**MEDICHAIN: E-HEALTH DATA PLATFORM**

PROJECT REPORT

Submitted by

**ABEL T VARGHESE(PTA19CS001)**

**AJU SEBASTIAN(PTA19CS004)**

**ALAN FRANCIS JOSE (PTA19CS006)**

**SONA CHACKO (PTA19CS048)**

**TINSU MARIYAM THOMAS(PTA19CS052)**

To

*The APJ Abdul Kalam Technological University*

*in partial fulfillment of the requirements for the award of the*

*Degree of Bachelor of Technology in Computer Science and Engineering*

****

**Department Of Computer Science & Engineering**

College of Engineering Kallooppara

Pathanamthitta

June 2023

**DECLARATION**

We undersigned hereby declare that the project report **MEDICHAIN**, submitted for partial fulfilment of the requirements for the awardof degree of Bachelor of Technology of the APJ Abdul Kalam Technological work done by us under supervision of University, Kerala is a bonafide work done by us under the supervision of **Mrs. Anitha Jose.** This submission represents our ideas in our own words and ideas or words of others have been included where we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the university and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other university.

Place: Kallooppara ABEL T VARGHESE

Date: 15/06 /2023 AJU SEBASTIAN

ALAN FRANCIS JOSE

SONA CHACKO

TINSU MARIYAM THOMAS

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**COLLEGE OF ENGINEERING KALLOOPPARA**

**PATHANAMTHITTA - 689603**



**CERTIFICATE**

This is to certify that the Project Design Report entitled the **MEDICHAIN** submitted by **Abel T Varghese**(PTA19CS001), **Aju Sebastian** (PTA19CS004), **Alan Francis Jose (**PTA19CS006), **Sona Chacko** (PTA19CS048), **Tinsu Mariyam Thomas** (PTA19CS052) in partial fulfilment with the requirements forthe award of the **Degree of Bachelor of Technology in Computer Science and Engineering** of APJ Abdul Kalam Technological University is a bonafide work carried out by them under the guidance and supervision of Mrs. Anitha Jose. This report in any form has not been submitted to any other University or Institute for any purpose.

**Coordinator**  **Guide** **Head of the Department**

**Ms. Leeba Merin Sam**  **Mrs. Anitha Jose Dr. Renu George**

Assistant Professor Assistant Professor Assistant Professor

Department of Department of Department of

Computer Science & Computer Science & Computer Science&

Engineering Engineering Engineering

**ACKNOWLEDGEMENT**

We hereby take the opportunity to thank **Dr. Nisha Kuruvilla**, The Principal, College of Engineering Kallooppara, **Dr. Renu George**, Head of the Department of Computer Science and Engineering for providing necessary facilities. We also express our gratitude to our guide **Mrs. Anitha Jose** and our project coordinator **Ms. Leeba Merin Sam**, Assistant Professor of the Department of Computer Science and Engineering, College of Engineering Kallooppara, for their valuable points and guidance throughout the course of our project. Last but not leaving the one, we would like to thank our parents for their motivation and our friends who gave us valuable contributions regarding our topic and the encouragement throughout the preparation of the mini project. Above all we thank to God for his abundant blessings without which this would not be possible.

ABEL T VARGHESE

AJU SEBASTIAN

ALAN FRANCIS JOSE

SONA CHACKO

TINSU MARIYAM THOMAS

**ABSTRACT**

This project tackle the problem of sharing eHealth data across different jurisdictions. As a general rule, and due to the sensitive nature of the information, different national regulations impose severe limits on what can be exchanged, even in case of emergencies. Furthermore, different systems in different jurisdictions do not communicate. **Medichain** is proposed as a scheme that allows eHealth data to be securely exchanged, with the data subject always in the position of mediation. Medichain combine several technologies namely Blockchain, Django and User-Managed Access to achieve our aim.

**CONTENTS**

Contents Page No.

**ACKNOWLEDGEMENT**

**ABSTRACT**

**LIST OF FIGURES**

**LIST OF ABBREVATIONS**

# **INTRODUCTION**……………………………………………………...…….1

1.1 General Background……………………………………………………...1

1.2 Problem Statement………………..…………………………………........1

1. 3 Objective…………………………………………………………………2

1.4 Purpose and scope…………………………………………………………3

# **LITERATURE REVIEW**………………………………….…...……………4

**Chapter 3. SYSTEM ANALYSIS**………………………………………………………..6

3.1 Expected System Requirements ………………………………………….6

3.1.1 Web3.py……………………………………………………………6

3.1.2 Django……………………………………………………………..7

3.1.3 Python……………………………………………………………...8

3.2 Software Requirements…………………………………………………...9

3.2.1 SQLite………………………………………………………………9

3.2.2 Mailtrap…………………………………………………………...10

3.2.3 Ganache CLI………………………………………………………12

3.2.4 Tailwind CSS……………………………………………………...13

3.2.5 HTML……………………………………………………………..14

**Chapter 4. METHODOLOGY**………………………………………….……………… 15

4.1 Proposed System……………………………….……………………….15

**Chapter 5. SYSTEM DESIGN**………………………………………………………….17

5.1 Architecture…………………………………………………………….17

5.2 Diagrams……………………………………………..………………...18

5.2.1 Use Case Diagrams………………………………………………18

5.2.2 Data Flow Diagram………………………………………………19

5.2.2.1 Level 0……………………………………………………19

5.2.2.2 Level 1……………………………………………………20

5.2.2.3 Level 2……………………………………………………21

5.2.3 E-R Diagram……………………………………………………..22

**Chapter 6. SYSTEM IMPLEMENTATION**……………...…………………………...23

6.1 Smart Contract for the System………………………………………...23

6.2 Code for Token Creation in Server Side………………………………25

6.3 Data Models…………………………………………………………...26

6.3.1 Request Model…………………………………………………..27

6.3.2 Token Model…………………………………………………….27

6.3.3 User Model………………………………………………………28

6.4 Token verify and download data………………………………………...28

**Chapter 7. TESTING**………………………………...………………………………….31

7.1 Unit Testing for Solidity Code…………………………………………31

7.2 Mail Testing……………………………………………………………33

**Chapter 8. RESULTS**………………………………...…………………………………34

**Chapter 9. CONCLUSION**………………………………...……………………………40

**REFERENCES**………………………………...……………………………………………41

**LIST OF FIGURES AND TABLES**

No. Title Page No.

4.1 Proposed System ………………………………………………………………………15

5.1 Architecture………….…………………………………...…….……………………...17

5.2 Use Case Diagram…….……..………………………………………………………...18

5.3 DFD Level 0……………………...……………………………………………………20

5.4 DFD Level 1…………………………………………………………………………...20

5.5 DFD Level 2…………………………………………………………………………...21

5.6 E-R Diagram ……………………………………………………………………….....32

8.1 Home Page …………,,,……………………………………………………………….35

8.2 Sign-up Page………………………………………..…………………………………36

8.3 Login Page…………………………………………………………………………….36

8.4 Send Request Page…………………………………………………………….............37

8.5 View Request Page………………………..…………………………………………..37

8.6 Create Token………………………………………………………………………….38

8.7 Send Mail………………………………………………………………………………38

8,8 Verify Token and Download data……………………………………………………..39

**ABBREVATIONS**

DS Data Subject

RO Resource owner

RqP Requesting party that wants to access the health records.

AS Authorization server

LAS Local Authorization Server

RAS Remote Authorization Server

RS Resource Servers

LRS Local Resource Server

RRS Remote Resource Server

PT Permission Token

VIT Verified Identity Token

RPT Requesting Party

**CHAPTER 1**

**INTRODUCTION**

**1.1 GENERAL BACKGROUND**

People like to travel and may visit different doctors across different jurisdictions (such as countries). eHealth data of individuals is managed by the diverse health service providers and stored in different locations. Although there are many agreements between different jurisdictions (such as countries), in general they do not allow eHealth data to be shared externally. Often, sharing is not even allowed within the same country between different healthcare providers. There are several reasons but, as a general theme, it is due to a lack of trust or data disclosure considerations stemming from compliance and regulations. There is a consensus that healthcare data is sensitive personal information that must be well protected.

This project tackle the problem of sharing eHealth data across different jurisdictions. The overall challenge stems from three fundamental problems. First, there needs to be full accountability (e.g., non-repudiation) when sharing data; accountability also refers to the possibility of someone sharing healthcare data without the authorization of the patient. Second, since there is no global infrastructure to discover the locations of eHealth data, it must be a truly decentralized scheme. Third, before sharing eHealth data, explicit consent must be obtained from the data subject. The eHealth data custodian needs to be able to demonstrate that appropriate measures had been taken to address these problems if there is an audit or breach in the future.

**1.2 PROBLEM STATEMENT**

Suppose a person ‘A’ with a health issue born in France registered a GP(FGP),after 10 years, moved to UK then to Canada when felt sick and visited CGP. Returned to UK, visited BGP. Here there is a need of sharing her previous medical records from Canada and France.

● Four entities :- Data Subject(A), Requesting Party(BGP),Data controllers-FGP and CGP-Independent parties in different jurisdictions, no global system to communicate.

● ‘A’ needs to intermediate request and grant access, in turn needs to authenticate against FGP and CGP.

The key requirements of a distributed cross-jurisdiction eHealth data exchange architecture needs to target auditability, non-repudiation, confidentiality, and compatibility.

There is a consensus that healthcare data is sensitive personal information that must be well protected. Therefore different countries have imposed severe national regulations on sharing of healthcare data cross jurisdiction .In case of emergencies, it is very difficult. To illustrate the problem with an example, analyse a simple working scenario of international eHealth data exchange. Alice (A) has had heart trouble since she was born in France. She has registered a GP (FGP) in her original city. Then she moved to the UK after 10 years. When she was 20 years old and travelled to Canada, she fell sick and visited a GP (CGP) to get e temporary treatment in Canada. After that trip, she returned to the UK and visited her British doctor (BGP, where GP stands for ”General Practitioner”). She would like to share the British GP with her previous medical data which is stored respectively in Canada and France. However, she doesn’t want to simply give British GP her personal credential but would rather authorize British GP which could gain access using GP’s credential. In the scenario, there are four entities: the data subject A, a requesting party BGP, and data controllers which are healthcare service providers FGP and CGP. To note that A,BGP, CGP, and FGP, are independent parties located in different jurisdictions. Some of them are not known to the others; their only point of contact is their relationship with A, the patient and data subject. Furthermore, assume that is overwhelmingly the case, that there is no global system in place that allows all parties to directly communicate, find each other, or self-certify. In other words, BGP needs to access A’s medical data from FGP and CGP but FGP and CGP do not recognize BGP so A needs to intermediate the request and grant access. A, in turn, needs to authenticate against FGP and CGP. Healthcare is a sensitive domain that poses and processes a large amount of personal medical data daily. Regarding the requirement of eHealth data exchange, the application of blockchain (Distribute ledger) technology in eHealth data exchange is a continuous hot topic. The above problem is trying to be solved by combining such different technologies.

**1.3 OBJECTIVE**

The problem of sharing eHealth data across different jurisdictions stems from these three fundamental problems.

* There needs to be full accountability (e.g., non-repudiation) when sharing data.
* There is no global infrastructure to discover the locations of eHealth data.
* Before sharing eHealth data, explicit consent must be obtained from the data subject.

**1.4 PURPOSE AND SCOPE**

This project focus on auditability, confidentiality, and distributedness in cross-jurisdiction eHealth data exchange. Firstly, minimize the information exchanged in the sharing network. It is designed to verify identity information in the local jurisdiction and it is not necessary to check and share identity information cross-jurisdiction. Secondly, a scheme where lightweight and short-lived authorization tokens are created and shared between entities to access eHealth data cross region. For the identity information, a token without identity information is produced by authorization party to represent the verified identity. All entities share and verify these tokens with each other to create an individual identity while not disclosing information. Tokens are shared through blockchain with smart-contracts, which are system-agnostic. This project tackle the problem of sharing eHealth data across different jurisdictions.

**CHAPTER - 2**

**LITERATURE SURVEY**

“User-Controlled, Auditable, Cross-Jurisdiction Sharing of Healthcare Data Mediated by a Public Blockchain”- Xiaohu Zhou, Vitor Jesus, Yonghao Wang and Mark Josephs-2020 [1]**.** This paper tackle the problem of eHealth data is managed by diverse health service providers with an agreement between different jurisdictions not to share the data externally due to lack of trust or data disclosure considerations by introducing a decentralized scheme known as BRUE.

The above paper propose a BRUE scheme:-

* Centring all information exchange and control on data subject,
* Combine a set of technologies and standards (Blockchain, Receipts, UMA for eHealth data sharing).

“Blockchain technology: Application in health care”- S. Angraal, M. H. Krumholz, and L. W. Schulz-2017 [3]**.** Assess several healthcare applications based on blockchain. Point out that key limitations of blockchain technology expanded to large-scale production deployment in future research are system scalability, security and cost-effectiveness

“Metrics for assessing blockchain-based healthcare decentralized apps,”-P. Zhang, M. A. Walker, J. White, D. C. Schmidt, and G. Lenz-2017 [4]. Define set of evaluation metrics for blockchain based healthcare decentralized applications to guild development of blockchain applications in health domain. It Includes cost effectiveness, patient-centered care model, system scalability, inter operability, user identification and so on. Concern only HIPAA requirements.(Health care regulation in US).

“Blockchain in healthcare applications: Research challenges and opportunities”-T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He-2019[5].It compares nine types of existing blockchain based applications in healthcare and give tips on how blockchain technology meet which requirements of the healthcare industry. They also present limitations and technical issues of blockchain technology, such as mining incentives and standardization, although its applications have potential benefits for the healthcare industry.

“Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems”- R. Guo, H. Shi, Q. Zhao, and D. Zheng-2018 [7]. MA-ABS scheme is using blockchain to exchange encapsulated electronic medical record with an attribute-based signature scheme authorized by multiple authorities. In this approach, although the exchanged message is endorsed by participants without any information disclosure, it has a scalability problem due to a significant volume of storage of EMRs data in the blockchain.

“Blockchain: A panacea for healthcare cloud-based data security and privacy?”- C. Esposito, A. De Santis, G. Tortora, H. Chang, and K. R. Choo-2018 [12]. Suggest an approach that uses conventional or distributed database to store medical data and an online chain to record hash values of those data. It uses a private blockchain to store eHealth data and a consortium blockchain to record the secure indexes of eHealth data. These approaches have benefits on access control and confidentiality because of storing all data in the blockchain, they are not compliant with regulations and specifications.

**CHAPTER-3**

**SYSTEM ANALYSIS**

**3.1 Expected System Requirements**

**3.1.1 Web3.py**

Web3.py is a Python library that serves as a bridge between Python applications and the Ethereum blockchain, enabling developers to interact with smart contracts, execute transactions, and retrieve data from the blockchain. It is part of the Web3 ecosystem, which provides tools and libraries for building decentralized applications (dApps) on the Ethereum network. Web3.py simplifies the process of interacting with Ethereum by abstracting away the complexities of interacting with the low-level JSON-RPC API. It provides a high-level and user-friendly interface for interacting with smart contracts and performing blockchain operations.

One of the primary features of Web3.py is its ability to connect to Ethereum nodes. It supports various Ethereum client implementations such as Geth and Parity, allowing developers to choose the client that best suits their needs. Web3.py can connect to both local and remote nodes, providing flexibility in deployment options. Once connected, developers can leverage the library's capabilities to interact with the Ethereum blockchain.

Web3.py allows developers to interact with smart contracts deployed on the Ethereum network. It provides an abstraction layer that enables developers to easily instantiate and interact with smart contracts using Python objects. Developers can define contract classes that represent the smart contract's functions, events, and variables. This abstraction simplifies the process of interacting with smart contracts, making it more intuitive and less error-prone.

 Using Web3.py, developers can send transactions to the Ethereum network to execute smart contract functions or transfer Ether. The library handles the transaction creation, signing, and submission process, abstracting away the low-level details. It provides methods for estimating gas costs, setting gas prices, and handling transaction receipts.

Web3.py also enables developers to retrieve data from the Ethereum blockchain. It provides methods to query account balances, fetch block information, and retrieve transaction details. Developers can also subscribe to events emitted by smart contracts, allowing them to react to specific events on the blockchain in real-time. This functionality is useful for building applications that require up-to-date information from the Ethereum network.

Web3.py supports Ethereum's standard data types, such as addresses, integers, strings, and byte arrays. It handles the conversion between Python objects and Ethereum data types, making it convenient to work with data on the blockchain. The library also supports various utility functions, such as encoding and decoding ABI (Application Binary Interface) data, hashing, and cryptographic operations. Web3.py provides a modular and extensible architecture. It allows developers to customize and extend its capabilities through plugins and middleware. This flexibility enables developers to integrate additional functionality into their applications, such as support for custom authentication mechanisms or alternative data storage solutions.

**3.1.2 Django**

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Django is a powerful and popular Python web framework that enables developers to quickly and efficiently build robust web applications. It follows the model-view-controller (MVC) architectural pattern, emphasizing the separation of concerns and promoting reusable code. With its clean design, pragmatic approach, and extensive ecosystem, Django has become a go-to choice for developers across the globe.

At its core, Django provides a collection of tools, libraries, and conventions that simplify the development process. It encourages the use of reusable components and follows the "Don't Repeat Yourself" (DRY) principle, which reduces code duplication and increases maintainability. Django's philosophy revolves around making it easier to build complex web applications with minimal effort.

One of Django's standout features is its object-relational mapping (ORM) layer. This ORM allows developers to interact with databases using Python objects, abstracting away the complexities of SQL queries. With the ORM, developers can define database models using Python classes and perform database operations such as querying, inserting, updating, and deleting records using intuitive and expressive Python syntax. The ORM supports various database backends, including popular choices like PostgreSQL, MySQL, and SQLite.

Django follows the convention over configuration principle, providing sensible defaults that enable developers to start building applications immediately. It includes an automatic admin interface that can be easily customized to manage application data without writing additional

code. The admin interface is highly extensible and allows developers to define custom views, forms, and actions tailored to their application's specific needs.

Another core component of Django is its URL routing system. It maps URLs to appropriate views, which are Python functions or classes that handle HTTP requests and return responses.

Django's URL routing system supports regular expressions and allows for dynamic URL patterns, enabling the creation of flexible and scalable applications. Additionally, Django provides a powerful template engine, which simplifies the process of rendering dynamic HTML pages by separating the presentation logic from the application's business logic.

Django is designed to facilitate the development of secure web applications. It includes built-in protection against common web vulnerabilities such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. Django's security features are enabled by default, providing developers with a solid foundation to build secure applications. Additionally, Django supports user authentication, authorization, and session management, making it easy to implement user-centric features.

Django's extensibility is one of its defining characteristics. It offers a vast ecosystem of third-party packages, known as Django apps, which provide additional functionality to enhance Django applications. These apps cover a wide range of areas, including user authentication, content management, RESTful APIs, caching, and much more. The Django community actively maintains and updates these packages, ensuring compatibility with the latest Django versions and promoting code reuse. Django also provides robust support for internationalization and localization. It offers built-in tools for translating web applications into multiple languages, handling date and time formats, and managing localized content. This makes it easy to develop applications that cater to a global audience.

**3.1.3 Python**

Python is a versatile and widely-used programming language known for its simplicity, readability, and flexibility. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability and focuses on providing a clear and expressive syntax, making it accessible to beginners while still being powerful enough for experienced developers. One of the key strengths of Python is its simplicity and ease of use. The language was designed to have a clean and straightforward syntax that prioritizes human-readable code. Python's syntax uses indentation to define code blocks, eliminating the need for braces or explicit delimiters. This readability contributes to the ease of understanding and maintaining Python code, even for developers new to the language. Python has a large and active community of developers who contribute to its growth and maintain a rich ecosystem of

libraries and frameworks. The Python Package Index (PyPI) hosts thousands of open-source libraries that cover a wide range of domains, enabling developers to leverage existing solutions and accelerate their development process. Popular frameworks like Django for web development, NumPy and pandas for data analysis, and TensorFlow for machine learning have made Python a go-to language for various applications. The flexibility of Python allows it to be used in diverse fields, including web development, data analysis, scientific computing, machine learning, artificial intelligence, automation, and more. Its extensive standard library provides a wide array of modules and functions for performing common tasks, reducing the need for external dependencies and simplifying the development process.

Python's dynamic typing system allows for quicker development and iteration. Variables in Python are not explicitly declared with types, making it easier to write and modify code. This dynamic nature also allows for more flexible and expressive code, as developers can create and manipulate data structures without strict type constraints.

Python supports multiple programming paradigms, including object-oriented, procedural, and functional programming. This versatility enables developers to choose the approach that best suits their problem domain and coding style. Python's object-oriented features make it straightforward to design and build complex applications by organizing code into reusable and modular components. The cross-platform compatibility of Python enables developers to write code once and run it on multiple operating systems, including Windows, macOS, and various flavors of Linux. This portability makes Python an attractive choice for projects that require cross-platform support.

**3.2 Software Requirements**

**3.2.1 SQLite**

SQLite is a popular and widely-used relational database management system (RDBMS) that is known for its simplicity, lightweight nature, and self-contained architecture. It is an open-source database engine that doesn't require a separate server process to operate, making it ideal for embedded systems, mobile applications, and small-scale web applications.

One of the key advantages of SQLite is its simplicity and ease of use. It offers a straightforward and intuitive SQL interface for managing databases and performing common database operations such as creating tables, inserting data, querying data, updating records, and deleting

data. The SQL syntax used in SQLite follows the SQL-92 standard with some additional features and optimizations. SQLite is designed to be lightweight and efficient. It is implemented as a small, compact library that can be directly integrated into applications. The

entire SQLite database is contained in a single file, making it easy to distribute and deploy. This self-contained architecture eliminates the need for a separate database server and simplifies the installation and maintenance process.

Despite its lightweight nature, SQLite is powerful and feature-rich. It supports most standard SQL features and provides a wide range of data types, including integers, floating-point numbers, strings, blobs, and dates. It also supports advanced features such as triggers, views, indexes, and full-text search, allowing developers to build complex and efficient database systems.

Another notable feature of SQLite is its transactional support. It provides transaction management capabilities, allowing multiple operations to be executed atomically and ensuring data consistency. SQLite follows the ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring that transactions are processed reliably and consistently, even in the presence of failures or interruptions. SQLite databases are highly portable and can be used across various platforms and operating systems, including Windows, macOS, Linux, iOS, Android, and more. This portability allows developers to build applications on one platform and seamlessly transfer them to another without significant changes to the codebase.

In conclusion, SQLite is a versatile and lightweight RDBMS that offers simplicity, portability, and powerful features. Its self-contained architecture, transactional support, SQL compatibility, and wide platform support make it an excellent choice for applications that require a compact and efficient database solution. Whether for embedded systems, mobile applications, or small-scale web applications, SQLite provides a reliable and easy-to-use database engine.

**3.2.2 Mailtrap**

Mailtrap is a popular email testing and debugging tool that provides a simulated email environment for developers and QA teams. It allows users to send and receive test emails without actually delivering them to real recipients, ensuring that email functionality is thoroughly tested and debugged before deploying it to production. At its core, Mailtrap provides a dummy SMTP server that captures all outgoing emails from the application under

test. Instead of sending emails to real recipients, Mailtrap stores them in a virtual inbox accessible through its web-based interface. This eliminates the risk of accidentally sending test emails to real users or flooding inboxes during development and testing phases.

One of the key advantages of Mailtrap is its ease of setup and integration. Developers can quickly configure their applications to use the Mailtrap SMTP server by updating the SMTP settings with the provided credentials. Mailtrap supports various programming languages and frameworks, making it compatible with a wide range of applications.

With Mailtrap, developers can test different aspects of their email functionality. They can verify the content and formatting of emails, test attachments, check for proper handling of email headers, and ensure that the emails are sent at the correct times. By inspecting the captured emails in the Mailtrap inbox, developers can review the email content and troubleshoot any issues that may arise.

Mailtrap offers a powerful set of features to assist with email testing and debugging. It allows users to organize emails into inboxes, making it easier to manage and categorize different types of test emails. Users can create multiple inboxes for different projects or testing scenarios. Additionally, Mailtrap provides features like search, filtering, and sorting to quickly find and analyze specific emails.

To facilitate collaboration among team members, Mailtrap allows sharing of inboxes with team members and stakeholders. This enables multiple people to access and review test emails, ensuring that everyone involved in the project is on the same page. Mailtrap also provides the ability to forward test emails to external email addresses, making it convenient to share email samples with clients or other stakeholders outside the testing environment.

Mailtrap offers integration with popular email delivery services, such as SendGrid and Mailgun. This allows developers to seamlessly switch between the testing environment provided by Mailtrap and the production email delivery services. By connecting Mailtrap to these services, developers can verify that emails are properly configured to work with the chosen email delivery provider.

In addition to the web-based interface, Mailtrap provides a powerful API that allows programmatic access to its features. This enables developers to automate testing workflows, integrate Mailtrap with testing frameworks or continuous integration pipelines, and perform advanced analysis or manipulation of test emails.

Mailtrap ensures the security and privacy of test emails. All data transmitted to and from the Mailtrap server is encrypted using industry-standard security protocols. Moreover, Mailtrap adheres to data protection regulations and follows best practices to protect user data.

**3.2.3 Ganache CLI**

Ganache CLI (Command Line Interface) is a powerful development tool for Ethereum blockchain developers. It is part of the Ganache suite of tools provided by Truffle, a popular development framework for Ethereum-based applications. Ganache CLI allows developers to create a local Ethereum network with a set of pre-funded accounts, making it easier to test and debug smart contracts without the need for a real blockchain network.

One of the key features of Ganache CLI is its ability to create a local Ethereum network that mimics the behaviour of a real network. This local network is fully functional and allows developers to deploy and interact with smart contracts, simulate transactions, and observe the behaviour of the blockchain. However, since it is a local network, developers have full control over the network's configuration, making it ideal for testing and development purposes.

Ganache CLI provides a simple and intuitive command line interface for managing the local Ethereum network. Developers can start the network with a single command, specifying various options such as the number of accounts to create, initial account balances, gas limits, and network ID. This allows developers to configure the network to match their specific testing requirements.

Once the local Ethereum network is running, developers can interact with it using tools such as Truffle, Web3.js, or any other Ethereum development libraries. They can deploy smart contracts to the network, execute transactions, and simulate various scenarios to test the functionality and behaviour of their smart contracts. Ganache CLI provides detailed logs and transaction information, making it easier to track and debug contract interactions.

Ganache CLI also offers a range of advanced features to enhance the development experience.

One of the notable features of Ganache CLI is its support for account management. It automatically generates a set of accounts with pre-defined private keys and balances when the

network starts. These accounts can be easily imported into development tools or wallets for testing and experimentation. Developers can also programmatically interact with these accounts using the provided account management APIs. Ganache CLI provides a clean and user-friendly interface to monitor and inspect the state of the local Ethereum network. It

includes a web-based dashboard that displays information such as account balances, transaction history, block details, and contract deployments. This dashboard allows developers to quickly observe and analyze the network's behaviour, making it easier to identify and resolve any issues that may arise during testing.

**3.2.4 Tailwind CSS**

Tailwind CSS is a highly popular utility-first CSS framework that streamlines the process of building user interfaces. It provides a comprehensive set of pre-designed utility classes that developers can leverage to style their web applications. Unlike traditional CSS frameworks that provide pre-built components, Tailwind CSS focuses on utility classes that can be combined to create customized and responsive designs.

One of the key features of Tailwind CSS is its extensive collection of utility classes. These classes encapsulate specific CSS properties, such as margins, padding, typography, colors, and more. By applying these utility classes directly to HTML elements, developers can quickly and efficiently style their UI without the need to write custom CSS. This modular approach offers flexibility and enables rapid development and prototyping.

Tailwind CSS follows a mobile-first approach, ensuring that designs are responsive and adapt well to different screen sizes. It provides a range of responsive utility classes that can be used to modify styles based on breakpoints, making it easy to create responsive layouts. These classes allow developers to control the appearance of elements on various devices, resulting in a consistent user experience across different screen sizes.

One of the advantages of Tailwind CSS is its customizable nature. Developers can configure the framework to tailor it to their specific project requirements. The configuration file allows for the customization of color palettes, spacing scales, font families, breakpoints, and more. This flexibility empowers developers to create unique and branded designs that align with their project's visual identity.

Tailwind CSS promotes a utility-first mindset, which encourages reusability and composability. By leveraging utility classes, developers can easily compose complex layouts and styles by combining and chaining classes. This approach fosters a modular and scalable development process, where styles can be reused and modified with ease, reducing code duplication and improving maintainability.

Despite its utility-first approach, Tailwind CSS does not sacrifice readability and maintainability. The framework provides an intuitive naming convention for utility classes that is easy to understand and remember. Additionally, Tailwind CSS includes a powerful set of tools, such as the JIT (Just-In-Time) compiler, which optimizes and purges unused styles, resulting in smaller file sizes and improved performance.

**3.2.5 HTML**

The Hyper Text Markup Language or HTML is the standard [markup language](https://en.wikipedia.org/wiki/Markup_language) for documents designed to be displayed in a [web browser](https://en.wikipedia.org/wiki/Web_browser). It can be assisted by technologies such as [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS) and [scripting languages](https://en.wikipedia.org/wiki/Scripting_language) such as [JavaScript](https://en.wikipedia.org/wiki/JavaScript).

[Web browsers](https://en.wikipedia.org/wiki/Web_browser) receive HTML documents from a [web server](https://en.wikipedia.org/wiki/Web_server) or from local storage and [render](https://en.wikipedia.org/wiki/Browser_engine) the documents into multimedia web pages. HTML describes the structure of a [web page](https://en.wikipedia.org/wiki/Web_page) [semantically](https://en.wikipedia.org/wiki/Semantic_Web) and originally included cues for the appearance of the document. [HTML elements](https://en.wikipedia.org/wiki/HTML_element) are the building blocks of HTML pages. With HTML constructs, [images](https://en.wikipedia.org/wiki/HTML_element#Images_and_objects) and other objects such as [interactive forms](https://en.wikipedia.org/wiki/Fieldset) may be embedded into the rendered page. HTML provides a means to create [structured documents](https://en.wikipedia.org/wiki/Structured_document) by denoting structural [semantics](https://en.wikipedia.org/wiki/Semantics) for text such as headings, paragraphs, lists, [links](https://en.wikipedia.org/wiki/Hyperlink), quotes, and other items. HTML elements are delineated by tags, written using [angle brackets](https://en.wikipedia.org/wiki/Bracket#Angle_brackets). Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page. HTML can embed programs written in a [scripting language](https://en.wikipedia.org/wiki/Scripting_language) such as [JavaScript](https://en.wikipedia.org/wiki/JavaScript), which affects the behaviour and content of web pages..

**CHAPTER 4**

**METHODOLOGY**

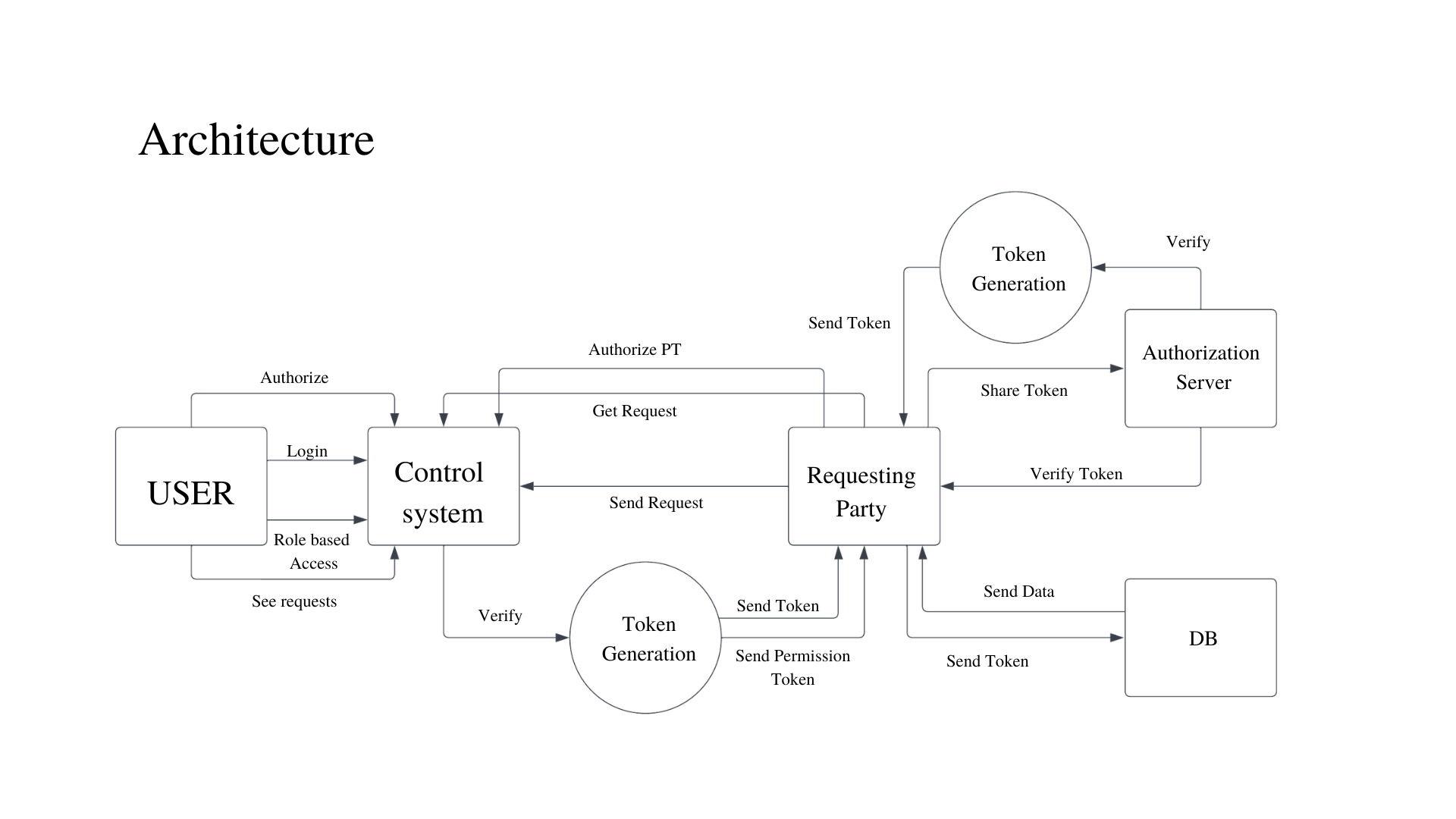
**4.1 Proposed System**

Fig 4.1 Proposed System

The proposed system follows an approach which provides the feasibility of cross-jurisdiction eHealth data exchange. It produces receipt record as participants’ consent integrated with UMA standards to target auditability and compatibility; in addition to, runs on a public blockchain to achieve non-repudiation and confidentiality with blockchain’s features.

At first, RqP requests service from the data subject. RO processes this request on behalf of the data subject. RO checks service request and then returns a PT. Then, RqP pulls PT and then sends to LAS with identity information for identity verification. LAS returns a V IT to RqP after verification. RqP uses VIT to RAS for identity and authorization check in the cross-jurisdiction. RAS then gives a RPT to RqP for final progress of eHealth data access. RqP shares RPT with RRS to get eHealth data. After permission check, RRS returns the required eHealth data to RqP. The whole data flow terminates at this point. During transactions progresses, all

authorized tokens are pushed to the blockchain firstly and then pulled by entities always mediated by the blockchain. In the blockchain, smart-contracts are triggered by transactions to process tokens sharing. A receipt is always generated following transaction of new information, such as a new token generation and push. All data flow of transactions from entities go through the blockchain network (for provenance and accountability). A receipt of each transaction is also produced following the data flow.

**CHAPTER 5**

**SYSTEM DESIGN**

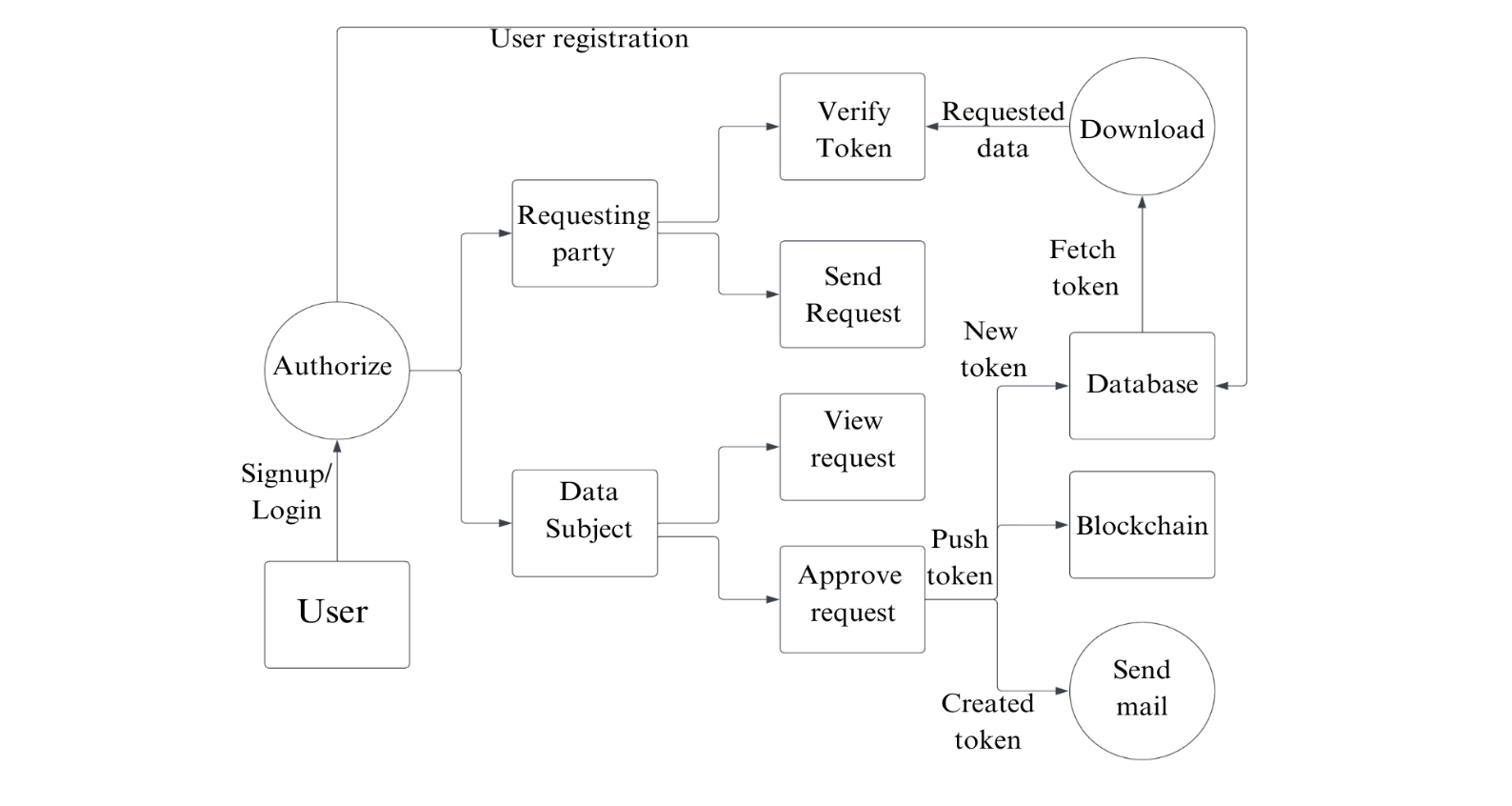
**5.1 Architecture**

Fig 5.1 Architecture

In the system, users are granted access based on their user type, allowing for role-based authentication. The Requesting Party, responsible for initiating data access requests, can send a request to the Data Subject, who holds the patient's eHealth data. Upon logging into the system, the Data Subject is presented with a filtered list of requests, specifically tailored to their role and permissions. They can review the requests and choose to grant access by creating a token that authorizes the Requesting Party to access the data. The Data Subject also has the flexibility to set the duration for which the token remains active, providing control over the data access period. Once the token is generated, it is securely transmitted to the Requesting Party. The Requesting Party, armed with the authorized token, can proceed to the Authorization Server, where the data is stored. After undergoing the necessary verification processes, the Requesting Party is granted access to download the authorized eHealth data. This

comprehensive workflow ensures that the Data Subject maintains control over the data while enabling seamless and secure access for authorized parties.

**5.2 Diagrams**

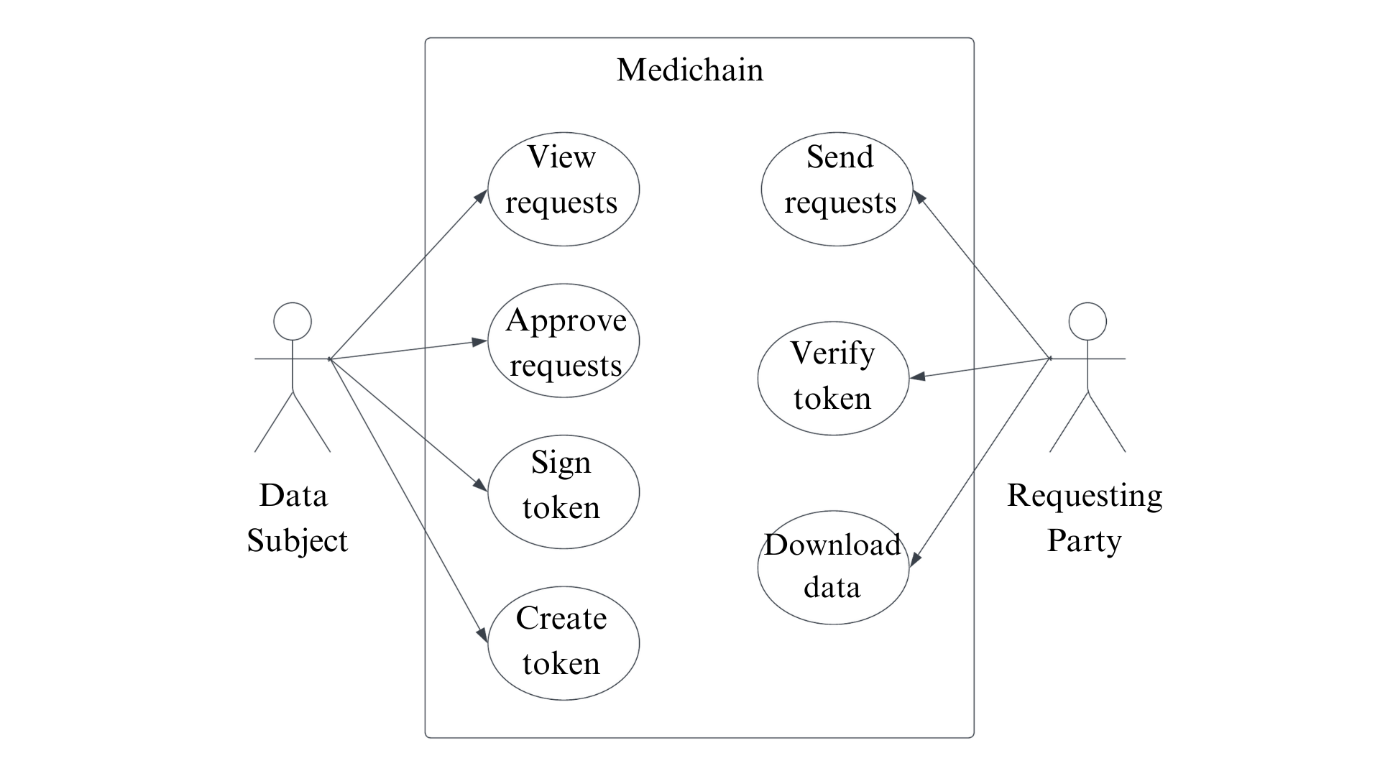
**5.2.1 Use Case Diagram**

Fig 5.2 Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system’s functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system. The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system’s requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

**Actors:**

* Data Subject: The patient who can review requests and generate permission tokens
* Requesting Party : Represents the individuals or organizations that will use the system to request and access healthcare data
* Control System : Represents the system that receives and processes requests for healthcare data from users

**Use Cases:**

* Request Healthcare Data : Represents the ability of users to request healthcare data from the control system
* Generate Token : Represents the ability of the authorization server to generate tokens for accessing healthcare data after verifying the identity of requesting parties
* Access Healthcare Data : Represents the ability of users to access healthcare data using the tokens issued by the authorization server

**5.2.2 DATA FLOW DIAGRAM**

A data-flow diagram is a way of representing a flow of data through a [process](https://en.wikipedia.org/wiki/Process) or a system (usually an [information system](https://en.wikipedia.org/wiki/Information_system)). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow — there are no decision rules and no loops. Specific operations based on the data can be represented by a [flowchart](https://en.wikipedia.org/wiki/Flowchart).

**5.2.2.1 LEVEL 0**

User is logged into the system based on the type of user they are. Requesting party sends request to the Data subject for gaining access to the patient’s eHealth data. The requests are filtered and shown to the Data subject upon logging in. They can approve the request by creating a token that grants them the permission to access the data. The Data subject can decide the time period for the token to be active. Then the token is send to the Requesting party, which they can use to go to the Authorization server and download the data after verification.

A picture containing text, diagram, font, line

Description automatically generated

Fig 5.3 DFD Level 0

**5.2.2.2 LEVEL 1**

A picture containing text, font, diagram, circle

Description automatically generatedThe requesting party sends the request and it is processed by the patient. A token is generated with the consent status. The generated token is pushed to the blockchain and the Database. After that the token is mailed to the Requesting party by fetching their email address from the database. This token can be used to access the data from the server where the data is stored. During transactions progresses, all authorized tokens are pushed to the blockchain In the blockchain, smart-contracts are triggered by transactions to process tokens sharing.

Fig 5.4 DFD Level 1

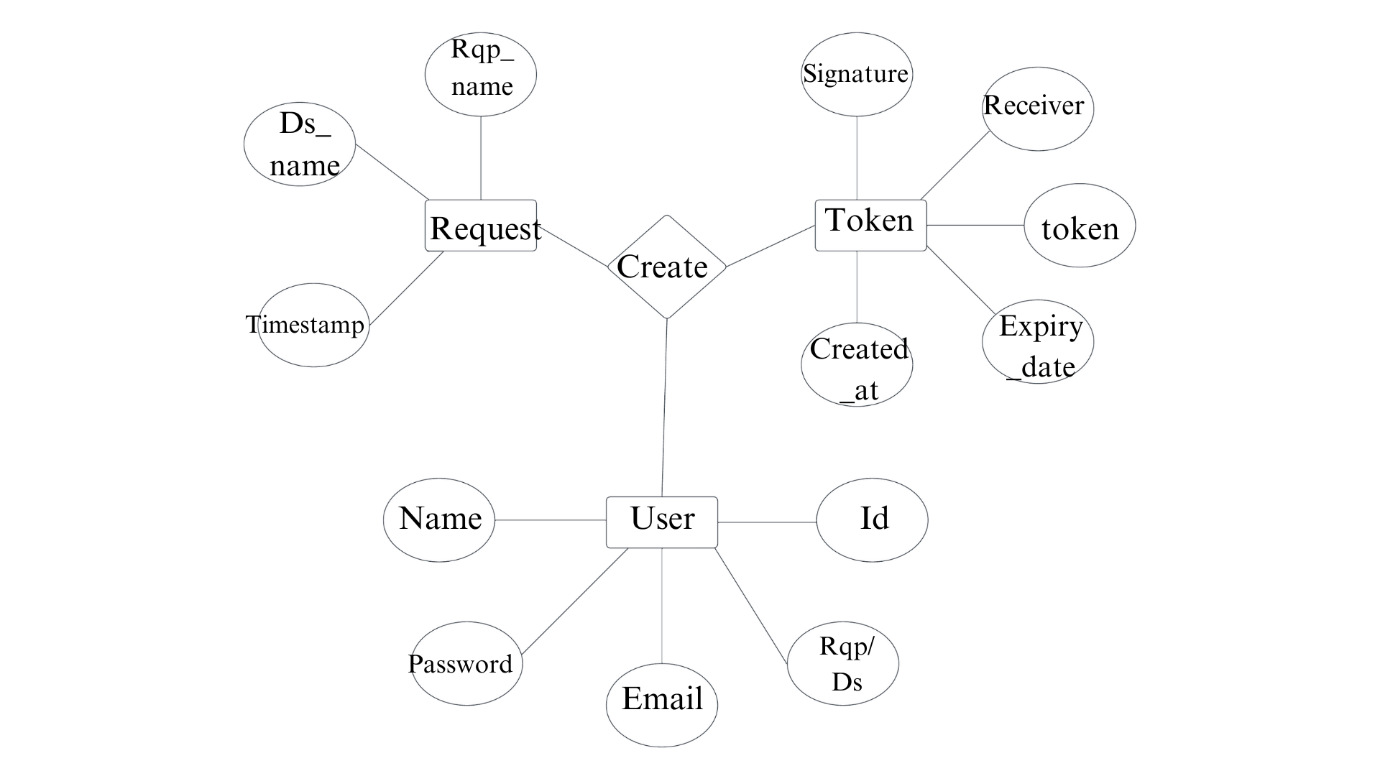
**5.2.2.3 LEVEL 2**

Requesting party goes to the authorization server to verify the token by entering the token number and the expiry date. After verification, the Requesting party can download the requested data.

A picture containing text, diagram, font, line

Description automatically generated

Fig 5.5 DFD Level 2

**5.2.3 E-R Diagram**

5.6 E-R Diagram

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

**6.1 Smart Contract For The System**.

// SPDX-License-Identifier: MIT

pragma solidity  ^0.8.0;

contract Token {

    event NewToken(string signature, string receiver, string token);

    struct TokenData {

        string signature;

        string receiver;

        string token;

    }

    TokenData[] public tokens;

    mapping(uint => address) public tokenToOwner;

    mapping(address => string) public signToToken;

    function saveToken(string memory signature, string memory receiver, string memory token) public payable returns (bool) {

        uint id = tokens.length;

        tokens.push(TokenData(signature, receiver, token));

        tokenToOwner[id] = msg.sender;

  signToToken[msg.sender] = signature;

emit NewToken(signature, receiver, token);

        return true;

    }

    function getToken(string memory signature, string memory receiver) public view returns (string memory) {

        for (uint I = 0; I < tokens.length; i++) {

            TokenData storage myToken = tokens[i];

            if (keccak256(bytes(signature)) == keccak256(bytes(myToken.signature)) && keccak256(bytes(receiver)) == keccak256(bytes(myToken.receiver))) {

                return myToken.token;

            }

        }

        return “wrong input or no record”;

    }

}

The given Solidity contract is called “Token,” and it allows users to save and retrieve token data based on a signature and receiver. The contract includes an event called “NewToken” that emits information about the new token being saved. It defines a struct called “TokenData” that holds the signature, receiver, and token information. The tokens are stored in an array called “tokens,” and mappings are used to associate each token’s ID with the owner’s address and each address with its corresponding signature. The “saveToken” function allows users to save a new token by providing the signature, receiver, and token data. It assigns an ID to the new token, updates the mappings, and emits the “NewToken” event. The “getToken” function enables users to retrieve the token data by providing the signature and receiver. It iterates through the tokens, compares the input with stored data, and returns the corresponding token if a match is found. If no match is found, it returns a message indicating a wrong input or no record. Overall, the contract provides basic functionality for storing and retrieving token information based on specified criteria.

**6.2 Code for token creation in server side**

def create\_token(request):

    if request.method == ‘POST’:

        form = TokenForm(request.POST)

        if form.is\_valid():

            # Create a new Token object from the form data

            signature = form.cleaned\_data[‘signature’]

            receiver = form.cleaned\_data[‘receiver’]

            token = str(random.randint(0, 9999))

            expiry\_date = form.cleaned\_data[‘expiry\_date\_choice’]

            user = User.objects.get(username=receiver)

            email = user.email

            token\_obj = Token(signature=signature, receiver=receiver, token=token, expiry\_date=expiry\_date)

            token\_obj.save()

            # Send a transaction to save a new token

            tx\_hash = contract.functions.saveToken(signature, receiver, token).transact()

            web3.eth.waitForTransactionReceipt(tx\_hash)

            print(“Transaction confirmed!”)

            send\_mail(‘Token is created’, str(token\_obj), ‘settings.EMAIL\_HOST\_USER’, [email])

return redirect(‘chain:token\_created’)

  else:

        form = TokenForm()

    return render(request, ‘chain/create\_token.html’, {‘form’: form})

The function create\_token(request) is triggered when an HTTP POST request is made to the corresponding URL. It first checks the request method to ensure it is a POST request. It then initializes a TokenForm object using the POST data received.

If the form is valid, the code proceeds to extract the relevant data from the form, such as the signature, receiver, and expiry date. It generates a random token and retrieves the email associated with the receiver’s username from the User model.

Next, a Token object is created using the extracted data, and it is saved to the database. The code then interacts with a smart contract using Web3 and Ethereum to save the token. It calls the saveToken function of the contract with the signature, receiver, and token data, and waits for the transaction to be confirmed using waitForTransactionReceipt.

Upon successful confirmation of the transaction, a confirmation message is printed, and an email is sent to the receiver with the token details using the send\_mail function. Finally, the user is redirected to a specific page indicating that the token has been successfully created.

If the request method is not POST, indicating an initial GET request, the code simply initializes an empty TokenForm object.

Overall, the code handles the creation of a new token by processing form data, saving the token in the database, interacting with a smart contract to store the token on the blockchain, sending email notifications, and rendering the appropriate template with the form.

**6.3 Data Models**

The various data models in our system are defined as below

from django.db import models

class Request(models.Model):

    Rqp\_name = models.CharField(max\_length=100)

Ds\_name = models.CharField(max\_length=100)

  timestamp = models.DateTimeField(auto\_now\_add=True)

class Token(models.Model):

signature = models.CharField(max\_length=100)

    receiver = models.CharField(max\_length=100)

    token = models.CharField(max\_length=100)

    expiry\_date = models.DateField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f”{self.signature} – {self.receiver} – {self.token} – {self.expiry\_date}”

class User(AbstractUser):

    Data\_Subject = models.BooleanField(‘Data Subject’, default=False)

    Requesting\_party = models.BooleanField(‘Requesting Party’, default=False)

**6.3.1 Request Model**

Rqp\_name: A CharField that stores the name of the request, with a maximum length of 100 characters.

Ds\_name: A CharField that stores the name of the data source, with a maximum length of 100 characters.

Timestamp: A DateTimeField that automatically records the timestamp when the object is created.

**6.3.2 Token Model**

Signature: A CharField that stores the signature of the token, with a maximum length of 100 characters.

Receiver: A CharField that stores the receiver of the token, with a maximum length of 100 characters.

Token: A CharField that stores the token value, with a maximum length of 100 characters.

Expiry\_date: A DateField that stores the expiry date of the token.

Created\_at: A DateTimeField that automatically records the timestamp when the object is created.

The Token model also defines a \_\_str\_\_ method that returns a formatted string representation of the model instance. The string includes the signature, receiver, token value, and expiry date.

**6.3.3 The User model**

Inherits from AbstractUser, which provides the default fields and functionality for a user in Django.

Adds two additional fields:

Data\_Subject: A BooleanField that represents whether the user is a data subject. It has the label ‘Data Subject’ and a default value of False.

Requesting\_party: A BooleanField that represents whether the user is a requesting party. It has the label ‘Requesting Party’ and a default value of False.

By extending AbstractUser and adding these custom fields, the User model allows for the creation of user accounts with the additional attributes of being a data subject or a requesting party. These fields can be used to store and retrieve information about the user’s role or permissions within the application.

The custom User model can be used in Django’s authentication system to manage user registration, login, and other authentication-related functionalities. It provides the flexibility to differentiate between different types of users based on the Data\_Subject and Requesting\_party fields.

**6.4 The functions used for downloading the data are:**

def token\_detail(request):

    if request.method == ‘POST’:

token\_number = request.POST[‘token\_number’]

expiry\_date = request.POST[‘expiry\_date’]

      try:

token = Token.objects.get(token=token\_number, expiry\_date=expiry\_date)

            token\_valid = True

  except Token.DoesNotExist:

            token\_valid = False

            token = None

        context = {

            ‘token’: token,

            ‘token\_valid’: token\_valid,

            ‘error’: ‘Invalid token or expiry date. Please try again.’ If not token\_valid else None

        }

        return render(request, ‘auth\_server/verify.html’, context)

    else:

        return render(request, ‘auth\_server/verify.html’)

def download\_file(request, path):

    file\_path = os.path.join(settings.MEDIA\_ROOT, path)

    with open(file\_path, ‘rb’) as fh:

        response = HttpResponse(fh.read(), content\_type=”application/octet-stream”)

        response[‘Content-Disposition’] = ‘attachment; filename=’ + os.path.basename(file\_path)

        return response

token\_detail(request):

This function handles a POST request to validate a token and its expiry date.

It retrieves the token\_number and expiry\_date from the request’s POST data.

It then tries to fetch a Token object from the database that matches the provided token number and expiry date.

If a matching token is found, the token\_valid variable is set to True, otherwise False.

The function prepares a context dictionary with the retrieved token, the token\_valid flag, and an error message if the token is invalid.

Finally, it renders the verify.html template, passing the context as the template’s variables.

download\_file(request, path):

This function is responsible for downloading a file located at the specified path.

It constructs the absolute file path by joining the MEDIA\_ROOT setting with the provided path argument.

The function opens the file in binary mode and reads its content.

It creates an HttpResponse object with the file content and sets the appropriate content type for a general binary file.

Additionally, it sets the Content-Disposition header to specify the file’s name for downloading.

Finally, it returns the HttpResponse object, which triggers the file download in the user’s browser.

These view functions can be used in conjunction with Django’s URL routing to handle specific URLs and provide corresponding functionality. The token\_detail function is typically used to validate tokens, while the download\_file function handles file downloads.

**CHAPTER 7**

**TESTING**

**7.1 Unit Testing for solidity code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "truffle/Assert.sol";

import "truffle/DeployedAddresses.sol";

import "../contracts/Token.sol";

contract TestToken {

Token token;

function beforeEach() public {

token = new Token();

}

function testSaveToken() public {

string memory signature = "abc";

string memory receiver = "john";

string memory tokenData = "12345";

bool result = token.saveToken(signature, receiver, tokenData);

Assert.isTrue(result, "Token not saved successfully");

Assert.equal(token.tokens.length, 1, "Token count mismatch");

Assert.equal(token.tokens[0].signature, signature, "Token signature mismatch");

Assert.equal(token.tokens[0].receiver, receiver, "Token receiver mismatch");

Assert.equal(token.tokens[0].token, tokenData, "Token data mismatch");

}

function testGetToken() public {

string memory signature = "abc";

string memory receiver = "john";

string memory tokenData = "12345";

token.saveToken(signature, receiver, tokenData);

string memory retrievedToken = token.getToken(signature, receiver):

Assert.equal(retrievedToken, tokenData, "Retrieved token mismatch");

}

function testGetTokenInvalid() public {

string memory signature = "def";

string memory receiver = "jane";

string memory retrievedToken = token.getToken(signature, receiver);

Assert.equal(retrievedToken, "wrong input or no record", "Invalid token mismatch");

}

}

A test contract TestToken is defined and use the beforeEach function to create a new instance of the Token contract for each test case.

The testSaveToken function tests the saveToken function by verifying that a token is successfully saved, and the relevant data matches the inputs.

The testGetToken function tests the getToken function by saving a token and then retrieving it using the same signature and receiver. It checks if the retrieved token matches the expected value.

The testGetTokenInvalid function tests the getToken function with invalid inputs and checks if the function returns the expected error message.

These tests cover the basic functionality of the contract, ensuring that tokens can be saved and retrieved correctly.

**7.2 Mail Testing**

Mail sending is done by using Django’s built-in SMTP settings. For testing this, Mailtrap.io was used and mails were sent to Mailtrap’s inbox. The settings for mail testing was defined as follows:

EMAIL\_HOST = 'sandbox.smtp.mailtrap.io'

EMAIL\_HOST\_USER = 'dd266e9a9b9b78'

EMAIL\_HOST\_PASSWORD = '#########'

EMAIL\_PORT = '2525'

**CHAPTER 8**

**RESULTS**

The eHealth data exchange platform project has been successfully implemented, leveraging the security benefits of blockchain technology and the robustness of Django web framework. The platform provides a secure and efficient solution for exchanging sensitive healthcare data while ensuring data integrity, privacy, and access control.

By incorporating blockchain technology, the platform ensures the immutability and tamper-proof nature of the data, making it highly resilient against unauthorized modifications or tampering attempts. The use of blockchain also enhances transparency and trust among the participating entities, as all data transactions and access permissions are recorded on the distributed ledger.

The integration of Django as the web framework has facilitated the development of a scalable, modular, and user-friendly platform. Django's powerful features such as authentication, authorization, and session management have been leveraged to implement robust user authentication mechanisms and role-based access control, ensuring that only authorized users can access and manipulate the data.

Throughout the project, various key functionalities have been successfully implemented. Users are categorized into Requesting Parties and Data Subjects, each with their respective roles and permissions. The Requesting Party can initiate data access requests, which are then filtered and presented to the Data Subject upon logging in. The Data Subject can review and approve the requests by generating tokens that grant temporary access to the eHealth data. The Data Subject has the flexibility to set the duration for which the tokens remain active, providing granular control over data access.

Furthermore, the platform incorporates an Authorization Server where the eHealth data is securely stored. The Requesting Party, armed with the authorized token, can undergo the necessary verification processes and download the authorized data from the Authorization Server.

Overall, the project has successfully achieved its objectives of building an eHealth data exchange platform that prioritizes security, privacy, and data integrity. By combining the strengths of blockchain technology and Django web framework, the platform provides a robust and user-friendly solution for secure healthcare data exchange. The implemented functionalities empower Data Subjects to maintain control over their data while enabling authorized parties to access the data seamlessly and securely.

Moving forward, the project opens avenues for future enhancements and expansion. Additional features such as data encryption, auditing capabilities, and integration with external systems can further enhance the platform's capabilities. Continuous monitoring, evaluation, and user feedback will be crucial in refining and optimizing the platform for better performance, usability, and security.

**SCREENSHOT**

A screenshot of a computer

Description automatically generated**Home Page**

Fig 8.1 Home Page

A screenshot of a computer

Description automatically generated with medium confidence**Sign-Up Page**

Fig 8.2 Sign-up Page

A screenshot of a computer

Description automatically generated with medium confidence**Login Page**

Fig 8, 3 Login page

A screenshot of a computer

Description automatically generated with medium confidence**Send Request**

Fig 8.4 Send Request Page

A screenshot of a computer

Description automatically generated with medium confidence**View Request**

Fig 8.5 View Request Page

A screenshot of a computer

Description automatically generated**Create Token**

Fig 8.6 Create Token

A screenshot of a computer

Description automatically generated**Send Mail**

Fig 8.7 Send Mail

A screenshot of a computer

Description automatically generated**Verify Token and Download Data.**

Fig 8.8 Verify Token and Download data

**CHAPTER 9**

**CONCLUSION**

Implemented the proposed system for eHealth data exchange and evaluated its feasibility. It relies on open protocols so that it is easier to adopt in existing systems. The system seems able to satisfy key requirements such as auditability, confidentiality, non-repudiation, and compatibility when supporting cross jurisdiction eHealth data exchange. Preliminary results of this prototype show it is not complex to implement and the user interface can be simple for non-technical users. Using Django made implementations so much easier with its built-in tools like, Authentication, SMTP server settings, SQLite etc.

This project also opens new research directions. On one hand, there are essential features that should be able to accommodate. For example, a break-the-glass procedure is needed for emergencies when the data subject cannot give timely consent for the resource owner. On the other hand, no assumptions were made on the actual eHealth data. Whereas the system is agnostic to it, selective disclosure of data, instead of everything, is a crucial feature that requires a data model and fine-grained permissions. Furthermore, the identity of participants is deliberately overlooked. Finally, a detailed threat model was not presented, and the assumption was made that all entities are honest. It is quite possible that the system, in its current stage, can be abused or manipulated, particularly in the case of availability and denial-of-service. Blockchain technology is also vulnerable to various attacks such as the Fork problem or scalability issues. Therefore, future focus should be given to avoid these attacks.

**REFERENCES**

[1] **E. Maler** (ed.), “User-managed access (uma) 2.0 grant for oauth 2.0 authorization,” Kantara Initiative Recommendation, 2018.

[2] **M. Lizar and D. Turner** (eds.), “Consent receipt specification,” Kantara Initiative Recommendation, 2017.

[3] **S. Angraal, M. H. Krumholz, and L. W. Schulz**, “Blockchain technology: application in health care,” Circulation: Cardiovascular Quality and Outcomes, vol. 10, no. 9, Sep 2017.

[4] **P. Zhang, M. A. Walker, J. White, D. C. Schmidt, and G. Lenz**, “Metrics for assessing blockchain-based healthcare decentralized apps,” in 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), 2017, pp. 1–4.

[5] **T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He**, “Blockchain in healthcare applications: Research challenges and opportunities,” Journal of Network and Computer Applications, vol. 135, pp. 62–75, Jun 2019.

[6] **T. K. Mackey, T. Kuo, B. Gummadi, A. K. Clauson, G. Church, D. Grishin, K. Obbad, R. Barkovich, and M. Palombini,** “’fit-for purpose?’ - challenges and opportunities for applications of blockchain technology in the future of healthcare,” BMC Medicine, vol. 17, no. 1, p. 68, 2019.

[7] **R. Guo, H. Shi, Q. Zhao, and D. Zheng**, “Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems,” IEEE Access, vol. 6, pp. 11 676–11 686, 2018.

[8] **A. Zhang and X. Lin,** “Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain,” Journal of Medical Systems, vol. 42, no. 140, June 2018.

[9] **K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang**, “Medblock: Efficient and secure medical data sharing via blockchain,” Journal of Medical Systems, vol. 42, June 2018.

[10] **Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani**, “Medshare: Trust-less medical data sharing among cloud service providers via blockchain,” IEEE Access, vol. 5, pp. 14 757–14 767, 2017.

[11] **A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman**, “Medrec: Using blockchain for medical data access and permission management,” in 2016 2nd International Conference on Open and Big Data (OBD), 2016, pp. 25–30.

[12] **C. Esposito, A. De Santis, G. Tortora, H. Chang, and K. R. Choo**, “Blockchain: A panacea for healthcare cloud-based data security and privacy?” IEEE Cloud Computing, vol. 5, no. 1, pp. 31–37, Jan 2018.

[13] **X. Zhou, A. Nehme, V. Jesus, Y. Wang, M. Josephs, and K. Mahbub**, “Towards blockchain-based auditing of data exchanges,” in Smart Blockchain. Cham: Springer, 2019, pp. 43–52.

[14] **P. Zhang, J. White, D. C. Schmidt, and G. Lenz**, “Chapter one - blockchain technology use cases in healthcare,” in Blockchain Technology: Platforms, Tools and Use Cases, ser. Advances in Computers, P. Raj and G. C. Deka, Eds. Elsevier, 2018, vol. 111, pp. 1 – 41.

[15] **P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom**, “Fhirchain: Applying blockchain to securely and saleably share clinical data,” Computational and Structural Biotechnology Journal, vol. 16, pp. 267 – 278, 2018.

[16] **HL7**, “Fast healthcare interoperability resources (FHIR),” online: https://www.hl7.org/fhir/overview.html, 2011.

[17] **V. Jesus**, “Towards an accountable web of personal information: the web-of-receipts,” IEEE Access,, vol. 8, 2020.

[18] **W. Diffie and M. Hellman**, “New directions in cryptography,” IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, 1976.