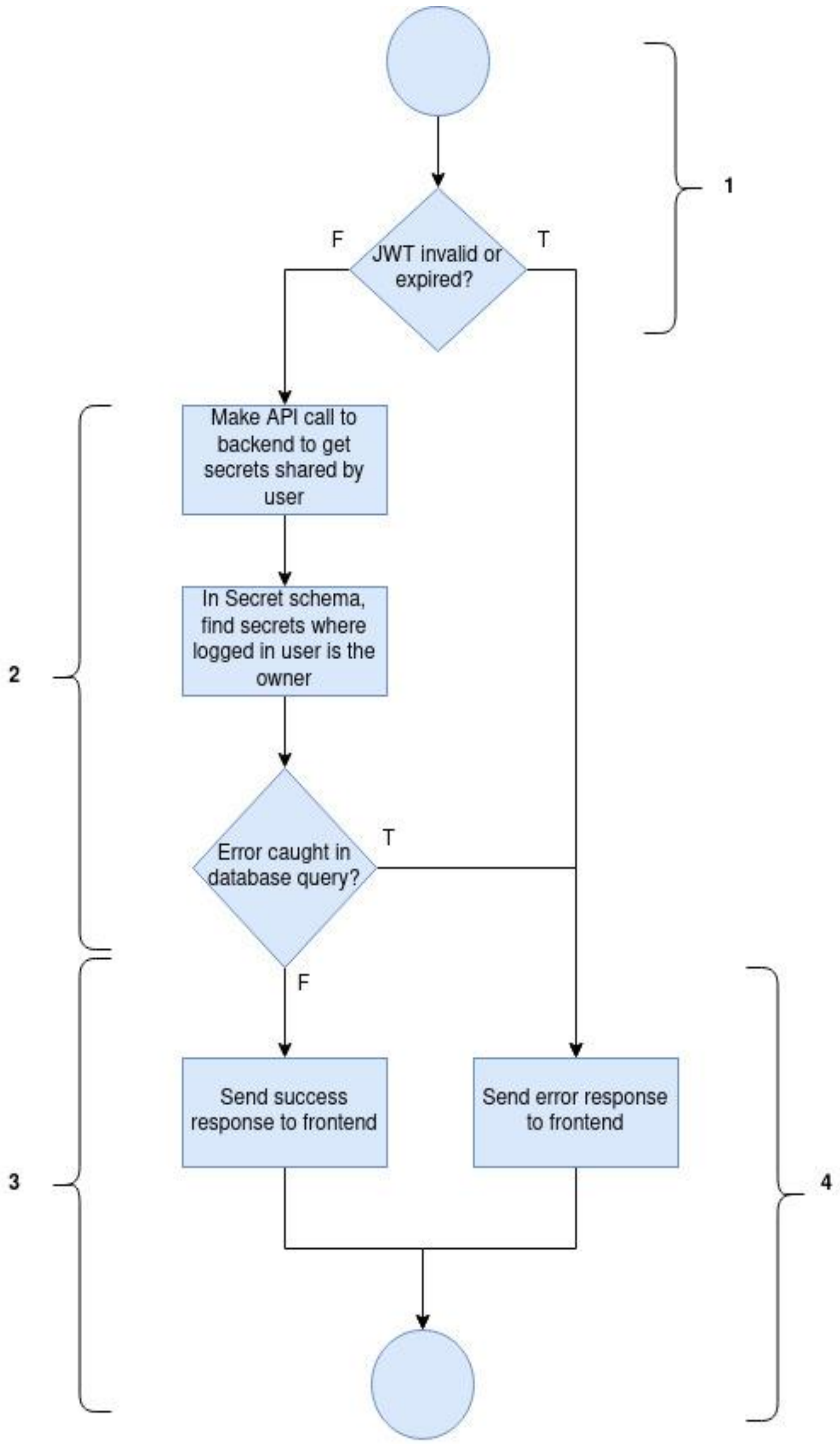
Algorithm:

1. Start
2. If isNotValid(JWT) then end
3. Get Form Data
4. If isSubmitted(formData) then make API call
5. if(isNan(n) || isNan(k)) then end
6. else if(n < k) then end
7. Else if(validateEmail(participants[i]) == false) then end
8. if formData._id == "", set id as undefined and create entry in database
9. Else, id=formData._id and update entry in database
10. Create shares from secret and send to all participants
11. If any error catch and send error response
12. Else, send success response
13. End

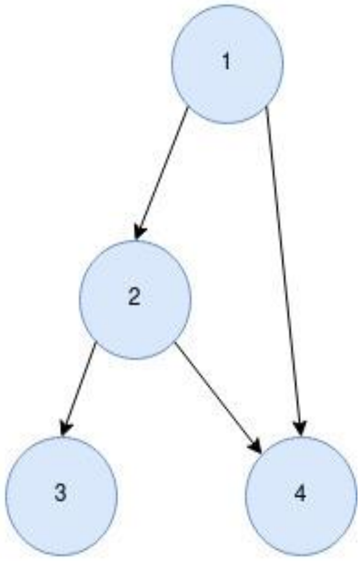
Module Name	Secrets Shared By User

S

Number of predicate nodes = 2
Cyclomatic Complexity = 3
Basis Paths:
1. 1-4
2. 1-2-3
3. 1-2-4


JWT invalid or expired?
F
T
1
Make API call to backend to get secrets shared by user
In Secret schema, find secrets where logged in user is the owner
Error caught in database query?
T
F
2
Send success response to frontend
Send error response to frontend
3
4

Flowchart



Flow Graph

Algorithm:
1. Start
2. If isNotValid(JWT) then end
3. Make API call to backend to get secrets shared by user
4. For each secret in secret database, find secrets where currently logged in user is the owner
5. If any error, handle it and send error response
6. Else, send success response
7. End

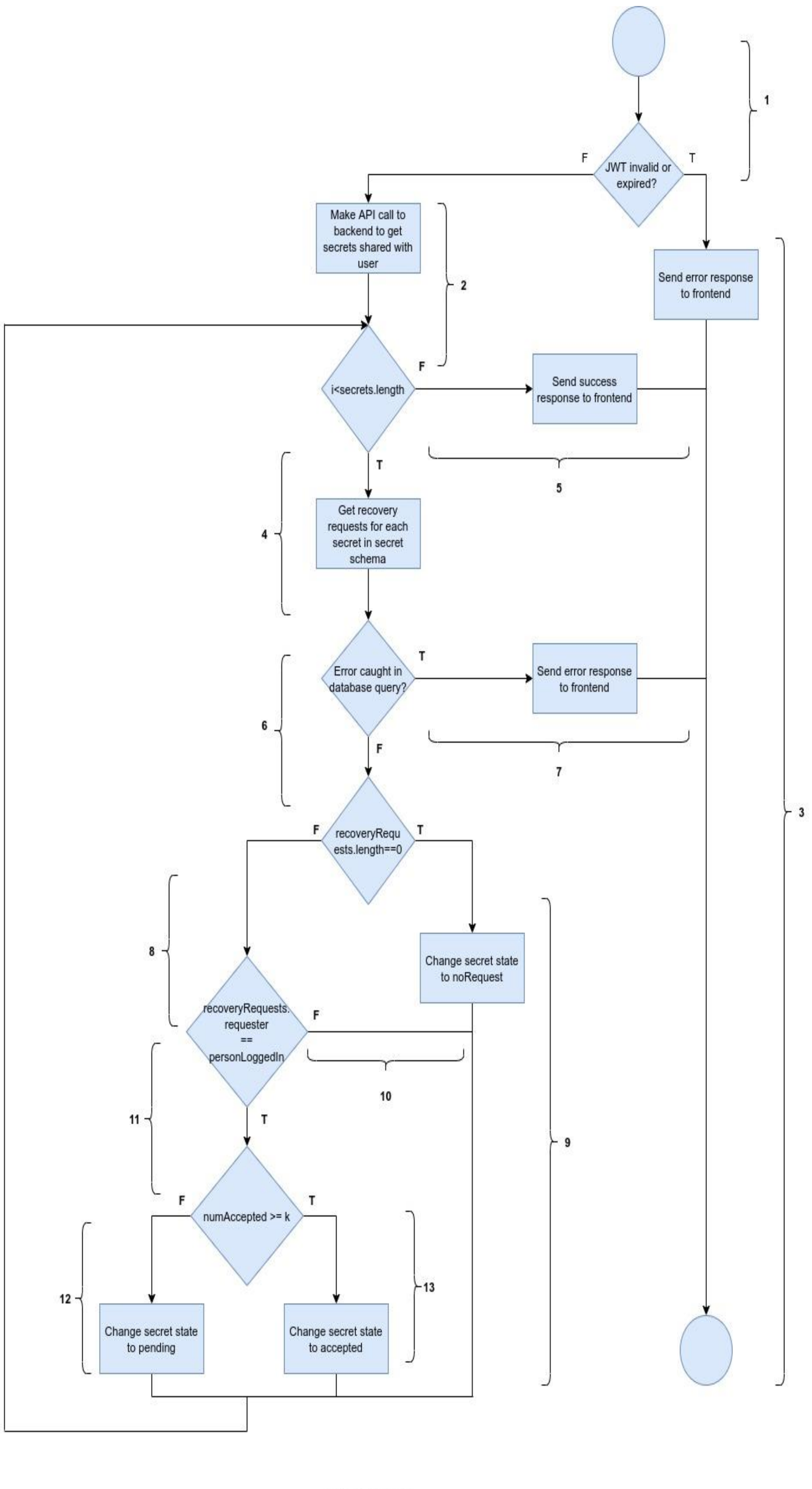
Module Name	Secrets Shared With User

Number of predicate nodes = 6

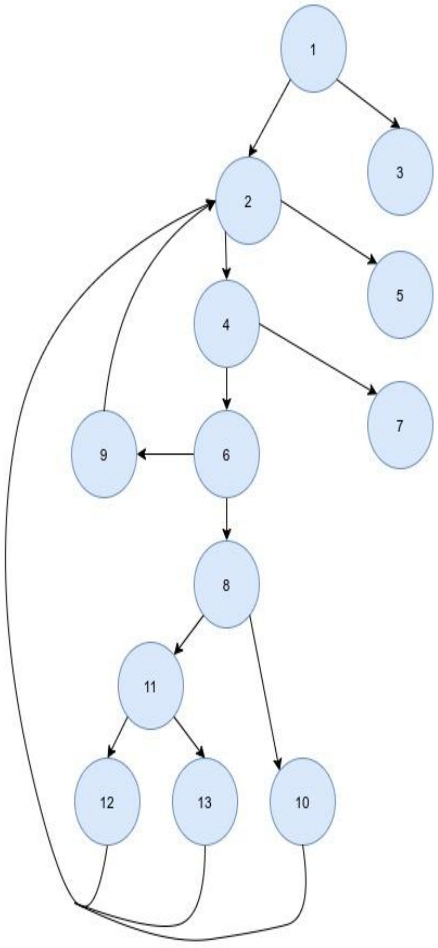
Cyclomatic Complexity = 7

Basis Paths:

1. 1-3
2. 1-2-5
3. 1-2-4-7
4. 1-2-4-6-9
5. 1-2-4-6-8-10
6. 1-2-4-6-8-11-12
7. 1-2-4-6-8-11-13


JWT invalid or expired?
F
T
1
Make API call to backend to get secrets shared with user
Send error response to frontend
2
i<secrets.length
F
T
Send success response to frontend
5
Get recovery requests for each secret in secret schema
4
Error caught in database query?
T
F
Send error response to frontend
6
7
3
recoveryRequests.length==0
F
T
Change secret state to noRequest
8
recoveryRequests.requester == personLoggedIn
F
T
10
11
9
numAccepted >= k
F
T
Change secret state to pending
Change secret state to accepted
12
13

Flowchart



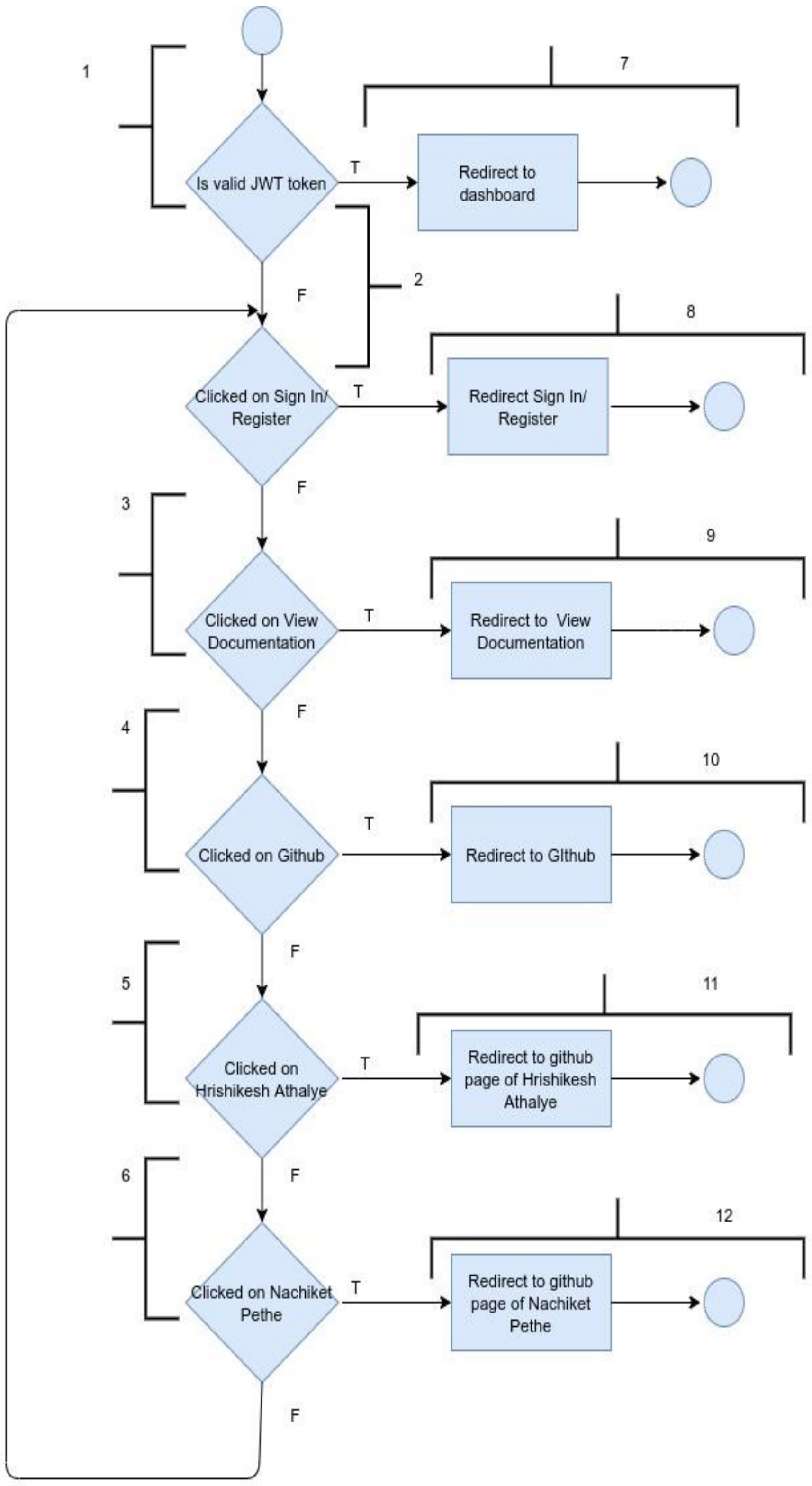
Flow Graph

Algorithm:

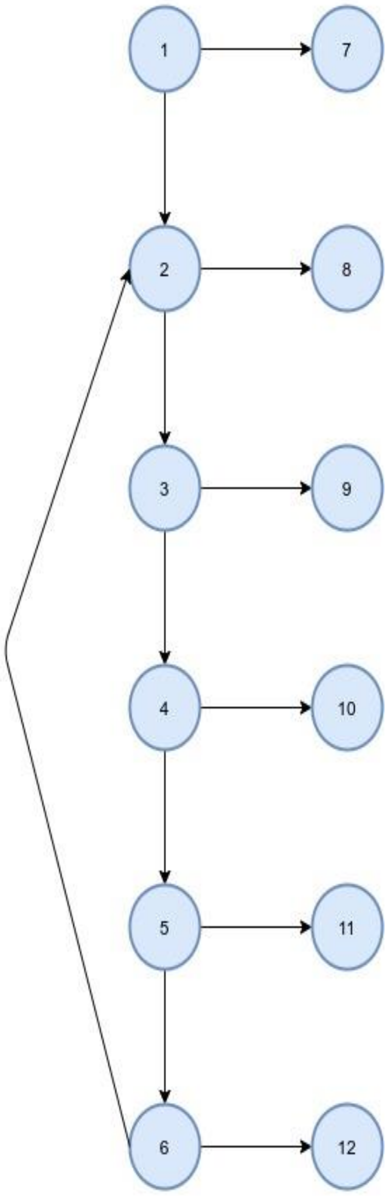
1. Start
2. If isNotValid(JWT) then end
3. Make API call to backend
4. If i<secrets.length send success response to frontend
5. Get recovery request for each secret in secret schema
6. If any error catch and send error response
7. If recoveryRequests.length == 0 then change secret state to noRequest
8. If requester in the recovery request is not the currently logged in person go to step 10
9. Else, if numAccepted>=k change secret state to accepted else change secret state to pending
10. Increment i and go to step 4
11. End

Module Name	Landing

Number of predicate nodes = 6
Cyclomatic Complexity = 7
Basis Paths:
1. 1-7
2. 1-2-8
3. 1-2-3-9
4. 1-2-3-4-10
5. 1-2-3-4-5-11
6. 1-2-3-4-5-6-12
7. 1-2-3-4-5-6-2


1
7
Is valid JWT token
T
Redirect to dashboard
F
2
8
Clicked on Sign In/ Register
T
Redirect Sign In/ Register
F
3
9
Clicked on View Documentation
T
Redirect to View Documentation
F
4
10
Clicked on Github
T
Redirect to GIthub
F
5
11
Clicked on Hrishikesh Athalye
T
Redirect to github page of Hrishikesh Athalye
F
6
12
Clicked on Nachiket Pethe
T
Redirect to github page of Nachiket Pethe
F

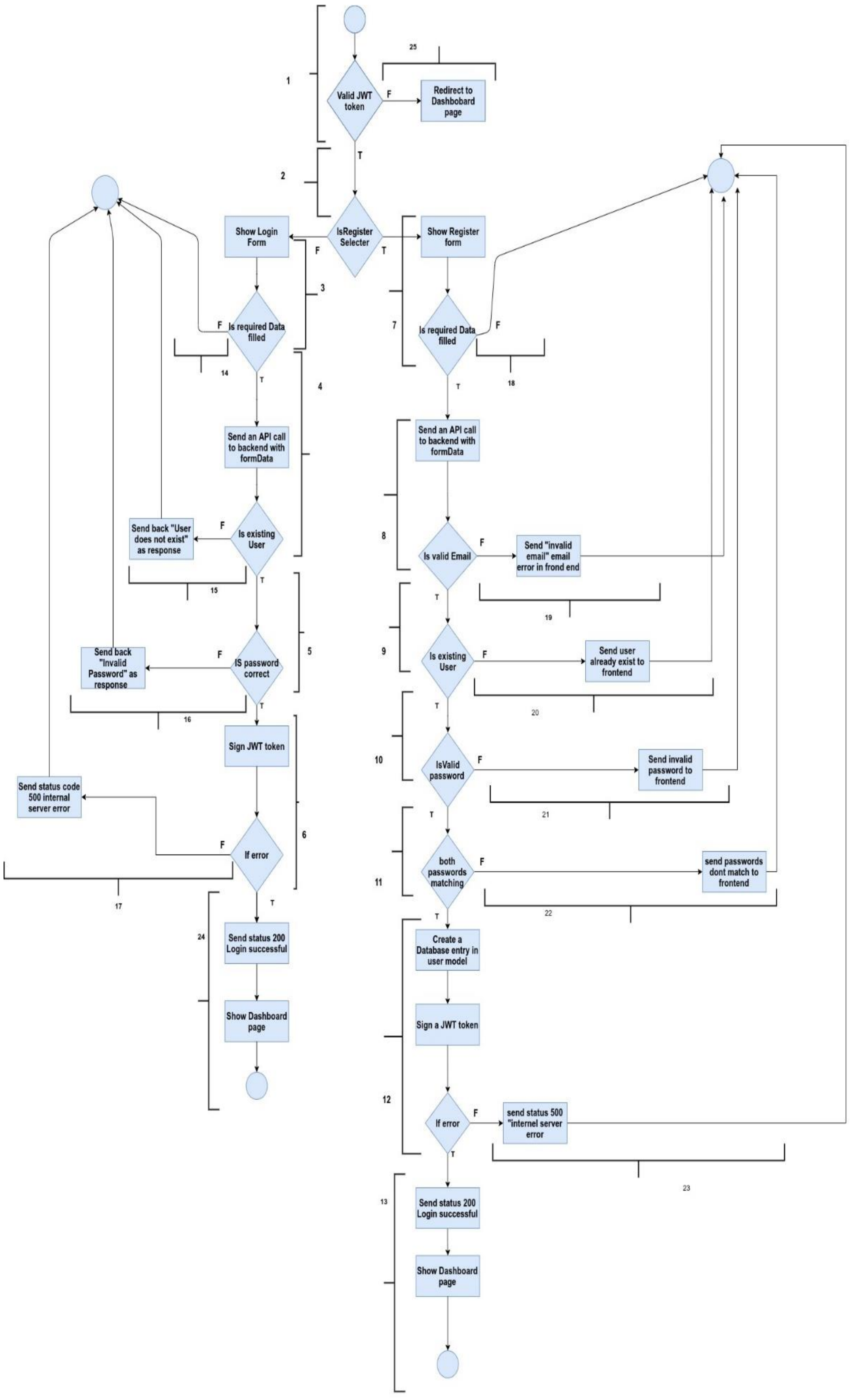
Flowchart



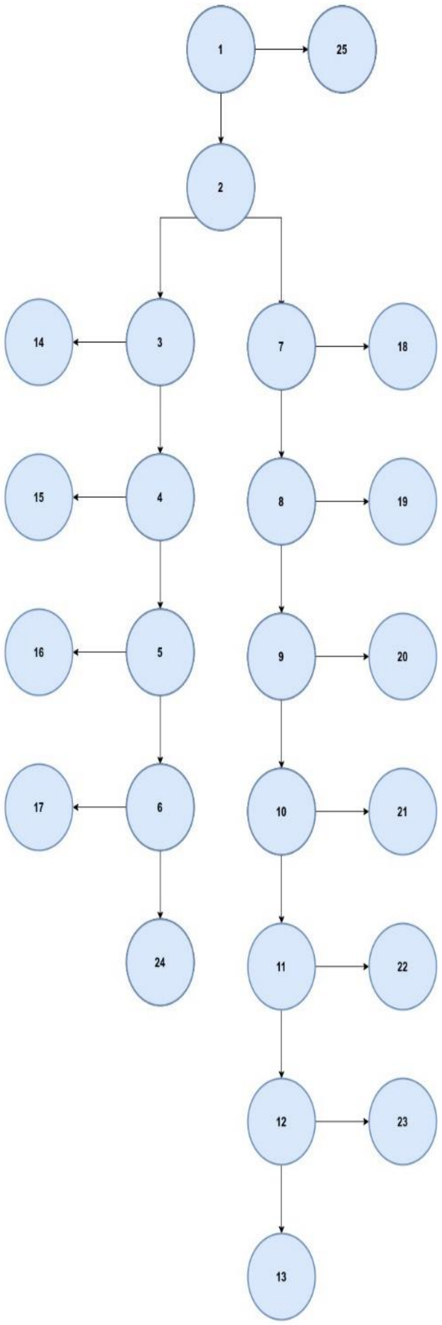
Flow Graph

Algorithm:
1. Start
2. If invalid JWT token:
Redirect to the landing page
Else:
Switch clicks:
3. Case : click on page tour
Display page tour
4. Case: click on view documentation
Redirect to documentation of project
5. Case: click on view github
Redirect to github page of the project
6. Case : click on Nachiket Pethe
Redirect to the github page of nachiket pethe
7. Case: click on Hrishikesh Athalye
Redirect to the github page of hrishkesh athalye.
8. Case: click on Sign in / Register
Redirect to the Sign in/ Register Page
9. End

Module Name	Login


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
Valid JWT token
F
Redirect to Dashbobard page
T
IsRegister Selecter
F
T
Show Login Form
Show Register form
Is required Data filled
F
T
Send an API call to backend with formData
Is existing User
F
T
Send back "User does not exist" as response
IS password correct
Send back "Invalid Password" as response
Sign JWT token
If error
Send status code 500 internal server error
Send status 200 Login successful
Show Dashboard page
Is valid Email
Send "invalid email" email error in frond end
Send user already exist to frontend
IsValid password
Send invalid password to frontend
both passwords matching
send passwords dont match to frontend
Create a Database entry in user model
Sign a JWT token
send status 500 "internel server error
Send status 200 Login successful
Show Dashboard page

Flowchart



Flow Graph

Number of predicate nodes = 12

Cyclomatic Complexity = 13

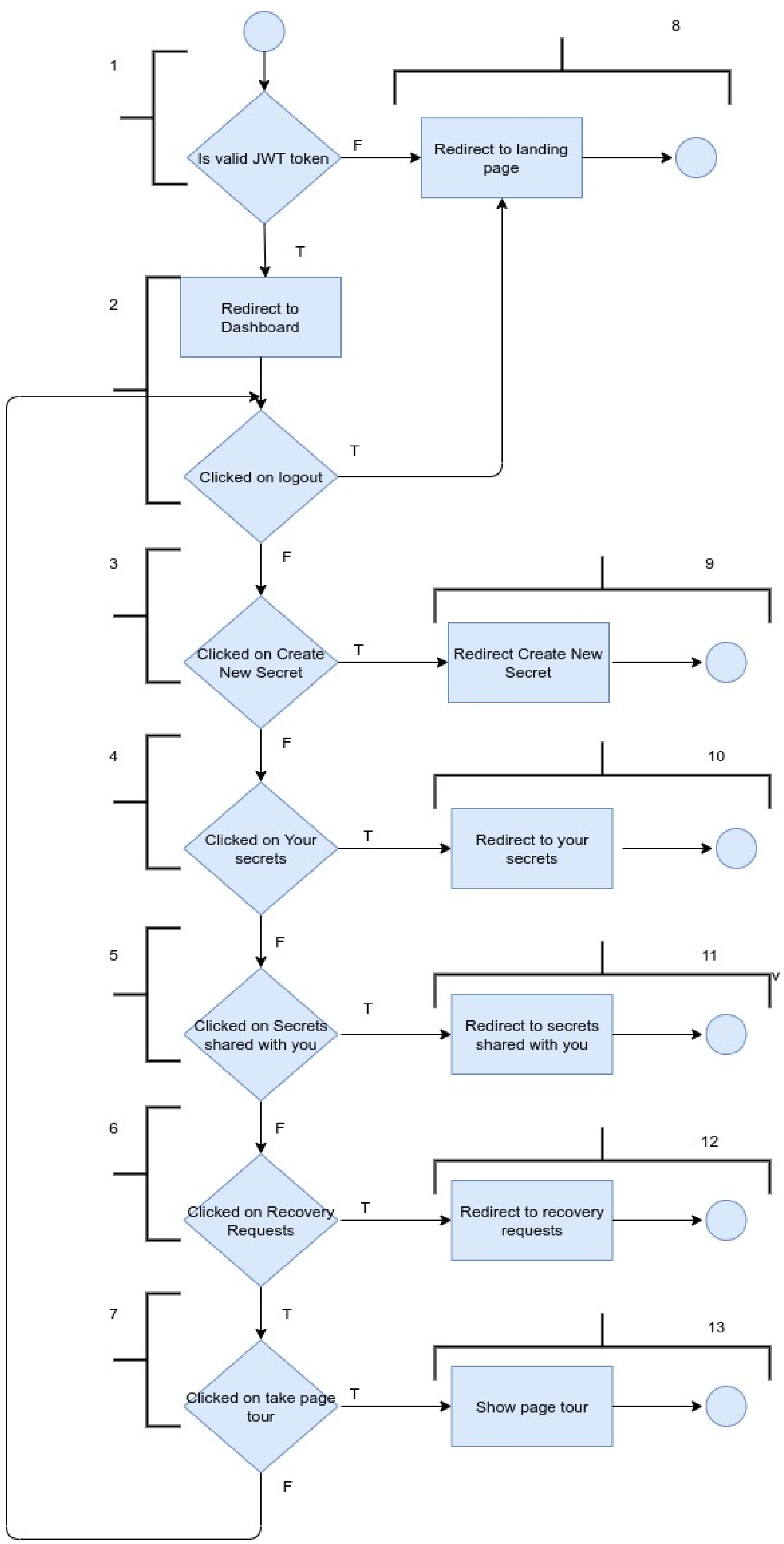
Basis Paths:

1. 1-25
2. 1-2-3-14
4. 1-2-7-18
5. 1-2-3-4-15
6. 1-2-7-8-19
7. 1-2-3-4-5-16
8. 1-2-7-8-9-20
9. 1-2-3-4-5-6-17
10. 1-2-7-8-9-10-21
11. 1-2-3-4-5-6-24
12. 1-2-7-8-9-10-11-22
13. 1-2-7-8-9-10-11-12-23

Algorithm:

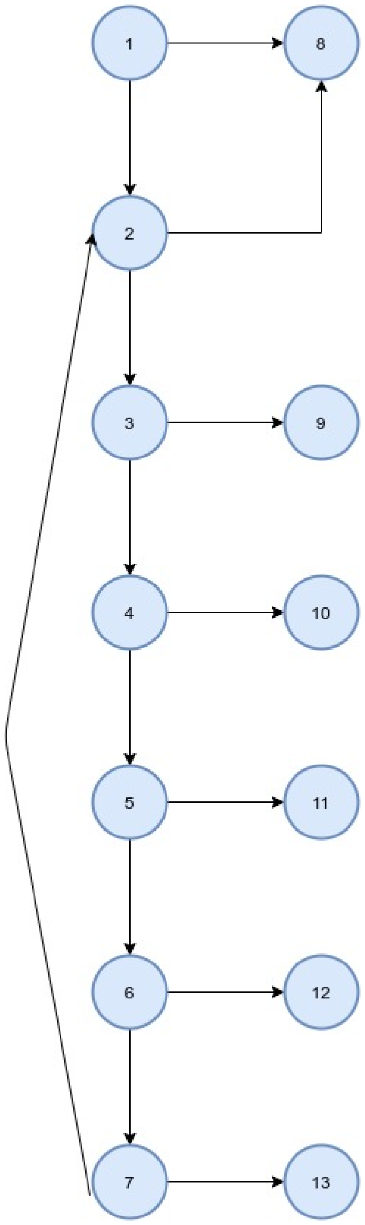
```
Login / Register page (frontend)
Label1: If valid JWT token:
Redirect to dashboard page
Else:
If isSignIn set:
Show sign up form
If sign up button is clicked:
If required fields not set:
Show error
Else:
Send API request to backend
Show response message
Goto label1
Else:
Show sign in form
If sign in button is clicked:
If required fields not set:
Show error
Else:
Send API request to backend
Show response message
 Goto label1
Login (backend)
API request from front end is received
If invalid email:
Send invalid email error in response
Find a record in users collection having exact same email and password
If record is found:
Set JWT token
Send response as success
Else:
Send response as invalid credentials.


Register Backend
API request from frontend is received
If invalid email
Send response as invalid email
If password is not valid
Send response as invalid password
If password != confirm password
Send response as passwords don't match.
Create a database entry in users collection.
```

Module Name	Dashboard



1
Is valid JWT token
F
Redirect to landing page
8
T
2
Redirect to Dashboard
Clicked on logout
T
F
3
Clicked on Create New Secret
T
Redirect Create New Secret
9
F
4
Clicked on Your secrets
T
Redirect to your secrets
10
F
5
Clicked on Secrets shared with you
T
Redirect to secrets shared with you
11
F
6
Clicked on Recovery Requests
T
Redirect to recovery requests
12
T
7
Clicked on take page tour
T
Show page tour
13
F

Flowchart



Flow Graph

Number of predicate nodes = 7
Cyclomatic Complexity = 8
Basis Paths:
1. 1-8
2. 1-2-8
3. 1-2-3-9
4. 1-2-3-4-10
5. 1-2-3-4-5-11
6. 1-2-3-4-5-6-12
7. 1-2-3-4-5-6-7-13
8. 1-2-3-4-5-6-7-13-2

Algorithm:
1. Start
2. If invalid JWT token:
Redirect to the landing page
3. Else:
4. Switch clicks:
5. Case : click on page tour
Display page tour
6. Case: click on create new secret
Redirect to the create new secret page
7. Case: click on your secret
Redirect to your secret
8. Case : click on secrets shared with you
Redirect to Secret shared with you 9. page.
10. Case: click on Recovery requests
Redirect to recovery request page.
11. Case : click on logout
Remove the JWT token and redirect to the landing page
12. End

10. Links

1. Github Repository - https://github.com/hrishikeshathalye/ShardShare
2. Deployed Application - https://shardshare-frontend.herokuapp.com/

11. References

3. Shamir’s Secret Sharing Scheme Paper - https://dl.acm.org/doi/10.1145/359168.359176
4. Numeric walkthrough of the scheme - https://medium.com/@apogiatzis/shamirs-secret-sharing-a-numeric-example-walkthrough-a59b288c34c4
5. Miscellaneous information - https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing
6. IEEE SRS Format - https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc
7. React,js Documentation - https://reactjs.org/
8. Node,js Documentation - https://nodejs.org/en/docs/
9. Secrets,js - https://www.npmjs.com/package/secrets.js-grempe
10. Mongoose Documentation - https://mongoosejs.com/docs/
11. PWA Reference - https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
12. Material UI Documentation - https://material-ui.com/