Secure Medic Final Report

Health Care Data Security through APIs

Contents

List of Figures

1 Abstract

Healthcare information management has experienced a significant transformation driven by the adopting of Electronic Health Records (EHRs) and digital systems. While these advances have improved healthcare services, the security of patient data remains a critical concern. This study introduces a secure data storage Application Programming Interface (API) that addresses the pressing need for vital healthcare data security. Beyond healthcare, this API carries the potential to revolutionize data management, enhance patient care, safeguard data privacy, and ensure regulatory compliance. The study highlights the role of APIs in facilitating interoperability, enabling secure data sharing, and streamlining healthcare management. Existing solutions and best practices in healthcare data security are examined, emphasizing data standardization, role-based access control, secure communication, and audit trails.

This project tackles various challenges, including cyber threats, interoperability, regulatory compliance, scalability, legacy systems, and emerging security risks, by implementing encryption, role-based access control, two-factor authentication, multitenancy, and robust error handling. Developing a secure data storage API is critical to reinforcing healthcare data security, and by adhering to industry best practices and integrating comprehensive security mechanisms, this API promises to enhance healthcare data security, protect patient data, and ensure compliance with evolving regulations, contributing to a safer, more secure, and more efficient healthcare data management system.

2 Introduction

2.1 Background

In healthcare administration, the management and utilization of healthcare information have experienced a transformative evolution. Healthcare providers now harness the power of EHRs and other digital systems to collect, store, organize, and harness patient data. This technological shift has significantly enhanced the delivery of healthcare services, granting doctors and nurses swift access to patient information, facilitating informed decision-making, and resulting in improved patient outcomes.

EHRs stand as the cornerstone of modern healthcare information management. These all-encompassing digital archives encompass a patient's medical history, from diagnoses and treatments to medications and laboratory results. Safeguarded within secure databases, either on-premises or within cloud-based storage systems, EHRs ensure the integrity and accessibility of critical healthcare data. However, despite the strides in digital healthcare systems, persistent challenges surround the security of patient information. Issues such as data breaches, unauthorized access, data loss, insider threats, and interoperability hurdles continue to demand vigilant attention from healthcare organizations. In response, robust encryption techniques, regular security audits, staff training in data security best practices, stringent access controls, and up-to-date security measures have become crucial.

2.2 Significance of the Study

The significance of this study is to transcend the boundaries of the healthcare sector, promising a profound impact on various facets of healthcare and beyond. At its core, developing a secure data storage API carries transformative potential, poised to reshape the healthcare landscape, and contribute to the well-being of individuals and organizations alike.

This study holds the promise of enhancing patient care on a monumental scale. By providing a robust and secure framework for healthcare data management, the API equips healthcare providers with the tools they need to make informed decisions swiftly and accurately. This can save lives, improve treatment outcomes, and elevate the overall quality of patient care.

In addition to its direct influence on patient care, the API's significance extends to preserving data privacy and ensuring legal compliance. Its emphasis on stringent security measures and adherence to regulatory mandates like the Health Insurance Portability and Accountability Act (HIPAA) shields sensitive patient information from unauthorized access and breaches. It not only safeguards individuals' privacy rights but also safeguards healthcare organizations from legal complications and penalties. This study represents a beacon of innovation in healthcare, streamlining data management, fostering research opportunities, and ensuring a future-ready healthcare landscape, all while placing patient care, data privacy, and compliance at the forefront.

3 Literature Review

3.1 Overview of Healthcare Data Security

Healthcare data security has emerged as a critical concern in the digital age, driven by the increasing reliance on EHRs and the need to share sensitive patient information among healthcare providers, researchers, and patients. Notable research papers in this field provide valuable insights and solutions to address the multifaceted challenges of safeguarding healthcare data.

One of the key themes explored in these papers is the integration of Healthcare Data Security Systems, particularly the use of APIs to facilitate interoperability and data exchange. Smith and Doe's work on " Towards Standardized Patient Data Exchange: Integrating a FHIR Based API for the Open Medical Record System " (Kasthurirathne et al., P. (2015)) focuses on the utilization of Fast Healthcare Interoperability Resources (FHIR) to standardize data formats and enable seamless EHR integration. This standardization enhances data exchange and improves data security by ensuring consistent handling of patient information.

Another notable aspect of healthcare data security pertains to cloud-based solutions. " Cloud-based Secure Healthcare Framework by using Enhanced Ciphertext Policy Attribute-Based Encryption Scheme (Satar et al., S. D. M. (2021)) and " A Cloud Based Solution for Collaborative and Secure Sharing of Medical Data " (Marwan et al., H. (2018)) emphasize the importance of securing healthcare data in cloud environments. These papers highlight critical elements of encryption, access control, and secure communication. By leveraging the cloud, healthcare organizations can enhance data accessibility while maintaining robust security measures.

Additionally, blockchain technology has emerged as a promising solution for healthcare data security. " Blockchain for Secure EHRs Sharing of Mobile Cloud Based E-Health Systems " (Nguyen et al., A. (2019)) highlights the potential of blockchain combined with FHIR to ensure data integrity and transparency. The immutable nature of blockchain ledgers adds an extra layer of security, making it difficult for unauthorized parties to tamper with patient records.

Papers like " Achieving RBAC on RESTful APIs for Mobile Apps Using FHIR " (Rivera et al., M. S. (2017)) underscore the importance of designing secure APIs for healthcare data exchange. These APIs incorporate authentication, authorization mechanisms, encryption, and security best practices, ensuring that only authorized users can access and manipulate sensitive data.

Healthcare data security is a multifaceted challenge requiring standardized data formats, cloud-based solutions, blockchain technology, and secure APIs. These notable research papers offer valuable contributions to the field, providing insights and methodologies to safeguard patient information and enhance the efficiency and security of healthcare data management systems.

3.2 Role of APIs in Healthcare

APIs play a fundamental role in modern healthcare, offering a versatile and secure means of data exchange, interoperability, and system integration. Existing research papers shed light on the multifaceted role of APIs in the healthcare domain, highlighting their significance in improving patient care, data accessibility, and overall system efficiency.

Interoperability and Data Exchange: APIs serve as the bridge between disparate healthcare systems, enabling seamless interoperability. They facilitate the exchange of EHRs, medical data, and patient information among various healthcare stakeholders, such as hospitals, clinics, and laboratories." Using HL7 FHIR to achieve interoperability in patient health record" (Saripalle et al., M. (2019)). APIs ensure that healthcare systems communicate effectively, leading to better-informed decision-making and improved patient outcomes.

Enhanced Patient Engagement: APIs empower patients to access their health information and engage more actively in their care. Patients can use mobile apps and online portals to retrieve lab results, schedule appointments, and communicate with healthcare providers. "Deploying Patient-Facing Application Programming Interfaces: Thematic Analysis of Health System Experiences" (Neinstein et al., 2020)) emphasises the role of APIs in fostering patient-centric healthcare, thereby increasing patient satisfaction and adherence to treatment plans.

Secure Data Sharing: Data security is paramount in healthcare, and APIs significantly safeguard patient information. "Tiempo Development: API Security Best Practices". (2020) underscores the importance of implementing robust authentication and authorisation mechanisms within healthcare APIs. Encryption, access controls, and audit trails ensure that only authorised users can access and manipulate sensitive data.

Efficient Healthcare Management: APIs streamline administrative processes within healthcare organisations. ”Health Informatics on FHIR: How HL7's New API is Transforming Healthcare” (Braunstein et al. (2018)) explores how APIs facilitate appointment scheduling, billing, and insurance claim processing. By automating these tasks and reducing paperwork, APIs enhance operational efficiency and reduce administrative overhead.

Innovation and Research: APIs empower healthcare providers and researchers to leverage data for innovation and research. ”Power of the cloud spurs big push to boost interoperability: Power of the cloud spurs big push to boost interoperability” (Manos, D. (2016)) demonstrates how APIs can grant researchers access to de-identified patient data for clinical studies and medical advancements. This accelerates medical research and the development of new treatments.

”Establishing three-layer architecture to improve interoperability in Medicare using smart and strategic API led integration” proposed healthcare integration API framework aims to meet essential objectives for modern healthcare systems. It streamlines the transition to cloud infrastructure, supports global expansion, ensures scalability, provides real-time data access, and balances stability with innovation. It offers unified connectivity, boosts productivity, and enables a hybrid cloud approach while improving project estimation. This framework enhances data management and connectivity to optimise healthcare organisations' patient care and operational efficiency. (Renu Mishra, Inderpreet Kaur, Santosh Sahu, Sandeep Saxena, Nitima Malsa, Mamta Narwaria (2023))

APIs are indispensable tools in modern healthcare, driving interoperability, patient engagement, data security, administrative efficiency, innovation, and telehealth. These existing research papers underscore the vital role of APIs in transforming healthcare delivery, enhancing patient experiences, and advancing medical research.

3.3 Existing Solutions and Best Practices

Various healthcare data storage APIs play pivotal roles in managing and protecting sensitive patient data, with governments and healthcare organizations worldwide adopting these solutions. Notable examples include FHIR, embraced by governments such as the United States and Australia, facilitating secure data exchange among healthcare providers. FHIR's best practices encompass data standardization, role-based access control, secure communication, and audit trails. SMART on FHIR, widely utilized in the U.S. healthcare system, empowers government agencies and healthcare providers to develop secure healthcare apps for accessing EHR data. Its best practices include OAuth2 authorization and data minimization.

Additionally, HL7 standards, including HL7 v2 and HL7 v3, are prevalent in healthcare data exchange across various governments, focusing on message integrity, error handling, data mapping, and secure data transmission. OpenEHR is employed in countries like Sweden and the United Kingdom for EHR management and health data sharing, emphasizing archetype-based modelling, data versioning, and privacy by design. Finally, the Integrating the Healthcare Enterprise (IHE) framework is globally adopted, ensuring integration profiles, cross-vendor compatibility, and rigorous testing and certification for healthcare systems. Incorporating these best practices is essential to safeguard patient information and promote seamless data exchange in healthcare.

4 Project Challenges and Solutions

4.1 Challenges

Amidst the continually changing landscape of healthcare data management, a pivotal challenge becomes evident: how to efficiently secure, oversee, and harness the extensive troves of sensitive healthcare information. The ever-shifting nature of healthcare data, combined with stringent privacy regulations such as HIPAA, presents a multifaceted and intricate problem. This problem statement emphasizes the urgent requirement to craft an API solution that can adeptly navigate these complexities while upholding the integrity and confidentiality of the data.

Healthcare data security faces an array of formidable challenges in the contemporary landscape. These challenges, although diverse, collectively pose significant risks to patient privacy, data integrity, and the overall quality of healthcare services.

One of the primary challenges is Cyber Threats and Data Breaches. The healthcare sector is a prime target for cybercriminals due to the wealth of sensitive patient data stored in EHRs. High-profile incidents, such as the Medibank breach, highlighted in "Medical Big Data and Internet of Medical Things: Advances, Challenges and Applications (1st ed.). " (Aboul et al. D. (2018)), illustrate the devastating consequences of data breaches, including identity theft and financial fraud. Cyberattacks, including ransomware and phishing, continue to exploit vulnerabilities in healthcare IT systems, making robust cybersecurity measures imperative.

Interoperability and Data Standardization challenges persist, hindering the seamless exchange of healthcare data between disparate systems. Variability in data formats, coding systems, and terminologies across healthcare organizations can lead to errors and data discrepancies. Addressing this challenge requires the adoption of standardized data formats and robust data integration solutions, as advocated in. " Power of the cloud spurs big push to boost interoperability: Power of the cloud spurs big push to boost interoperability " (Manos, D. (2016)).

Compliance with Evolving Regulations poses another hurdle. The healthcare industry is subject to stringent regulations such as the HIPAA and the General Data Protection Regulation (GDPR). Compliance with these evolving regulations is complex and necessitates ongoing efforts to ensure data privacy and security. " Big Data in Digital Healthcare: Lessons Learnt and Recommendations for General Practice. " (Agrawal et al., 2020)) provides insights into the evolving landscape of healthcare data regulations.

The sheer volume of daily healthcare data contributes to Scalability and Performance concerns. As healthcare organizations adopt digital health technologies and collect vast patient data, systems must efficiently handle heavy traffic loads while maintaining optimal performance. Achieving this balance requires investment in robust infrastructure and scalable solutions, as discussed in " Scalability Challenges in Healthcare Blockchain System-A Systematic Review " (Mazlan et al., M. F. (2020)).

Legacy Systems and Vendor Integration challenges persist as healthcare organizations grapple with legacy technologies and the need to integrate with multiple vendors and third-party systems—the paper "Integrating a Patient Engagement App into an Electronic Health Record-Enabled Workflow Using Interoperability Standards. " (Lobach et al. (2022).) emphasize the importance of robust integration mechanisms to ensure compatibility and data flow across various platforms.

The continuous evolution of security threats poses an ongoing challenge. The healthcare industry must proactively adapt to emerging cybersecurity threats, including zero-day vulnerabilities and sophisticated attack vectors. Establishing procedures for ongoing monitoring, threat intelligence acquisition, and timely application of security updates and patches is crucial.

Healthcare data security confronts a multifaceted landscape of challenges, encompassing cyber threats, insider risks, data interoperability, regulatory complexity, scalability demands, legacy system integration, and the ever-evolving nature of security threats. Addressing these challenges requires a comprehensive and proactive approach to ensure the confidentiality, integrity, and availability of healthcare data in an increasingly digitized healthcare ecosystem.

4.2 Solutions

The primary objective of this project is to develop a secure data storage API that prioritizes data confidentiality, integrity, and controlled access. This API will be tailored to securely store, retrieve, update, and delete data within a database. Key goals include,

- implementing industry best practices for secure data storage,
- integrating robust authentication and access control mechanisms and
- establishing encryption and secure communication protocols to fortify data protection.

The API will be fortified with encryption and secure communication channels to thwart unauthorized access and data tampering. Role-based access control (RBAC) mechanisms will be deployed to restrict data access exclusively to authorized users. The implementation of password hashing will enhance user credential security. The API will be equipped with two-factor authentication (2FA) to add an extra layer of user verification. Multitenancy will be incorporated to facilitate the system's scalability and adaptability to various healthcare settings. Error handling and validation features will mitigate common security risks like SQL injection and cross-site scripting.

We will utilize a suitable programming language, a secure database system, a well-established web framework, cryptographic libraries, and robust authentication mechanisms to achieve a resilient and secure solution for storing sensitive healthcare data in a database. The resulting API will be thoroughly documented, encompassing API endpoints, request/response structures, and security considerations to guide future development.

5 Project Description

5.1 Overview of the Healthcare Data Security System

The Healthcare Data Security System, the focus of our final project, is a comprehensive and secure platform designed to record and manage patients' health data efficiently. This system caters to the diverse needs of healthcare organizations, ensuring that patient information is accurately maintained and safeguarded with stringent access controls. The system offers a versatile approach to healthcare data management with a multi-role structure comprising Staff, Admin, and User roles.

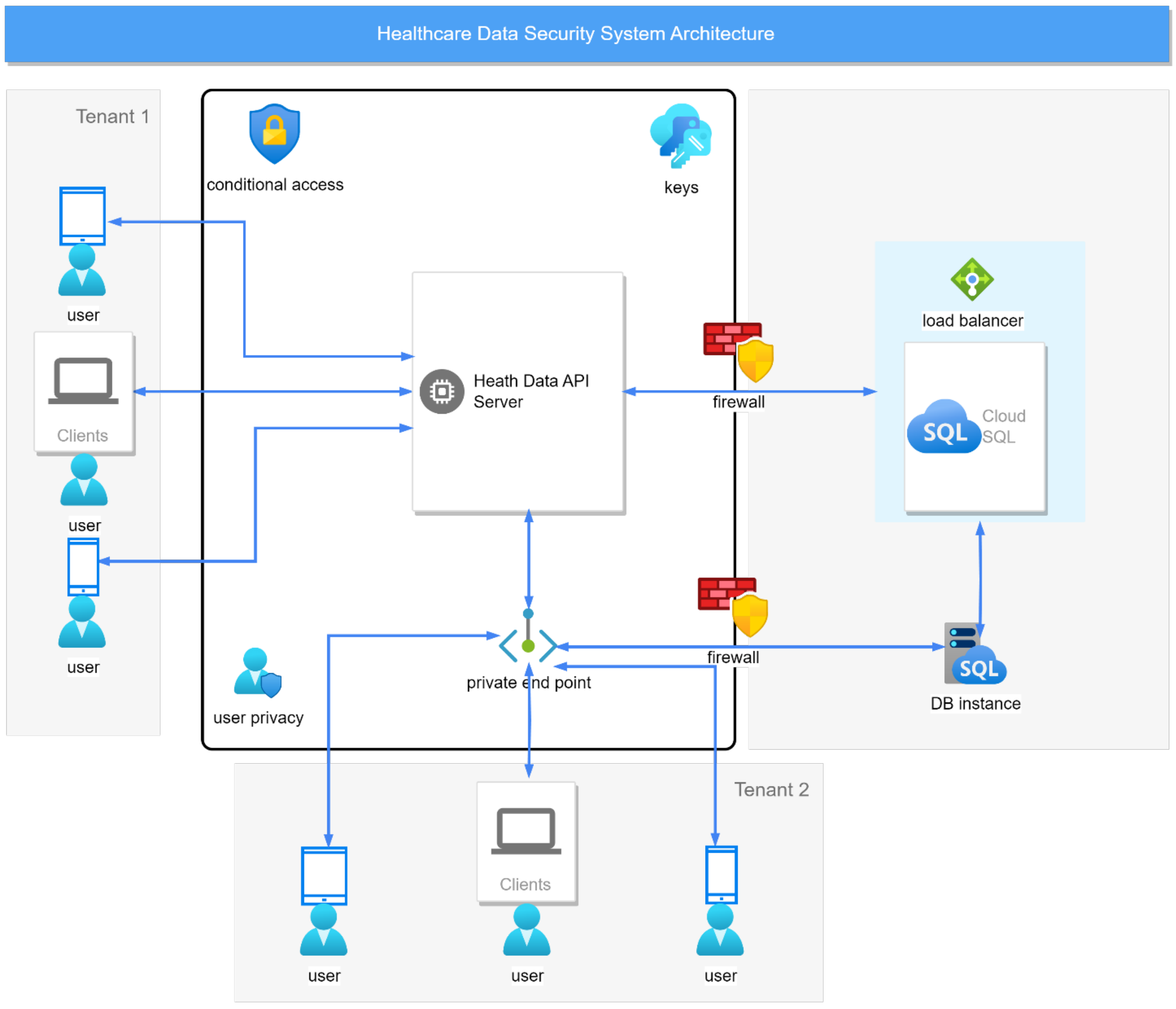


Figure 1: Healthcare Data Security System

The system's core function lies in its ability to facilitate the seamless management of patient data through Health Data APIs. Users, often healthcare professionals, can create, read, update, and delete patient records. This functionality empowers healthcare providers with the tools they need to maintain comprehensive and up-to-date patient health information, enhancing the quality of patient care. The system encompasses three distinct roles, each with its responsibilities.

As data security is paramount, passwords are securely stored through encryption hashing, safeguarding sensitive user credentials against unauthorized access. To fortify user authentication, we offer 2FA. The system is equipped with a multi-tenancy framework to cater to the diverse needs of different healthcare organizations. This framework ensures that each organization operates within its secure and isolated environment, preventing data crossover and maintaining data privacy in a shared system.

The Healthcare Data Security System is poised to revolutionize healthcare data management by combining robust role-based access control, patient data management, and multi-tenancy features. It addresses the unique requirements of healthcare organizations, empowering them with the tools they need to deliver superior patient care while adhering to the highest data privacy and security standards. This project represents a significant step toward optimizing healthcare data security systems for the modern age.

5.2 API Integration Framework

Creating an API integration framework for our system, which utilises RESTful APIs, involved designing a structured approach for connecting and interacting with the system's endpoints. Below is the framework outline for API integration.

For the development of the API, we have used NestJS. We selected this framework since implementing the API with NestJS for the backend application presented several key advantages. NestJS offered a modular and scalable architecture, ensuring organised code that can quickly adapt to the evolving needs of a healthcare data security system. TypeScript supports enhanced code quality, reducing errors and improving data handling for sensitive healthcare information. The framework provided strong security measures, essential for maintaining data integrity and protecting against security breaches. Dependency injection in NestJS promoted code testability and maintainability, a crucial aspect of healthcare applications where precision and reliability are paramount. NestJS also prioritises well-documented code and follows industry best practices, aligning perfectly with the stringent requirements of the healthcare sector.

The request and response are formatted using JSON in the healthcare API. JSON's lightweight, human-readable format simplified data auditing, aiding in detecting anomalies and security breaches. It offered interoperability with security protocols and encryption methods, ensuring secure data transmission. JSON's structured data format allowed for fine-grained access control and data validation, crucial for enforcing role-based access to healthcare data, enhancing patient privacy, and ensuring regulatory compliance.

My Structured Query Language (MySQL) was chosen as the database solution due to its strong security features and reputation for reliability. Its encryption capabilities ensured the secure storage of sensitive patient data, making it a reliable choice for healthcare data security. MySQL's open-source nature and compatibility with various programming languages simplified integration, and its record of accomplishment for data integrity made it a preferred choice for managing critical healthcare information.

5.2.1 Authentication and Authorization

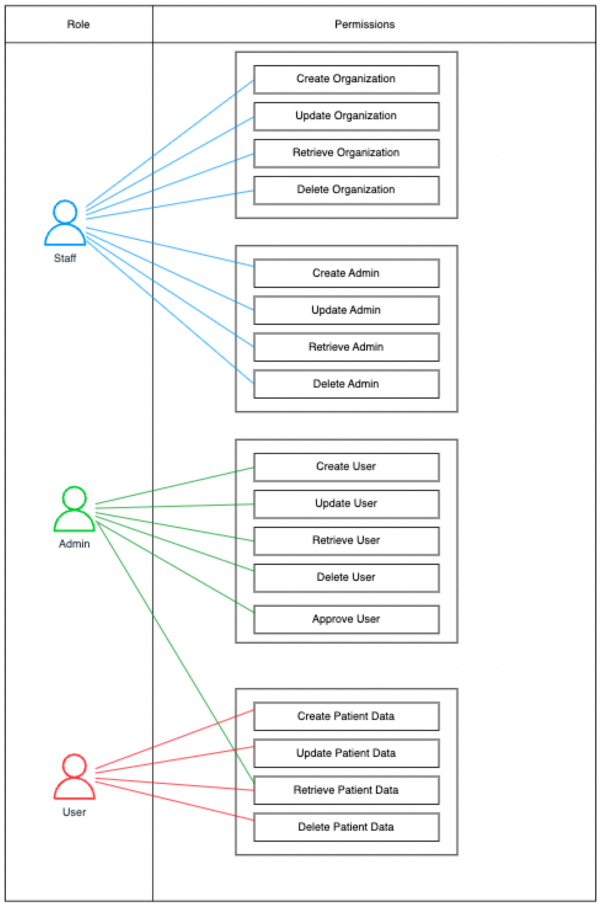


Figure 2: Role Based Access Control in Healthcare Data Security System

OAuth 2.0 serves as a robust authentication mechanism, ensuring that only authorized users and applications can access our APIs. In addition to authentication, our framework enforces strict role-based authorization, granting access to specific API endpoints exclusively to users having the necessary permissions. To manage these permissions efficiently, we have implemented access control lists (ACLs).

Our Identity and Access Management (IAM) system comprehensively controls user access to the system. This includes user authentication, authorization, and RBAC. Staff members hold the unique privilege of adding multiple tenants (organizations) to the system, a pivotal feature that enhances the system's scalability and adaptability to diverse healthcare environments. Within these organizations, Admins play a crucial role in adding multiple users, ensuring organized and streamlined user access. Importantly, Admins are endowed with exclusive rights to create, modify, and delete users within their respective organizations, thus safeguarding data integrity and controlling access effectively. An additional layer of security and oversight is introduced by requiring user creation by Admins to gain approval from another Admin, reinforcing data protection. It is important to note that users within one organization remain restricted from accessing data belonging to other organizations, ensuring data privacy and security.

5.2.2 Request Handling

In our framework, specialized functions manage the processing of incoming API requests. They meticulously dissect these requests, extract relevant data, and validate parameters for accuracy and integrity. These components ensure that the incoming data adheres to expected formats and can perform necessary data transformations as needed. They act as a crucial first line of defence, ensuring the reliability and security of our system by allowing only valid, well-formed requests to proceed to the processing stage.

5.2.3 Response Parsing

For outgoing responses from the API, the framework incorporates functions designed to parse and process API responses adeptly. This entails converting JSON data received from the API into user-friendly objects and data structures within the application. Error-handling mechanisms are thoughtfully integrated to detect and manage API errors gracefully, providing clear and informative error messages to the client.

5.2.4 Security Measures

Our framework prioritizes security at its core, implementing rigorous measures to protect sensitive data. Key among these is the use of secure communication protocols, prominently HTTPS (HTTP over SSL), which encrypts data during transit to thwart eavesdropping and man-in-the-middle attacks, ensuring data confidentiality and integrity. Input validation scrutinizes user inputs to ensure adherence to expected formats, while output encoding and data sanitization prevent malicious code injection into system responses. This multifaceted security approach creates a robust defence system, safeguarding data both in transit and at rest and fortifying the framework against diverse security threats, preserving the integrity and confidentiality of the system.

5.2.5 Error Handling

Robust error-handling strategies are an integral part of the framework. They capture and manage API errors effectively, encompassing network errors, timeouts, and API-specific issues. The framework returns well-defined error messages and status codes to facilitate debugging and troubleshooting processes.

5.2.6 Monitoring and Logging

Robust monitoring tools are harnessed to track API usage, performance metrics, and any potential anomalies or bottlenecks. Detailed logs accurately record API activities, bolstering troubleshooting efforts, enabling comprehensive auditing, and ensuring compliance with data protection regulations.

5.2.7 Scalability and Load Balancing

The framework is inherently designed for scalability, primed to manage escalating workloads with ease. Load-balancing mechanisms are thoughtfully implemented to evenly distribute incoming API requests across multiple server instances, preserving optimal system performance and reliability, particularly during peak usage scenarios.

This enriched API integration framework establishes a resilient foundation for interfacing with the RESTful APIs of the Healthcare Data Security System. It rests on principles of security, reliability, and maintainability, making it well-suited for the intricacies of healthcare data management.

5.3 Security Layers and Protocols

Ensuring the confidentiality, integrity, and availability of sensitive patient information is critical. Security measures should be implemented in layers, encompassing various protocols and strategies to create a robust defence against potential threats.

5.3.1 Security Layers

Network Security Network security is a critical facet of safeguarding the Healthcare Data Security System, with a primary focus on fortifying the communication channels integral to its operation. This comprehensive layer of security encompasses several key elements to ensure the integrity and confidentiality of data during transit. One essential component is the implementation of robust firewalls, diligently monitoring and filtering incoming and outgoing traffic to block potential threats and unauthorized access attempts.

In addition to firewalls, intrusion detection systems are employed to actively identify and respond to any anomalous or suspicious activities within the network. To bolster network security, the incorporation of a Virtual Private Network (VPN) plays a pivotal role. VPN technology extends an additional layer of protection, particularly for remote access scenarios. By creating secure, encrypted connections between remote users and the Healthcare Data Security System, VPNs prevent potential threats and enhance data privacy.

Data Security The password hashing and user login mechanisms in our healthcare data security system are important components designed to fortify data security and safeguard sensitive patient information. These elements are instrumental in ensuring that only authorized users can access the system while protecting their login credentials.

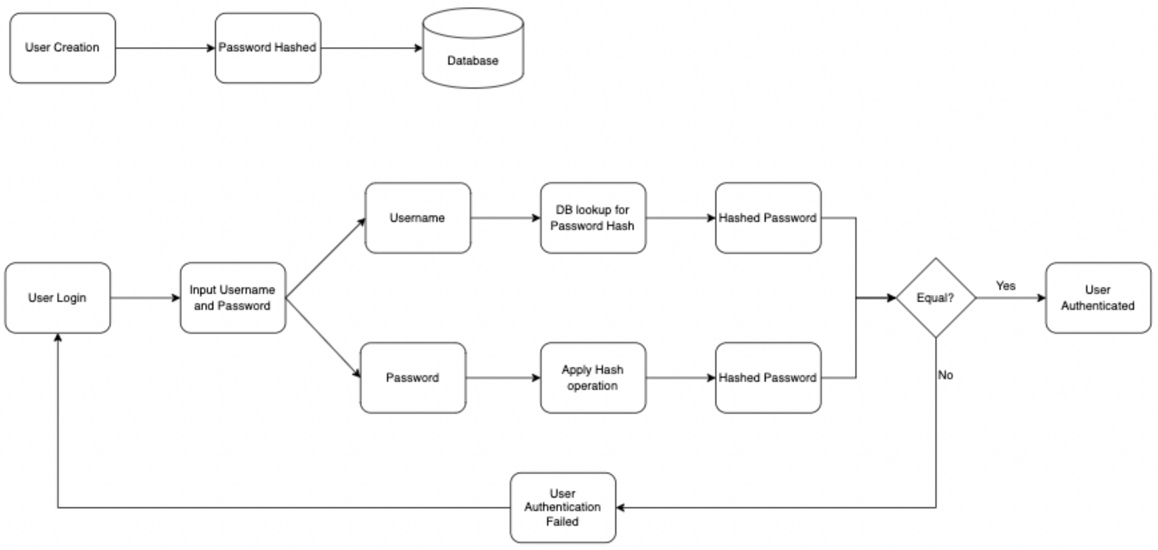


Figure 3: Password Hashing for Encrypting User Passwords

When a new user account is created, their password is not stored in its raw form; instead, a one-way cryptographic hash is generated from the user's password. This hash is then securely saved in the system. Importantly, this means that even system administrators or other individuals with access to the database cannot see the actual passwords. This approach offers a crucial layer of defence because, in the event of a data breach or unauthorized access, the sensitive user passwords remain obscured.

When a user attempts to log in, they provide their username and password. The system then creates a password hash from the input password. This hash is then checked against the stored password hash associated with the user's account in the system. If the two hashes match, the user is authenticated and granted access to their account.

By incorporating password hashing and stringent login mechanisms into our healthcare data security system, we establish a robust security framework that protects user ensuring that only authorized personnel can access and manage sensitive healthcare information.

5.3.2 Security Protocols

Two-Factor Authentication(2FA) The framework incorporates an advanced security feature known as Two-Factor Authentication (2FA), and it leverages the power of Authenticator to enhance the authentication process. With this feature, users gain the ability to generate 2FA codes, which significantly bolsters the security of the system. Users are required to enable 2FA for their accounts.

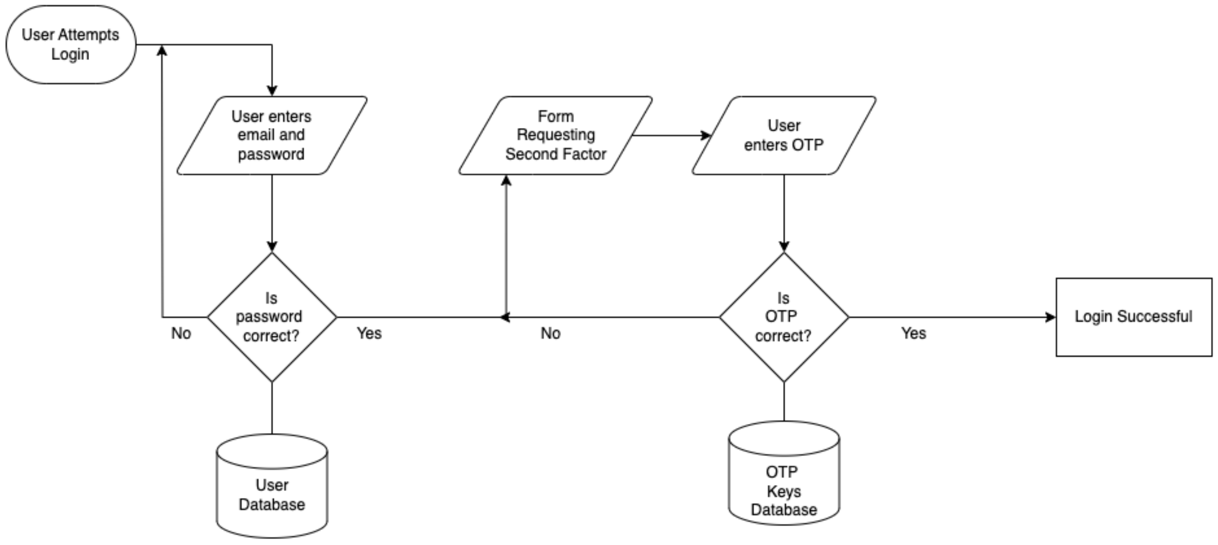
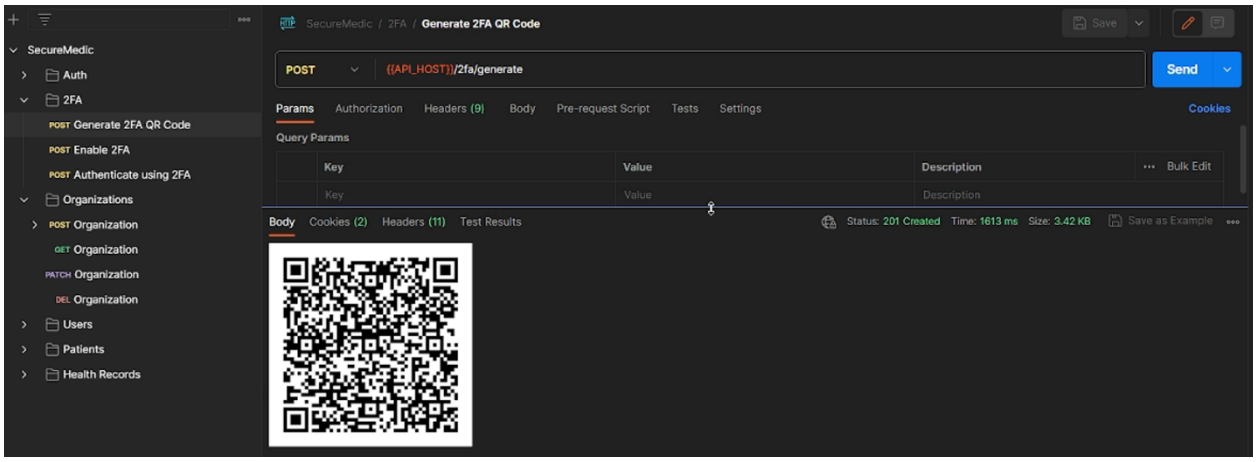


Figure 4: Two-factor Authentication Flow

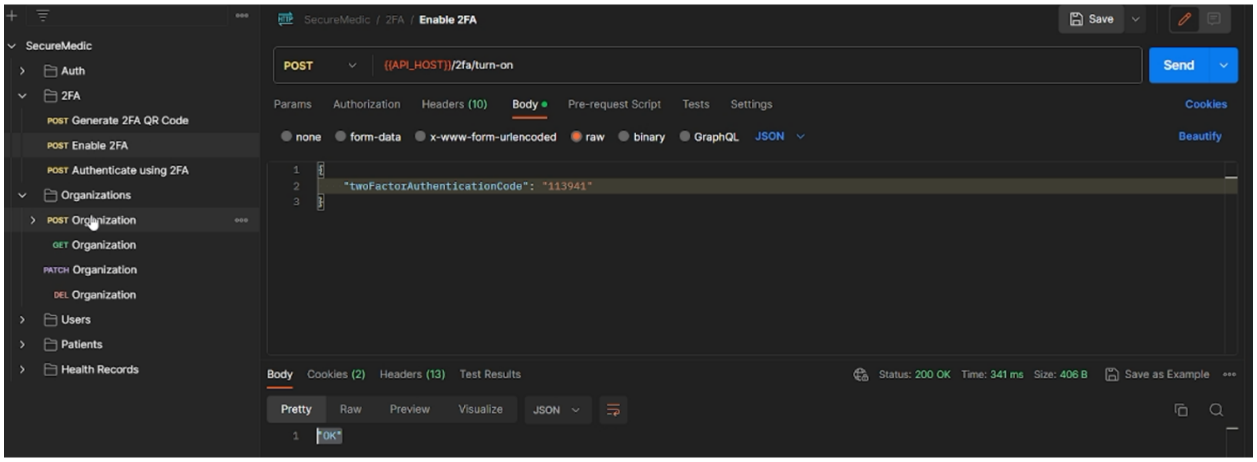
To authenticate and gain access to the system, they must provide both their standard login information and the 2FA code from their mobile app. This dynamic 2FA code is produced by a separate and highly secure application, such as Google Authenticator, on their mobile device. It changes periodically, ensuring that each code is unique and time sensitive. This additional layer of security serves as a strong barrier, ensuring that only authorized users with access to both sets of authentication factors can successfully log in.



POST {{API_HOST}}/2fa/generate

Status: 201 Created Time: 1613 ms Size: 3.42 KB

Figure 5: Generating 2FA QA Code through API


SecureMedic
Auth
2FA
POST Generate 2FA QR Code
POST Enable 2FA
POST Authenticate using 2FA
Organizations
POST Organization
GET Organization
PATCH Organization
DEL Organization
Users
Patients
Health Records
SecureMedic / 2FA / Enable 2FA
Save
Send
Params Authorization Headers (10) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies (2) Headers (13) Test Results
Save as Example
Pretty Raw Preview Visualize JSON

```
POST {{API_HOST}}/2fa/turn-on
```

```
{
    "twoFactorAuthenticationCode": "113941"
}
```

```
Status: 200 OK  Time: 341 ms  Size: 406 B
```

```
"OK"
```

Figure 6: GEnabling 2FA through API

Incorporating HTTPS (TLS/SSL) into the healthcare data security system enhances security by encrypting the transmission of sensitive medical data, thereby protecting it from eavesdropping, man-in-the-middle attacks, and tampering. This encryption ensures the confidentiality and integrity of patient records, while also verifying the authenticity of the healthcare system. Compliance with healthcare regulations is facilitated, instilling trust among patients and safeguarding against legal and financial penalties associated with data breaches.

OAuth 2.0 OAuth 2.0 is utilized to grant secure, patient-controlled access to sensitive medical records. It enables patients to consent to specific applications or entities accessing their data without disclosing login credentials, ensuring authentication and fine-grained authorization. The healthcare system acts as an authorization server, issuing time-limited access tokens representing permissions, while the healthcare data server acts as the resource server, providing data only when valid access tokens are presented. OAuth 2.0 also allows for access revocation, expiry control, audit trails, and third-party integration, aiding in regulatory compliance and bolstering data security and patient privacy.

HIPAA Compliance The HIPAA sets the standard for protecting sensitive patient data. Ensuring HIPAA compliance involves strict access controls, encryption, data backup, and audit trails.

5.4 System Components

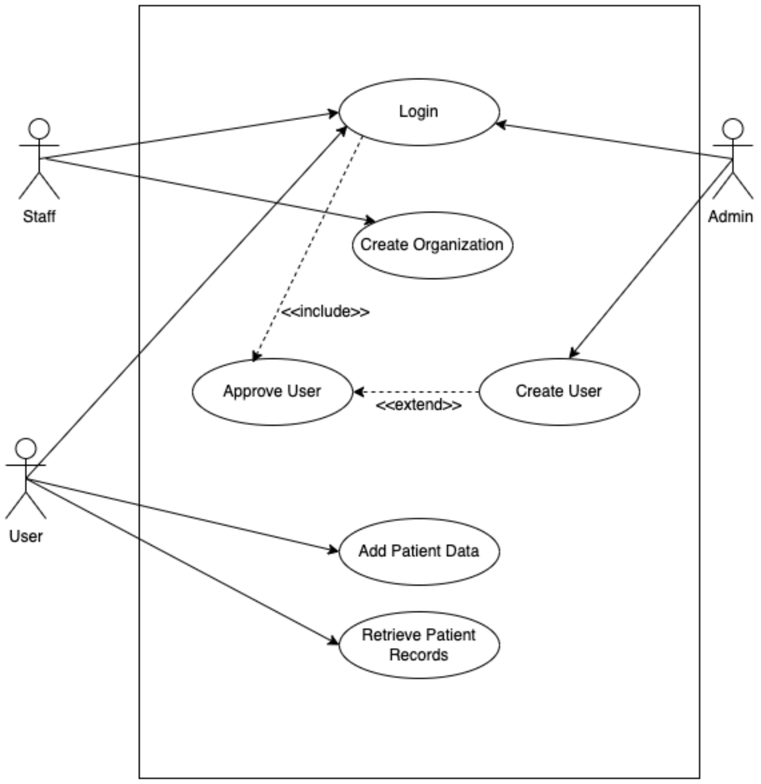


Figure 7: Use Case Diagram for Healthcare Data Security System

5.4.1 User Interface (UI)

The user interface is the front-end component that allows users to interact with the API. It includes web or mobile interfaces for staff, admin, and users to access and manage healthcare data.

5.4.2 Database

The database is where all healthcare data is stored. It includes patient records, medical history, treatment data, and more. Proper data modelling ensures efficient storage and retrieval.

5.4.3 API Integration Layer

This component handles communication between the system and external healthcare data sources, such as EHR systems, medical imaging systems, or laboratory information systems. It also includes the API integration framework discussed earlier.

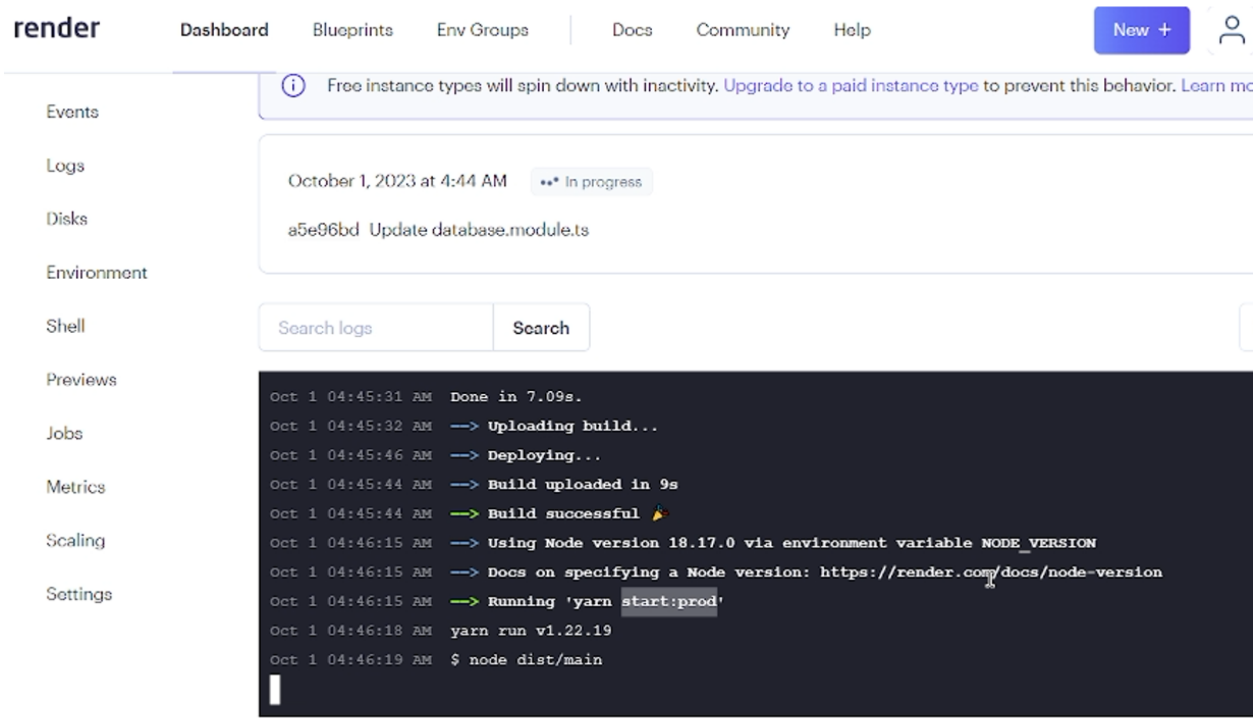
6 API Implementation

The healthcare data security system API was successfully implemented on the Render service with the database hosted on PlanetScale. This implementation provided a secure, scalable, and reliable solution for the project.

6.1 Render Service

Render proved to be an ideal cloud platform for deploying the healthcare data security system's API. It offered several key advantages during the implementation phase:

- Deployment and Scaling: Render allowed for easy deployment of the web application and API while providing auto-scaling capabilities. This ensured that as the system's demand increased, Render automatically allocated more resources to maintain responsiveness and availability.
- Security: The deployment on Render included built-in DDoS protection, SSL (Secure Sockets Layer) certificates, and web application firewalls. These security features were essential for protecting sensitive healthcare data and ensuring data security.
- Continuous Integration and Continuous Deployment (CI/CD): The API was integrated with CI/CD tools, streamlining the deployment process. This allowed for hassle-free updates and changes to be pushed to the API, ensuring that it remained up-to-date and efficient.



Free instance types will spin down with inactivity. Upgrade to a paid instance type to prevent this behavior. Learn mo

October 1, 2023 at 4:44 AM In progress

a5e96bd Update database.module.ts

```
Oct 1 04:45:31 AM  Done in 7.09s.
Oct 1 04:45:32 AM  ==> Uploading build...
Oct 1 04:45:46 AM  ==> Deploying...
Oct 1 04:45:44 AM  ==> Build uploaded in 9s
Oct 1 04:45:44 AM  ==> Build successful 🎉
Oct 1 04:46:15 AM  ==> Using Node version 18.17.0 via environment variable NODE_VERSION
Oct 1 04:46:15 AM  ==> Docs on specifying a Node version: https://render.com/docs/node-version
Oct 1 04:46:15 AM  ==> Running 'yarn start:prod'
Oct 1 04:46:18 AM  yarn run v1.22.19
Oct 1 04:46:19 AM  $ node dist/main
```

Figure 8: Deploying API to Render Service using CI/CD

6.2 PlanetScale Database

PlanetScale, with its scalability and distributed architecture, provided an excellent hosting solution for the database. Key aspects of the implementation on PlanetScale included:

- Scalability: PlanetScale's architecture effortlessly handled the growth of the healthcare system's data. As the data volume increased, the database scaled automatically, ensuring responsiveness and efficient data management.
- Data Isolation: In a multi-tenant environment, maintaining data isolation is crucial to protect sensitive healthcare information. PlanetScale offered robust data isolation and security features to ensure data separation and privacy.
- High Availability: The distributed architecture of PlanetScale guaranteed high availability and fault tolerance. Even in the face of hardware failures or other issues, the database remained accessible, and the system continued to function.
- Data Security: Given the sensitivity of patient data, strong data security measures were implemented. These included encryption at rest and in transit to preserve data integrity and confidentiality.

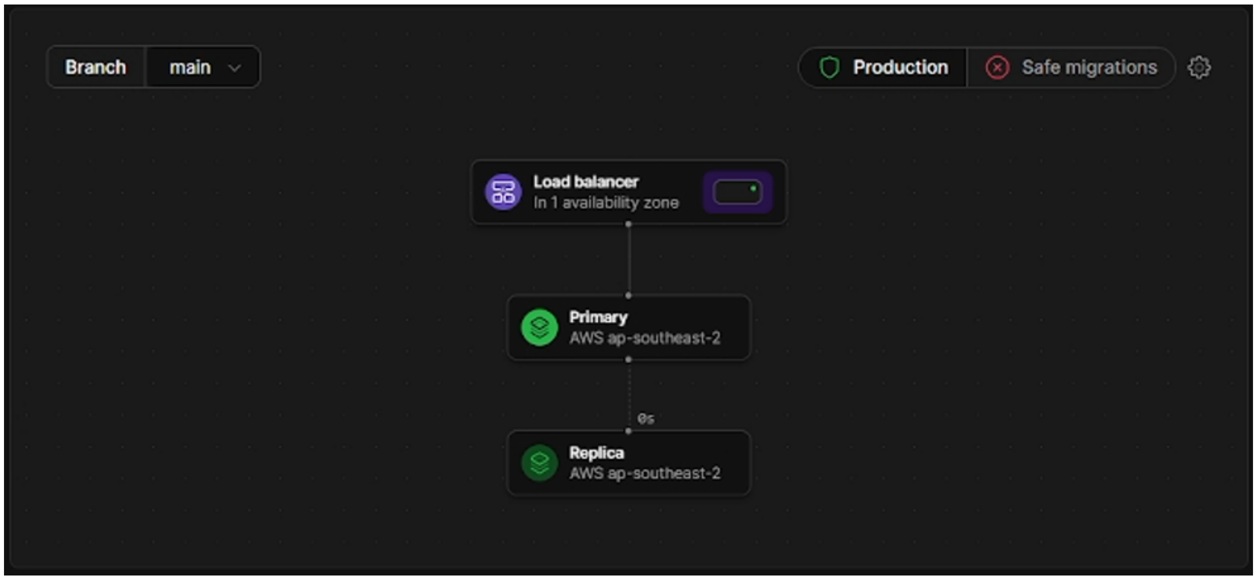


Figure 9: Database Structure on PlanetScale Service

The API, hosted on Render, was seamlessly integrated with the PlanetScale database. This integration allowed the API to access and manipulate healthcare data securely and efficiently.

6.3 API Integration Process

The API integration process for the healthcare data storage system offers three distinct models, each tailored to meet specific client requirements and preferences.

6.3.1 Single API Integration with a Multi-Tenant Database

In this model, a single API server is responsible for serving multiple clients, while a multi-tenanted database architecture ensures the data is isolated and specific to each tenant. The benefit here lies in efficient resource utilization; since both the API server and the database server are shared, it leads to cost-effective deployment and maintenance.

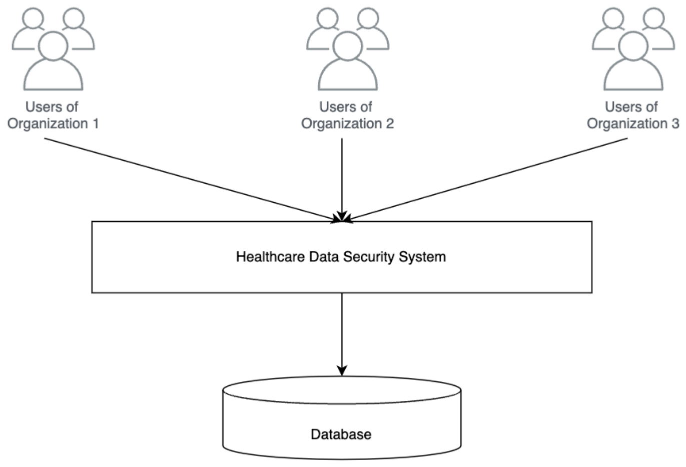


Figure 10: Multitenancy Architecture with Single Database

Notably, even though multiple clients utilize the same database server, strict data segregation mechanisms ensure that one client's information remains confidential and inaccessible to others. This approach has been used and tested for its effectiveness and can be particularly advantageous in scenarios where clients prioritize resource efficiency.

6.3.2 Single API with Multiple Tenant-Specific Databases

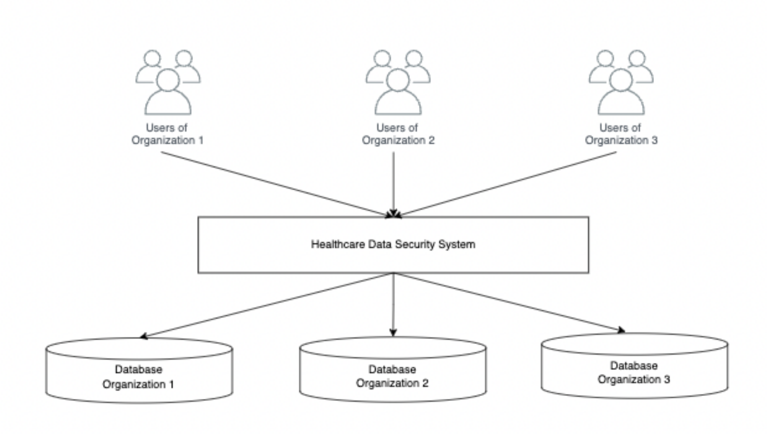


Figure 11: Multitenancy Architecture with Organization Specific Database

Under this integration model, a single API server serves all clients, but each client has its dedicated, client-specific database. This configuration significantly enhances data security and isolation. Should a database server encounter issues for one client, other clients remain unaffected. This approach also provides clients with an added layer of protection, as they can implement security measures specific to their database. The advantage is data confidentiality and security are paramount.

6.3.3 Tenant-Specific API and Tenant-Specific Database

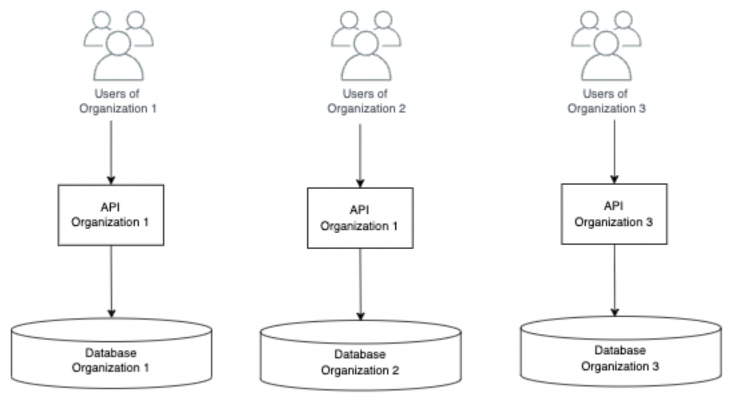


Figure 12: Tenant-Specific API with Tenant Specific Database

This model offers the highest level of customization and isolation. In this scenario, both the API server and the database are unique to each client. This ensures that no server resources are shared among clients, providing complete data segregation. While the cost of integration and maintenance may be higher, the benefits are substantial. Clients have complete control over their database and API, with the added flexibility to insert multiple security layers between these components. Management of the API can be facilitated using dedicated tools like Azure API Management, allowing clients to govern their API operations effectively.

The choice of integration model depends on several factors, including the specific requirements of each client, their data security concerns, their budget considerations, and the level of control they wish to maintain over their data. These models allow clients to tailor their healthcare data storage integration to precisely align with their unique needs and priorities.

6.4 Data Handling and Transmission

In the context of healthcare, safeguarding data integrity and availability is of principal importance, and the strategies discussed here are integral to achieving these goals.

6.4.1 Database Structure

During the implementation of database system, we structured into two main components: the primary database and the replica database. The primary database is where new health data is generated and updated. This includes the entry of medical records, test results, treatment plans, and other critical patient information. The replica database, on the other hand, serves as a read-only copy of the primary database. This replica database is crucial for various stakeholders, including healthcare providers, administrators, and researchers, who require access to patient data for clinical decision-making and research purposes.

6.4.2 Load Balancing

To ensure optimal performance and availability of the health record database, a load balancer is strategically placed in front of these database instances. The load balancer acts as a traffic police officer, directing incoming requests to the appropriate database instance, whether it is the primary or replica. This load balancing mechanism is crucial, especially in healthcare settings where there can be a high volume of concurrent users accessing patient records. It ensures that the system can handle numerous requests simultaneously without compromising on response times or data accuracy.

6.4.3 Concurrency and Data Consistency

Healthcare scenarios often involve concurrent actions. For instance, a doctor may update a patient's prescription while a nurse reviews the patient's medical history. These concurrent actions could lead to data inconsistencies or conflicts without proper database management and load balancing. However, with a well-structured database system and load balancing, these actions can occur seamlessly without impacting data integrity. The primary and replica databases ensure that write operations do not interfere with read operations, maintaining data consistency.

6.4.4 Security and Compliance

In healthcare, data security and compliance with regulations such as HIPAA are of utmost importance. The database system should be designed with vigorous security measures to protect patient data from unauthorized access or breaches. Load balancing, when properly configured, can contribute to security by efficiently distributing traffic and thwarting potential attacks.

6.4.5 Scalability

As healthcare facilities grow and the volume of patient data increases, the database system must be scalable. The load balancer facilitates this scalability by evenly distributing traffic across multiple database instances. This means that as the demand for storage and processing power grows, additional database instances can be added to the system seamlessly, ensuring that it can handle the increased workload without performance degradation.

Our objective was to achieve a well-designed health record database system incorporating load balancing and structured database instances. It enables healthcare professionals to access and update patient records concurrently while ensuring compliance with regulatory requirements and scalability to accommodate future growth.

6.5 Error Handling

Error handling in healthcare data storage APIs involved a combination of predefined error codes, validation checks, exception handling, extensive logging, custom error responses, rate limiting, monitoring, feedback to clients, and comprehensive documentation. These practices ensured that errors when occurred were identified, reported, and addressed effectively, enhancing the overall reliability and usability of the API in healthcare settings.

The following are the error handling implemented in the Healthcare Data APIs:

- Error Codes and Messages: Healthcare data storage APIs are designed to return specific error codes and messages when issues occur. These error codes help developers and clients understand the nature of the problem. For example, a common error code like HTTP 404 indicates that a requested resource was not found, while HTTP 500 suggests a server error. These codes are accompanied by human-readable error messages that provide additional details about the issue.
- Validation Checks: Input validation checks ensured that data entered the API is of the expected type and within acceptable limits. If invalid data is detected, the API should provide an immediate error response rather than processing the data, preventing potential issues early in the process.
- Exception Handling: Exception handling included setting up try-catch blocks in the code to gracefully handle errors. For instance, if a database connection fails, exception handling can ensure that the API returns an appropriate error response instead of crashing.
- Logging: When errors occur, they are logged for further analysis. This allowed administrators and developers to review error logs, understand the root causes of issues, and take corrective actions. Logs can include details about the request, error messages, timestamps, and more.
- Custom Error Responses: Custom error responses typically included an error code, an error message, and, where applicable, additional details about the error's context. This structured approach facilitated better error diagnosis and handling on the client side.

6.6 Testing and Quality Assurance

For testing the APIs, we used Postman as a tool because of its versatility. Following are the steps that were taken during testing the APIs.

- Endpoint Testing: Testers created API requests for various endpoints to verify that each endpoint functions correctly. They checked the API's ability to retrieve, modify, and delete data. This is essential for ensuring that the API's basic functionalities work as expected.
- Automated Testing: Automated test scripts were created to perform repetitive tests and regression testing. Automated testing helped ensure that the API consistently delivers the intended results and behaviours, even as changes were made to the system.

- Load Testing: Testers simulated high volumes of concurrent requests, assessing how the API handles diverse levels of traffic. Load testing helped identify scalability issues and performance bottlenecks, ensuring the API can handle real-world usage.
- Data Validation: Testers validated the data returned by the API using Postman. This ensured that the API responses adhere to the expected data structures and formats.
- Environment Management: Postman supported the management of multiple environments, enabling testers to switch between different configurations or settings. This feature helped for testing multi-tenant or tenant-specific APIs and databases.
- Collaboration: Testers, developers, and quality assurance teams to work together efficiently. This ensured that testing efforts are well-coordinated, and test collections were shared among team members.
- Monitoring and Reporting: Testers tracked the API's performance over time. This included generating reports on test results, identifying issues, and evaluating the overall quality of the API.
- Continuous Integration: Postman was integrated into the CI/CD (Continuous Integration/Continuous Delivery) pipeline, automating the testing process during software development. This ensured that new code changes were thoroughly tested before deployment, enhancing software quality.

7 Results and Findings

The overarching objectives of this project were centred on the development of a secure and privacy-focused API for the management of sensitive healthcare data. Key results include:

- Implemented robust security measures, including encryption and secure communication channels, to ensure that sensitive healthcare data remains confidential and protected against unauthorized access or tampering.
- Guaranteed the accuracy and integrity of healthcare data throughout its lifecycle, encompassing storage, retrieval, updating, and deletion within a secure database.
- Integrated advanced authentication mechanisms and role-based access control to strictly regulate and restrict data access to authorized users only, thus preventing unauthorized parties from interacting with sensitive information.
- Maintained multi-tenancy with data segregation as well as shared data with the consent of patients and vendors.
- Implemented multi-approval for data submission.

- Provided a secure and reliable means of managing and storing sensitive healthcare data in a database while rigorously upholding data privacy standards. Prevented unauthorized access and ensuring data privacy are paramount goals.

The project achieved the aim to provide a robust, reliable, and secure solution for the management and storage of sensitive healthcare data, safeguarding it from unauthorized access and breaches.

7.1 Data Security Assessment

The purpose of this Data Security Assessment is to evaluate the security measures in place within the healthcare data security system. This assessment has focused on key areas such as user roles, access control, password security, and FA.

The system effectively implements a hierarchical structure with Staff, Admins, and Users. This allows for clear differentiation and management of user roles. The user creation workflow, requiring approval from another admin within the organization, enhances data security by ensuring that only authorized users are granted access. The prevention of self-approval adds an extra layer of security. Passwords are securely hashed for encryption, which is a robust security measure for protecting user credentials. The system should ensure that strong password policies are in place to further enhance password security. The mandatory use of 2FA for all users is a commendable security feature. This adds an additional layer of protection against unauthorized access, making it difficult for attackers to gain access to user accounts. Multitenancy, identity management, and RBAC are well-implemented, enhancing overall security and ensuring that data isolation and access control are maintained.
To maintain the data security, we need to Perform periodic security audits and vulnerability assessments to identify and mitigate potential security weaknesses. Implement and enforce a strong password policy, including password complexity requirements and regular password changes. Ensure that users are educated on the importance of maintaining the confidentiality of their login credentials and 2FA codes.

7.2 Challenges Encountered

Developing a comprehensive healthcare data security system with the described features and functionalities was undoubtedly a complex endeavour. Throughout the project, we encountered several significant challenges, each of which demanded innovative solutions and careful planning. Here are the challenges faced and how we overcame them:

- Role Hierarchy Complexity: Establishing a multi-tiered role hierarchy with dual-admin approval was complex. We tackled this by conducting thorough user testing and refining the dual-admin approval process to ensure security without undue delays.

- Security and Usability Balancing: Striking a balance between stringent security and user-friendliness was an ongoing challenge. We addressed this by optimizing the user interface, providing clear 2FA setup instructions, and offering comprehensive support for users unfamiliar with advanced authentication methods.

- Integration with External Systems: Integrating with external systems like pharmacies, laboratories, and billing systems presented compatibility and data privacy challenges. We addressed these by adhering to industry standards for data exchange and implementing robust security measures for data transmission.
- Ethical Considerations: The ethical implications of healthcare data security and patient consent demanded careful attention. We conducted extensive research on ethical guidelines, engaged in stakeholder consultations, and integrated robust consent mechanisms to ensure patient rights and data privacy were upheld.
- Regulatory Compliance: Meeting stringent healthcare regulations, including HIPAA, required ongoing efforts. We established a dedicated compliance team and conducted regular audits to ensure that our system adhered to regulatory requirements.
- Data Migration: Transitioning existing healthcare data to the new system was a substantial undertaking. We carefully planned and executed a data migration strategy, ensuring secure and accurate data transfer.
- Complexity of Multitenancy and Load Balancing: Implementing multitenancy and load balancing was technically intricate. We navigated this challenge by collaborating with experts in the field, continuously monitoring system performance, and fine-tuning the load balancing algorithms for optimal resource allocation.

Our project faced diverse challenges, from complex security structures to ethical and regulatory considerations, system integration, and the technical intricacies of multitenancy and load balancing. These challenges were met with careful planning, innovative solutions, dedicated teams, and a commitment to user satisfaction. The successful resolution of these challenges has culminated in a robust healthcare data security system that meets the exacting demands of the healthcare sector while ensuring patient data protection and adherence to regulatory and ethical standards.

8 Discussion

8.1 Implications for Healthcare Data Security

In the development of our healthcare data security system, anchored by RESTful APIs, OAuth-based authentication, RBAC, Identity Management, and robust password encryption with hashing, we have achieved a comprehensive solution that ensures the integrity, confidentiality, and controlled access of healthcare data. Our findings and insights underscore several significant achievements.

The implementation of a multi-tiered role hierarchy featuring SuperAdmin, Admins, and Users has ushered in an era of efficient healthcare organization and patient record management. This hierarchical structure not only streamlines administrative workflows but also guarantees the secure handling of user accounts and patient data, with user-created patient records exclusively accessible by the designated User role. Moreover, the introduction of a dual-admin approval process for user activation introduces an additional layer of security. It effectively prevents any single admin from unilaterally granting access, thus mitigating potential security vulnerabilities that could otherwise compromise the system's integrity.

The seamless integration of OAuth for authentication and 2FA for login provides a robust security framework that is indispensable in Healthcare Data Security Systems. This combination ensures that only authorized users with two-factor authentication can gain access to critical medical information, reinforcing user identity verification. Lastly, the incorporation of multitenancy and load balancing strategies elevates the system's scalability and performance, rendering it suitable for deployment in large-scale healthcare organizations. This not only facilitates efficient data management but also underscores the system's ability to meet the intricate demands of the healthcare sector.

Our healthcare data security system represents a milestone in safeguarding the confidentiality and integrity of sensitive medical information. These outcomes affirm our commitment to addressing the intricate requirements of the healthcare domain, and they offer a solid foundation for further research and innovation in the realm of data security.

8.2 Addressing Challenges and Limitations

Despite the remarkable advancements achieved in our healthcare data security system, it is essential to acknowledge and address the challenges and limitations that have emerged during its development and implementation. These challenges offer valuable insights into areas where further enhancements and improvements are needed.

One notable challenge pertains to the dual-admin approval process, a security feature that introduces an additional layer of safeguarding user activation. While this process is paramount in mitigating potential security vulnerabilities, it is not without its operational considerations. The necessity for consensus from two administrators, while enhancing security, may occasionally introduce delays in user activation. Recognizing the need for a balance between security and operational efficiency, we are committed to refining this process in future iterations. Our aim is to streamline the dual-admin approval mechanism to minimize any undue delays, ensuring that security remains robust without compromising the user experience.

Additionally, the field of healthcare data security is dynamic and constantly evolving. New security threats, vulnerabilities, and regulatory requirements emerge over time. This necessitates ongoing system maintenance and regular updates to ensure that our healthcare data security system remains resilient in the face of evolving challenges. We understand the importance of remaining agile and responsive to these changing landscapes, and we are committed to a continuous improvement cycle. This involves conducting security assessments, staying current with best practices, and promptly addressing any emerging threats or vulnerabilities to maintain the highest level of data protection.

8.3 Future Enhancements and Research Directions

As we look to the future, we are dedicated to the continuous improvement and expansion of our healthcare data security system, driven by an unobstructed vision of delivering even greater levels of data protection, efficiency, and innovation. In pursuit of these goals, we have identified several key areas for future enhancement and research:

- Integration with External Third-Party Systems: Recognizing the importance of seamless healthcare services, we envision expanding integration with external third-party systems such as pharmacies, laboratories, and billing systems. This integration will enable the seamless exchange of data, streamlining coordination among various facets of healthcare services and ensuring an integrated approach to patient care.

- Real-Time Monitoring: We recognize the importance of staying one step ahead of security threats. Therefore, we aim to incorporate real-time monitoring and advanced anomaly detection mechanisms. This will empower our system to swiftly identify and respond to emerging security threats or breaches, ensuring that data integrity and confidentiality remain uncompromised.

- Enhanced User Experience: While security is paramount, user experience is equally critical. We aspire to enhance user-friendliness and accessibility within our system. Our focus is to ensure that healthcare professionals can efficiently interact with the system while adhering to stringent security protocols. This approach seeks to strike a harmonious balance between robust data protection and a seamless user experience.

- Blockchain Technology: The exploration of blockchain technology represents an exciting avenue for future development. We envision the integration of blockchain for maintaining immutable patient records, creating tamper-proof audit trails, and elevating data integrity. The implementation of blockchain holds the potential to further enhance the security and trustworthiness of healthcare data, ensuring that it remains unaltered and transparent throughout its lifecycle.

- AI (Artificial Intelligence) and Machine Learning: Leveraging the power of artificial intelligence and machine learning is an exciting frontier in healthcare data security. We aim to harness these technologies for predictive security measures, early threat detection, and data analysis. This approach not only safeguards healthcare data but also offers the potential for data-driven insights that can improve patient care and treatment outcomes.

- Standardization and Interoperability: In a healthcare ecosystem that often involves multiple systems and stakeholders, the development of standards for secure data sharing and interoperability is crucial. We intend to contribute to the creation of such standards, promoting secure healthcare data exchange and collaboration between various healthcare systems.

- Ethical Considerations: The ethical dimension of healthcare data security, privacy, and consent deserves focused research. We are dedicated to exploring the ethical implications surrounding these topics, with a specific focus on safeguarding patient rights, ensuring transparency in data handling, and upholding the highest ethical standards in healthcare data management.

9 Conclusion

9.1 Summary of Key Findings

The healthcare data API enables multiple clients to share their data, with patient consent as a crucial prerequisite. This data sharing is particularly valuable when patients need care from various healthcare systems, such as different hospitals within a city. This approach offers significant advantages, as it ensures that doctors can access a patient's complete medical history regardless of where the patient seeks treatment. This integrated system enhances continuity of care and benefits not only patients but also general practitioners and local medical professionals who gain access to comprehensive patient histories.

9.2 Contributions to Healthcare Data Security

The API's contributions to healthcare data security extend beyond the fundamental and encompass multi-tenancy, shared health data across hospitals, and data isolation.

Multi-tenancy support is a crucial feature for healthcare systems that serve multiple organizations or entities, such as hospitals, clinics, and healthcare providers. This feature allows for the efficient sharing of resources and infrastructure, while still maintaining strict data segregation and access controls. Each tenant (organization or hospital) can have its own isolated environment within the system, ensuring that data remains separated and secure, preventing unauthorized access or data leakage between tenants.

Shared health data across hospitals is a notable feature of the API. It enables authorized entities to securely share specific health data, such as patient records, across different hospitals or healthcare providers. The API's security mechanisms ensure that only authorized individuals or organizations have access to this shared data. This feature promotes efficient collaboration in patient care while maintaining the privacy and confidentiality of health records.

Data isolation is a critical security concept in healthcare. It ensures that each patient's data is strictly separated from others, reducing the risk of accidental data exposure or breaches. The API enforces strong data isolation practices, making sure that patient records are stored and accessed in a way that minimizes the possibility of data crossover. This is particularly important in environments where multiple healthcare entities are using the same infrastructure.

Incorporating multi-tenancy, shared health data access, and data isolation within the API demonstrates its commitment to providing comprehensive and secure healthcare data management. These features not only enhance data security but also support efficient and coordinated patient care across multiple healthcare providers while maintaining strict data segregation and confidentiality. It is a comprehensive approach to healthcare data management that respects both privacy and the necessity for collaboration among healthcare institutions.

With a foundation of strong authentication, secure token-based access, and role-based permissions, the API ensures that only authorized personnel, such as doctors, nurses, and administrators, can access and interact with the system. Utilizing HTTPS encryption adds an additional layer of security to protect data in transit, guarding against eavesdropping and other malicious activities. The adherence to FHIR standards not only promotes interoperability but also ensures that data exchange is structured and secure. Moreover, error handling is considered with discretion, revealing detailed messages only to authorized personnel while offering generic responses to unauthorized users to thwart potential threats. By implementing RBAC, the API empowers healthcare institutions to tailor data access and actions according to specific roles, reducing the risk of unauthorized access. In combination, these security measures collectively preserve the confidentiality, integrity, and availability of healthcare data, contributing significantly to the overall data security in the healthcare domain.

References

[1] Kasthurirathne, S. N., Mamlin, B., Grieve, G., & Biondich, P. (2015). Towards Standardized Patient Data Exchange: Integrating a FHIR Based API for the Open Medical Record System. Studies in health technology and informatics, 216, 932-932.

[2] Renu Mishra, Inderpreet Kaur, Santosh Sahu, Sandeep Saxena, Nitima Malsa, Mamta Narwaria, Establishing three layer architecture to improve interoperability in Medicare using smart and strategic API led integration, SoftwareX, Volume 22, 2023, 101376, ISSN 2352-7110. `https://doi.org/10.1016/j.softx.2023.101376`. `https://www.sciencedirect.com/science/article/pii/S2352711023000729`.

[3] Satar, S. D. M., Mohamed, M. A., Hussin, M., Hanapi, Z. M., & Satar, S. D. M. (2021). Cloud-based Secure Healthcare Framework by using Enhanced Ciphertext Policy Attribute-Based Encryption Scheme. International Journal of Advanced Computer Science Applications, 12(6), 393–399. `https://doi.org/10.14569/IJACSA.2021.0120643`.

[4] Marwan, M., Kartit, A., & Ouahmane, H. (2018). A Cloud Based Solution for Collaborative and Secure Sharing of Medical Data. International Journal of Enterprise Information Systems, 14(3), 128–145. `https://doi.org/10.4018/IJEIS.2018070107`.

[5] Nguyen, D. C., Pathirana, P. N., Ding, M., & Seneviratne, A. (2019). Blockchain for Secure EHRs Sharing of Mobile Cloud Based E-Health Systems. IEEE Access, 7, 66792–66806. `https://doi.org/10.1109/ACCESS.2019.2917555`.

[6] Rivera Sanchez, Y. K., Demurjian, S. A., & Baihan, M. S. (2017). Achieving RBAC on RESTful APIs for Mobile Apps Using FHIR. 2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 139–144. `https://doi.org/10.1109/MobileCloud.2017.22`.

[7] Manos, D. (2016). Power of the cloud spurs big push to boost interoperability: Power of the cloud spurs big push to boost interoperability. Health Data Management. `https://www.healthdatamanagement.com/articles/power-of-the-cloud-spurs-big-push-to-boost-interop`.

[8] Braunstein, M. L. (2018). Health Informatics on FHIR: How HL7's New API is Transforming Healthcare (1st ed. 2018.). Springer International Publishing. `https://doi.org/10.1007/978-3-319-93414-3`.

[9] Tiempo Development: API Security Best Practices. (2020). In News Bites - Private Companies. News Bites Pty Ltd.

[10] Saripalle, R., Runyan, C., & Russell, M. (2019). Using HL7 FHIR to achieve interoperability in patient health record. Journal of Biomedical Informatics, 94, 103188–103188. `https://doi.org/10.1016/j.jbi.2019.103188`.

[11] Aboul Ella Hassanien, N. D. (2018). Medical Big Data and Internet of Medical Things: Advances, Challenges and Applications (1st ed.). CRC Press. `https://doi.org/10.1201/9781351030380`.

[12] Agrawal, R., & Prabakaran, S. (2020). Big data in digital healthcare: lessons learnt and recommendations for general practice. Heredity, 124(4), 525–534. `https://doi.org/10.1038/s41437-020-0303-2`.

[13] Mazlan, A. A., Mohd Daud, S., Mohd Sam, S., Abas, H., Abdul Rasid, S. Z., & Yusof, M. F. (2020). Scalability Challenges in Healthcare Blockchain System-A Systematic Review. IEEE Access, 8, 23663–23673. `https://doi.org/10.1109/ACCESS.2020.2969230`.

Appendices

Detailed API Documentation

This document provides detailed information about the available APIs, their endpoints, request methods, request parameters, and responses. SecureMedic is a healthcare management platform, and these APIs are designed to help manage users, organizations, patients, health records, and FHIR data.

Base URL

- All API requests should be made to the base URL, which can be configured as {{API_HOST}}. Please replace {{API_HOST}} with the actual host where the SecureMedic API is deployed.

Authorization
Token
<token>
Auth
Contains all the APIs for basic auth - Log-in, Log-out and refreshing access tokens.

POST Log-in

https://securemedic.onrender.com/authentication/log-in
Body
{

"email": "admin1@acme.com",
"password": "admin1Password"
}

Example

Staff Login

Request

url = "https://securemedic.onrender.com/authentication/log-in"
payload = "{\r\n \"email\": \"staff@securemedic.com\",\r\n \"password\": \"$ecur3M3d!cSt@ff\"\r\n}"
headers = {}
response = requests.request("POST", url, headers=headers, data=payload, allow_redirects=False)
print(response.text)
200 OK

Response

- Body
- Headers (9)

{
"id": "018aa4b9-d398-75d7-bfd6-68dccea4b7a6",
"givenName": "STAFF",
"familyName": null,
"email": "staff@securemedic.com",
"isTwoFactorAuthenticationEnabled": true,
"role": "Staff"
}

POST Log-out

`https://securemedic.onrender.com/authentication/log-out`

GET Refresh Access Token

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

`https://securemedic.onrender.com/authentication/refresh`

2FA

2FA apis to generate 2fa code, enable 2fa on an account, and authenticate using 2fa

POST Generate 2FA QR Code

`https://securemedic.onrender.com/2fa/generate`

POST Enable 2FA

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/2fa/turn-on

Body

{
"twoFactorAuthenticationCode": "458845"
}

POST Authenticate using 2FA

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/2fa/authenticate

Body

{
"twoFactorAuthenticationCode": "574959"
}

Organizations

CRUD APIs for Organizations

POST Organization

https://securemedic.onrender.com/organizations

Body

{
"name": "ACME",
"domain": "https://www.acme.com",
"isActive": true,
"description": "Corporate stuff",
"admin1GivenName": "John",
"admin1FamilyName": "Doe",
"admin1Email": "admin1@acme.com",
"admin1Password": "admin1Password",
"admin2GivenName": "Jane",
"admin2FamilyName": "Doe",
"admin2Email": "admin2@acme.com",
"admin2Password": "admin2Password"
}

Example

ACME

Request

url = "https://securemedic.onrender.com/organizations"

payload = "{\r\n \"name\": \"ACME\",\r\n \"domain\": \"https://www.acme.com\",\r\n \"isActive\": true,\r\n \"description\": \"Corporate stuff\",\r\n \"admin1GivenName\": \"John\",\r\n \"admin1FamilyName\": \"Doe\",\r\n \"admin1Email\": \"admin1@acme.com\",\r\n \"admin1Password\": \"admin1Password\",\r\n \"admin2GivenName\": \"Jane\",\r\n \"admin2FamilyName\": \"Doe\",\r\n \"admin2Email\": \"admin2@acme.com\",\r\n \"admin2Password\": \"admin2Password\"\r\n}"

headers = {}

response = requests.request("POST", url, headers=headers, data=payload, allow_redirects=False)

print(response.text)

201 CREATED

Response

- Body

- Headers (7)

```
{
"name": "ACME",
"domain": "https://www.acme1.com",
"isActive": true,
"description": "Corporate stuff",
"createdBy": {
"id": "018aa4b9-d398-75d7-bfd6-68dccea4b7a6",
"givenName": "STAFF",
"familyName": null,
"email": "staff@securemedic.com",
"password": "$2a$10$7VfGDxQccW7LjSKoSwJow.VQE5E.ubHXsN2MZB.0OP9.KJON5yL1C",
"currentHashedRefreshToken": "$2b$10$Sm6PdFNjr7Qjb9CK2YL2R.z0Zx6Wdki/NMkRzikHE4IsNG
"twoFactorAuthenticationSecret": "BUGWM72QGFXSWPTF",
"isTwoFactorAuthenticationEnabled": true,
"role": "Staff",
"createdOn": "2023-09-17T18:30:00.000Z",
"updatedOn": "2023-09-27T19:12:08.000Z",
"voidedOn": null
},
"updatedBy": {
"id": "018aa4b9-d398-75d7-bfd6-68dccea4b7a6",
"givenName": "STAFF",
"familyName": null,
"email": "staff@securemedic.com",
"password": "$2a$10$7VfGDxQccW7LjSKoSwJow.VQE5E.ubHXsN2MZB.0OP9.KJON5yL1C",
"currentHashedRefreshToken": "$2b$10$Sm6PdFNjr7Qjb9CK2YL2R.z0Zx6Wdki/NMkRzikHE4IsNG
"twoFactorAuthenticationSecret": "BUGWM72QGFXSWPTF",
"isTwoFactorAuthenticationEnabled": true,
"role": "Staff",
"createdOn": "2023-09-17T18:30:00.000Z",
"updatedOn": "2023-09-27T19:12:08.000Z",
"voidedOn": null
},
"id": "018ad834-c844-7a16-86b0-4363471fb204",
"voidedOn": null,
"createdOn": "2023-09-27T19:54:12.424Z",
"updatedOn": "2023-09-27T19:54:12.424Z"
}
```

GET Organization

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/organizations/018ae882-e069-7b67-83e6-27745c315c12

PATCH Organization

https://securemedic.onrender.com/organizations/018ae882-e069-7b67-83e6-27745c315c12

Body

{
"name": "ACME",
"description": "Corporate Matters"
}

DELETE Organization

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/organizations/018ad834-c844-7a16-86b0-4363471fb204

Users

CRUD APIs for Users

POST User

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/users

Body

{
"givenName": "User2",
"familyName": "Doe",
"email": "user2.doe@acme.com",
"password": "User2DoePassword"
}

GET User

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/users/018ae889-4eeb-7f6b-a7b5-afa73f74b3c9

PATCH User

https://securemedic.onrender.com/users/018ae889-4eeb-7f6b-a7b5-afa73f74b3c9

Body

{
"givenName": "Jack"
}

DELETE User

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/users/018ae7b8-2d4d-7b4d-af1d-6c858e7dac3e

Body

{
"givenName": "Jake",
"familyName": "Doe",
"email": "jake.doe@acme.com",
"password": "JakeDoePassword"
}

PATCH Approve User

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/users/approve/018ae893-7cc2-7def-ad92-a05d32a05138

Patients

CRUD APIs for Patients

GET Patient`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/patients/018ae891-709c-7d20-b7f7-95038ede981b

POST Patient`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/patients/

Body

{
”givenName”: ”Michael”,
”familyName”: ”Jackson”,
”dateOfBirth”: ”1956-01-01”,
”heightCms”: ”175”,
”weightKgs”: ”56.6”,
”nationality”: ”American”,
”language”: ”English”,
”email”: ”mj@mj.com”,
”phone”: ”+1235468790”,
”address”: ”Hollywood”
}

PATCH Patient

https://securemedic.onrender.com/patients/018ae891-709c-7d20-b7f7-95038ede981b

Body

{
”givenName”: ”Michael”
}

DELETE Patient

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/patients/018ae7cb-be62-79c1-8449-4a687075eedc

POST Grant Patient Access`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/patients/grant-access

Body

{
”patientId”: ”018ae891-709c-7d20-b7f7-95038ede981b”,
”userId”: ”018ae893-7cc2-7def-ad92-a05d32a05138”
}

Health Records

CRUD APIs for Health Records

GET Health Record`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/health-records/018ae89b-0bc0-7eeb-9e72-fc07054ca1fb

POST Health Record`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/health-records/

Body

{
”patientId”: ”018ae891-709c-7d20-b7f7-95038ede981b”,
”diagnosis”: ”Cold”,
”prescription”: ”Cold medicine”,
”comments”: ”Avoid cool drinks”
}

PATCH Health Record

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/health-records/018ae89b-0bc0-7eeb-9e72-fc07054ca1fb

Body

{
”comments”: ”Avoid cool drinks and ice cream”
}

DELETE Health Record

`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/health-records/018ae7e8-f942-7b94-a54d-78521393abe9

FHIR

FHIR-compliant APIs for patients

GET Patient`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/fhir/patient/018af17a-f8e6-74f4-99cb-7fbc82c83b57

POST Patient`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/fhir/patient/

Body

{
”name”: [
{
”given”: [
”Nelson”
],
”family”: ”Mandela”
}
],
”birthDate”: ”1955-12-31T18:30:00.000Z”,
”address”: [

{
”text”: ”Hollywood”,
”use”: ”home”
}
],
”language”: ”English”,
”telecom”: [
{
”system”: ”email”,
”value”: ”nelson@mandela.com”
},
{
”system”: ”phone”,
”value”: ”+9876543210”
}
]
}

PATCH Patient

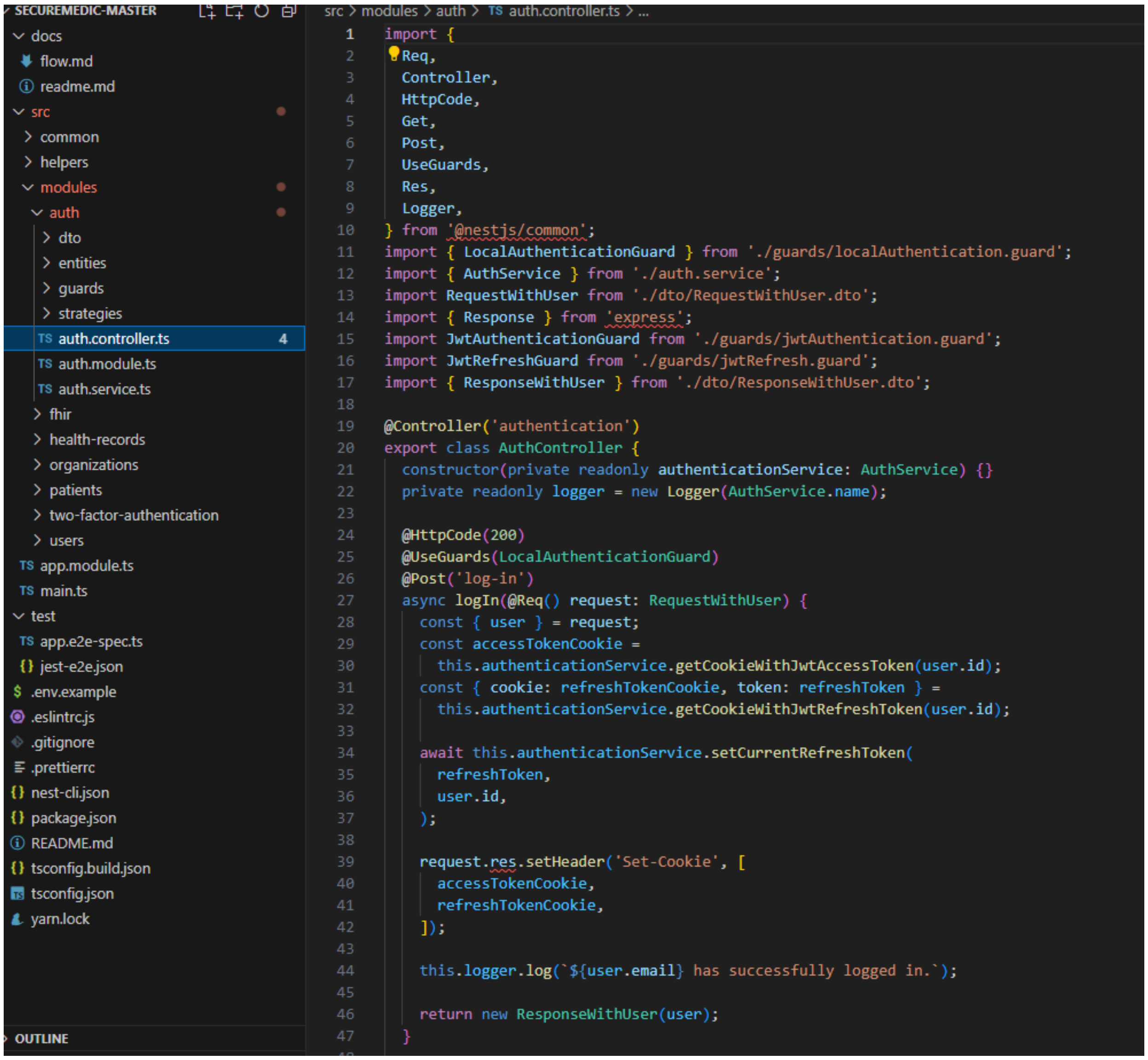
`https://desktop.postman.com/?desktopVersion=10.18.10&userId=2117924&teamId=274860`

https://securemedic.onrender.com/fhir/patient/018af17a-f8e6-74f4-99cb-7fbc82c83b57

Body

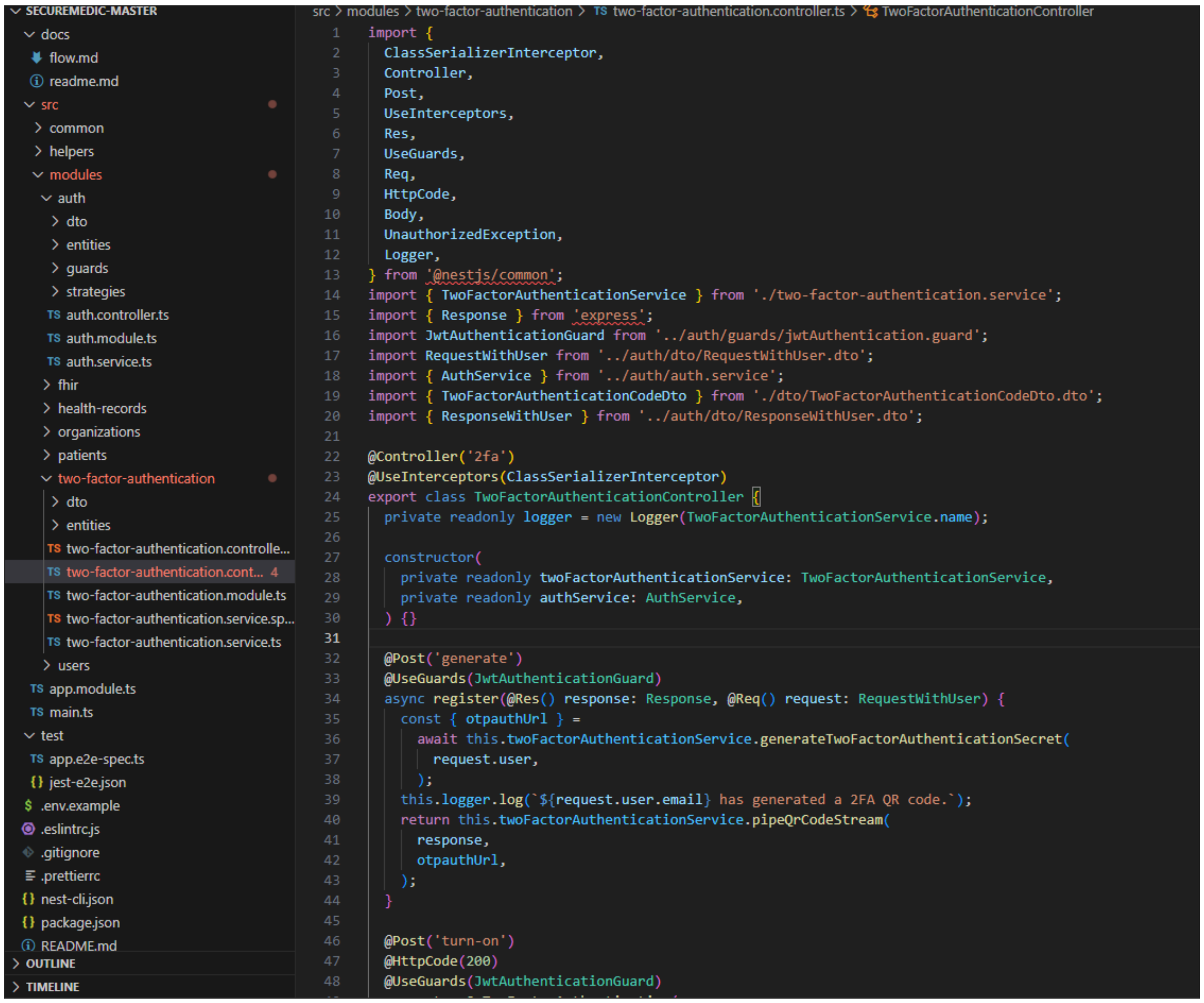
{
”name”: [
{
”given”: [
”Nelsonn”
]
}
]
}

Code Samples



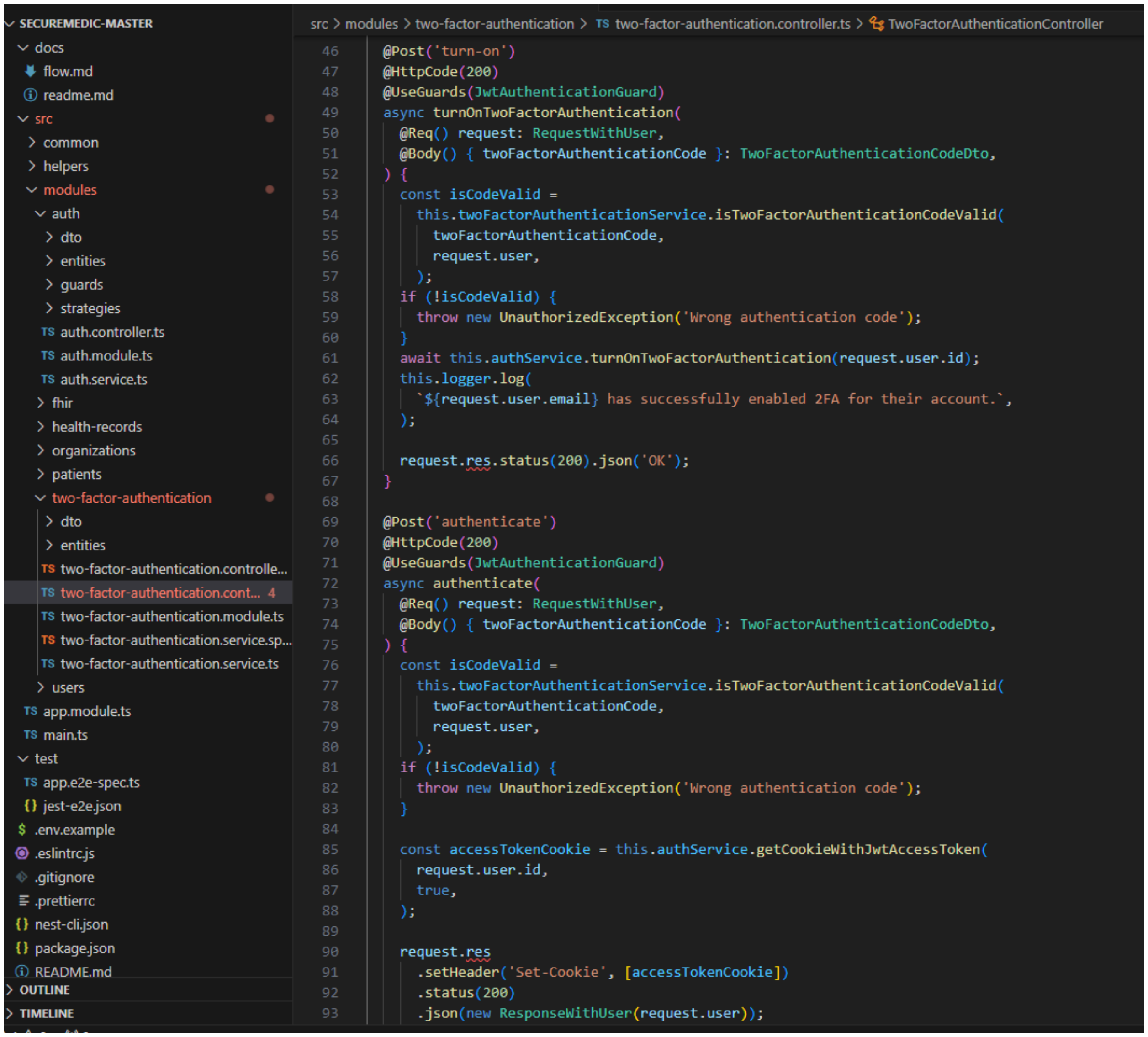
```typescript
import {
  Req,
  Controller,
  HttpCode,
  Get,
  Post,
  UseGuards,
  Res,
  Logger,
} from '@nestjs/common';
import { LocalAuthenticationGuard } from './guards/localAuthentication.guard';
import { AuthService } from './auth.service';
import RequestWithUser from './dto/RequestWithUser.dto';
import { Response } from 'express';
import JwtAuthenticationGuard from './guards/jwtAuthentication.guard';
import JwtRefreshGuard from './guards/jwtRefresh.guard';
import { ResponseWithUser } from './dto/ResponseWithUser.dto';

@Controller('authentication')
export class AuthController {
  constructor(private readonly authenticationService: AuthService) {}
  private readonly logger = new Logger(AuthService.name);

  @HttpCode(200)
  @UseGuards(LocalAuthenticationGuard)
  @Post('log-in')
  async logIn(@Req() request: RequestWithUser) {
    const { user } = request;
    const accessTokenCookie =
      this.authenticationService.getCookieWithJwtAccessToken(user.id);
    const { cookie: refreshTokenCookie, token: refreshToken } =
      this.authenticationService.getCookieWithJwtRefreshToken(user.id);

    await this.authenticationService.setCurrentRefreshToken(
      refreshToken,
      user.id,
    );

    request.res.setHeader('Set-Cookie', [
      accessTokenCookie,
      refreshTokenCookie,
    ]);

    this.logger.log(`${user.email} has successfully logged in.`);

    return new ResponseWithUser(user);
  }
```

Figure 13: Authorization Code



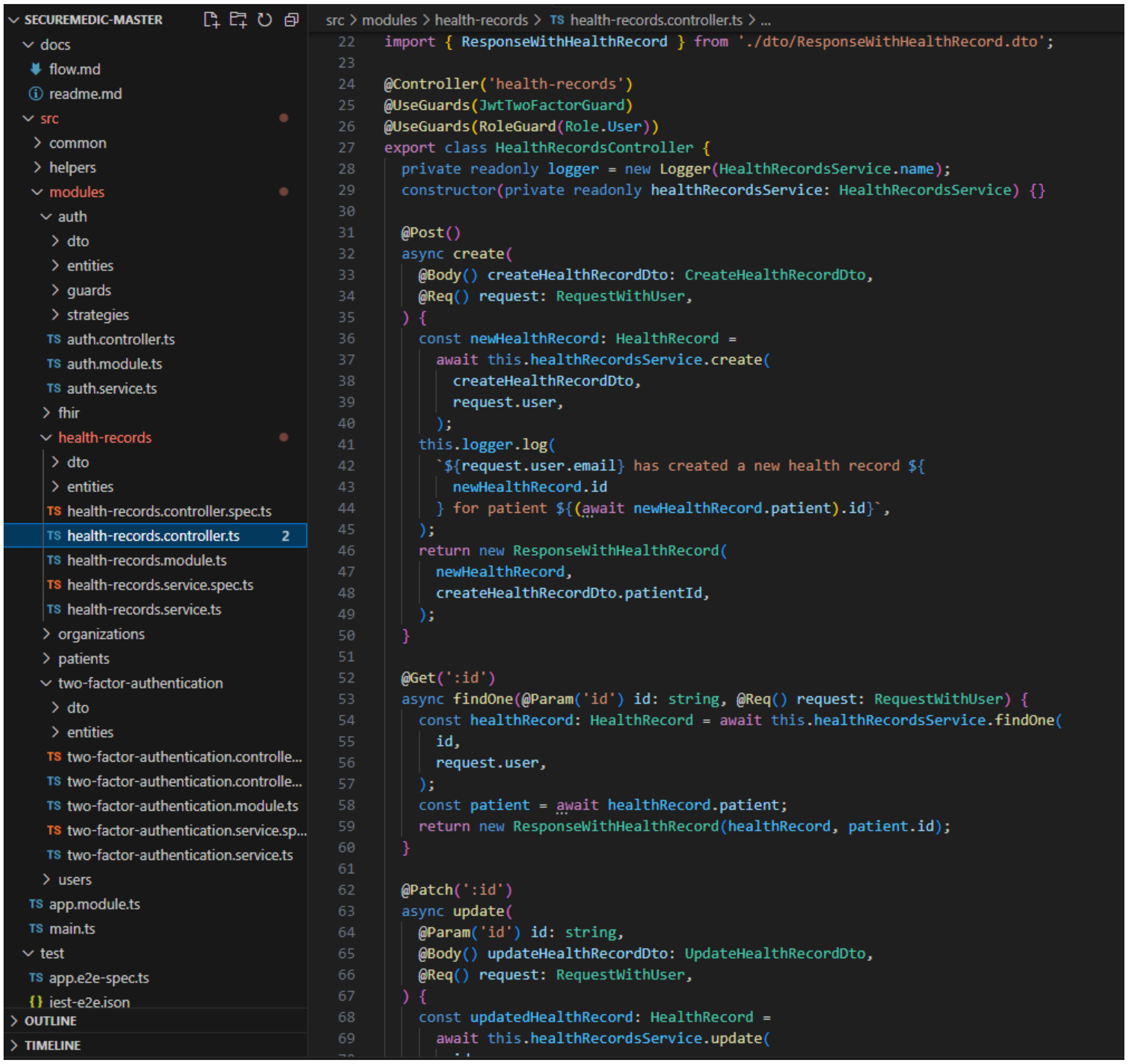
```
import {
  ClassSerializerInterceptor,
  Controller,
  Post,
  UseInterceptors,
  Res,
  UseGuards,
  Req,
  HttpCode,
  Body,
  UnauthorizedException,
  Logger,
} from '@nestjs/common';
import { TwoFactorAuthenticationService } from './two-factor-authentication.service';
import { Response } from 'express';
import JwtAuthenticationGuard from '../auth/guards/jwtAuthentication.guard';
import RequestWithUser from '../auth/dto/RequestWithUser.dto';
import { AuthService } from '../auth/auth.service';
import { TwoFactorAuthenticationCodeDto } from './dto/TwoFactorAuthenticationCodeDto.dto';
import { ResponseWithUser } from '../auth/dto/ResponseWithUser.dto';

@Controller('2fa')
@UseInterceptors(ClassSerializerInterceptor)
export class TwoFactorAuthenticationController {
  private readonly logger = new Logger(TwoFactorAuthenticationService.name);

  constructor(
    private readonly twoFactorAuthenticationService: TwoFactorAuthenticationService,
    private readonly authService: AuthService,
  ) {}

  @Post('generate')
  @UseGuards(JwtAuthenticationGuard)
  async register(@Res() response: Response, @Req() request: RequestWithUser) {
    const { otpauthUrl } =
      await this.twoFactorAuthenticationService.generateTwoFactorAuthenticationSecret(
        request.user,
      );
    this.logger.log(`${request.user.email} has generated a 2FA QR code.`);
    return this.twoFactorAuthenticationService.pipeQrCodeStream(
      response,
      otpauthUrl,
    );
  }

  @Post('turn-on')
  @HttpCode(200)
  @UseGuards(JwtAuthenticationGuard)
```

Figure 14: Two Factor Authentication Code



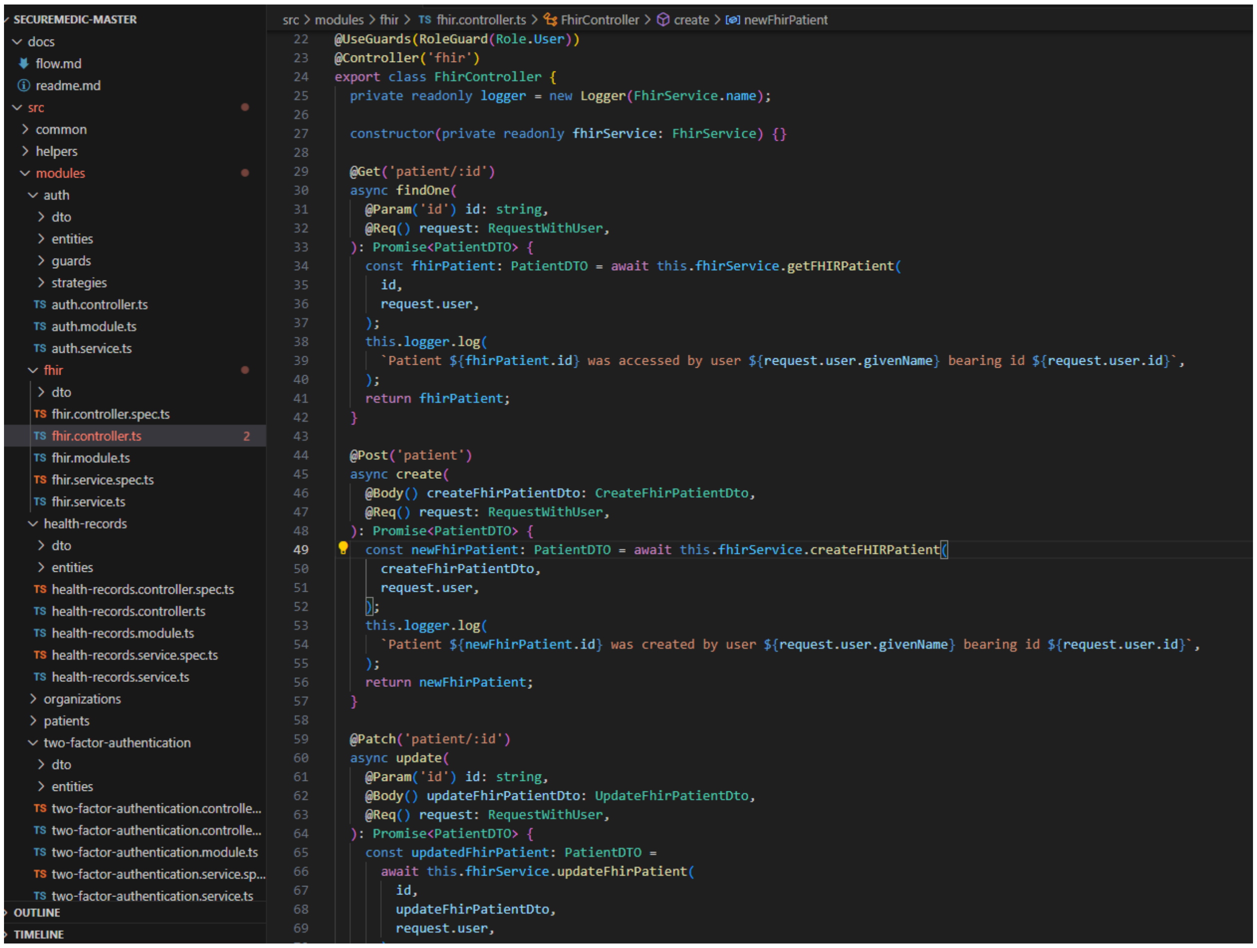
```
@Post('turn-on')
@HttpCode(200)
@UseGuards(JwtAuthenticationGuard)
async turnOnTwoFactorAuthentication(
  @Req() request: RequestWithUser,
  @Body() { twoFactorAuthenticationCode }: TwoFactorAuthenticationCodeDto,
) {
  const isCodeValid =
    this.twoFactorAuthenticationService.isTwoFactorAuthenticationCodeValid(
      twoFactorAuthenticationCode,
      request.user,
    );
  if (!isCodeValid) {
    throw new UnauthorizedException('Wrong authentication code');
  }
  await this.authService.turnOnTwoFactorAuthentication(request.user.id);
  this.logger.log(
    `${request.user.email} has successfully enabled 2FA for their account.`,
  );

  request.res.status(200).json('OK');
}

@Post('authenticate')
@HttpCode(200)
@UseGuards(JwtAuthenticationGuard)
async authenticate(
  @Req() request: RequestWithUser,
  @Body() { twoFactorAuthenticationCode }: TwoFactorAuthenticationCodeDto,
) {
  const isCodeValid =
    this.twoFactorAuthenticationService.isTwoFactorAuthenticationCodeValid(
      twoFactorAuthenticationCode,
      request.user,
    );
  if (!isCodeValid) {
    throw new UnauthorizedException('Wrong authentication code');
  }

  const accessTokenCookie = this.authService.getCookieWithJwtAccessToken(
    request.user.id,
    true,
  );

  request.res
    .setHeader('Set-Cookie', [accessTokenCookie])
    .status(200)
    .json(new ResponseWithUser(request.user));
```

Figure 15: Two Factor Authentication Request Code


SECUREMEDIC-MASTER
docs
flow.md
readme.md
src
common
helpers
modules
auth
dto
entities
guards
strategies
auth.controller.ts
auth.module.ts
auth.service.ts
fhir
health-records
dto
entities
health-records.controller.spec.ts
health-records.controller.ts 2
health-records.module.ts
health-records.service.spec.ts
health-records.service.ts
organizations
patients
two-factor-authentication
dto
entities
two-factor-authentication.controlle...
two-factor-authentication.controlle...
two-factor-authentication.module.ts
two-factor-authentication.service.sp...
two-factor-authentication.service.ts
users
app.module.ts
main.ts
test
app.e2e-spec.ts
jest-e2e.json
OUTLINE
TIMELINE
src > modules > health-records > health-records.controller.ts > ...

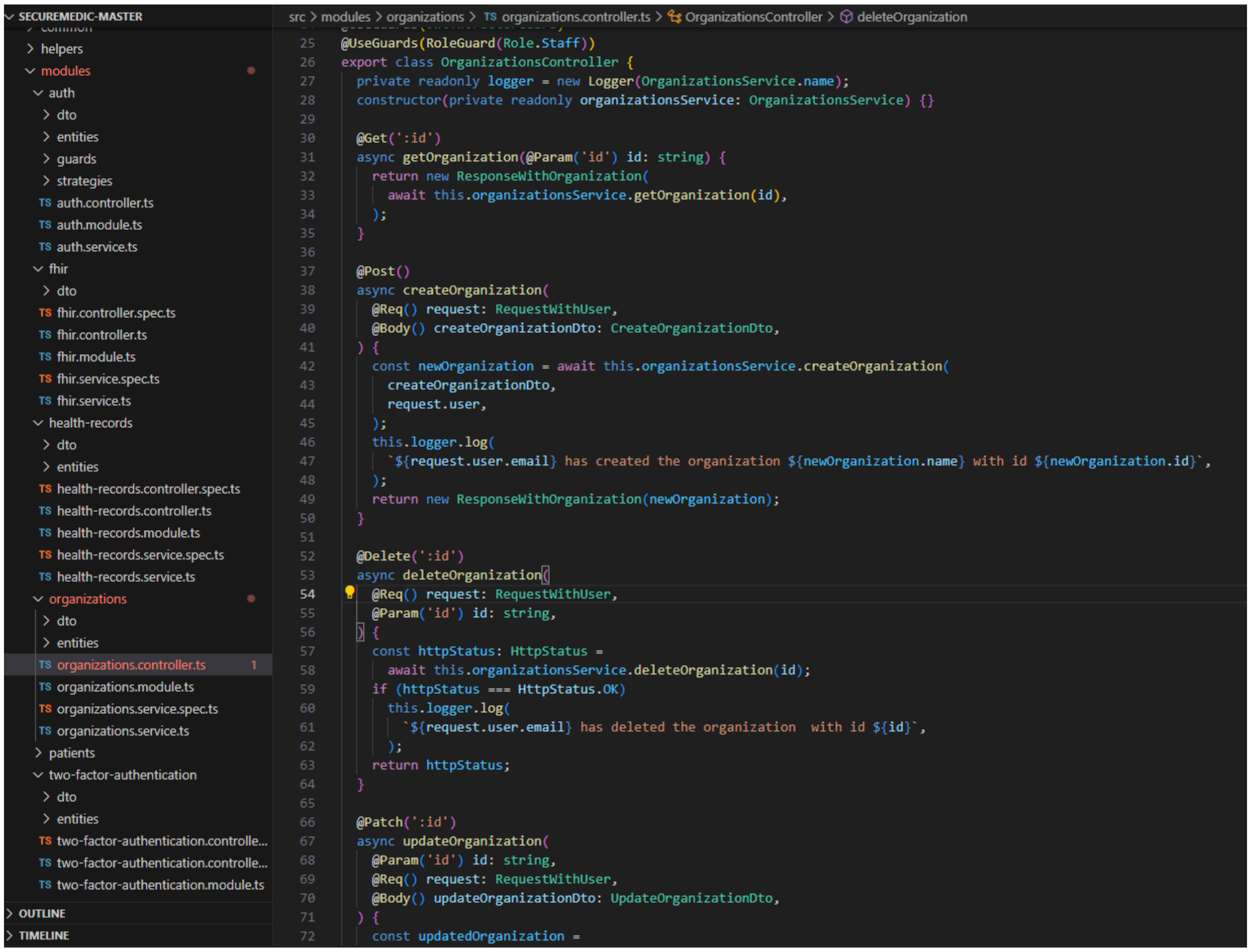
```
import { ResponseWithHealthRecord } from './dto/ResponseWithHealthRecord.dto';

@Controller('health-records')
@UseGuards(JwtTwoFactorGuard)
@UseGuards(RoleGuard(Role.User))
export class HealthRecordsController {
  private readonly logger = new Logger(HealthRecordsService.name);
  constructor(private readonly healthRecordsService: HealthRecordsService) {}

  @Post()
  async create(
    @Body() createHealthRecordDto: CreateHealthRecordDto,
    @Req() request: RequestWithUser,
  ) {
    const newHealthRecord: HealthRecord =
      await this.healthRecordsService.create(
        createHealthRecordDto,
        request.user,
      );
    this.logger.log(
      `${request.user.email} has created a new health record ${
        newHealthRecord.id
      } for patient ${(await newHealthRecord.patient).id}`,
    );
    return new ResponseWithHealthRecord(
      newHealthRecord,
      createHealthRecordDto.patientId,
    );
  }

  @Get(':id')
  async findOne(@Param('id') id: string, @Req() request: RequestWithUser) {
    const healthRecord: HealthRecord = await this.healthRecordsService.findOne(
      id,
      request.user,
    );
    const patient = await healthRecord.patient;
    return new ResponseWithHealthRecord(healthRecord, patient.id);
  }

  @Patch(':id')
  async update(
    @Param('id') id: string,
    @Body() updateHealthRecordDto: UpdateHealthRecordDto,
    @Req() request: RequestWithUser,
  ) {
    const updatedHealthRecord: HealthRecord =
      await this.healthRecordsService.update(
```

Figure 16: CRUD Health Records Code


SECUREMEDIC-MASTER
docs
flow.md
readme.md
src
common
helpers
modules
auth
dto
entities
guards
strategies
auth.controller.ts
auth.module.ts
auth.service.ts
fhir
dto
fhir.controller.spec.ts
fhir.controller.ts
fhir.module.ts
fhir.service.spec.ts
fhir.service.ts
health-records
dto
entities
health-records.controller.spec.ts
health-records.controller.ts
health-records.module.ts
health-records.service.spec.ts
health-records.service.ts
organizations
patients
two-factor-authentication
dto
entities
two-factor-authentication.controlle...
two-factor-authentication.controlle...
two-factor-authentication.module.ts
two-factor-authentication.service.sp...
two-factor-authentication.service.ts
OUTLINE
TIMELINE
src > modules > fhir > fhir.controller.ts > FhirController > create > newFhirPatient

```typescript
@UseGuards(RoleGuard(Role.User))
@Controller('fhir')
export class FhirController {
  private readonly logger = new Logger(FhirService.name);

  constructor(private readonly fhirService: FhirService) {}

  @Get('patient/:id')
  async findOne(
    @Param('id') id: string,
    @Req() request: RequestWithUser,
  ): Promise<PatientDTO> {
    const fhirPatient: PatientDTO = await this.fhirService.getFHIRPatient(
      id,
      request.user,
    );
    this.logger.log(
      `Patient ${fhirPatient.id} was accessed by user ${request.user.givenName} bearing id ${request.user.id}`,
    );
    return fhirPatient;
  }

  @Post('patient')
  async create(
    @Body() createFhirPatientDto: CreateFhirPatientDto,
    @Req() request: RequestWithUser,
  ): Promise<PatientDTO> {
    const newFhirPatient: PatientDTO = await this.fhirService.createFHIRPatient(
      createFhirPatientDto,
      request.user,
    );
    this.logger.log(
      `Patient ${newFhirPatient.id} was created by user ${request.user.givenName} bearing id ${request.user.id}`,
    );
    return newFhirPatient;
  }

  @Patch('patient/:id')
  async update(
    @Param('id') id: string,
    @Body() updateFhirPatientDto: UpdateFhirPatientDto,
    @Req() request: RequestWithUser,
  ): Promise<PatientDTO> {
    const updatedFhirPatient: PatientDTO =
      await this.fhirService.updateFhirPatient(
        id,
        updateFhirPatientDto,
        request.user,
```

Figure 17: CRUD FHIR Cod


SECUREMEDIC-MASTER
helpers
modules
auth
dto
entities
guards
strategies
auth.controller.ts
auth.module.ts
auth.service.ts
fhir
dto
fhir.controller.spec.ts
fhir.controller.ts
fhir.module.ts
fhir.service.spec.ts
fhir.service.ts
health-records
dto
entities
health-records.controller.spec.ts
health-records.controller.ts
health-records.module.ts
health-records.service.spec.ts
health-records.service.ts
organizations
dto
entities
organizations.controller.ts 1
organizations.module.ts
organizations.service.spec.ts
organizations.service.ts
patients
two-factor-authentication
dto
entities
two-factor-authentication.controlle...
two-factor-authentication.controlle...
two-factor-authentication.module.ts
OUTLINE
TIMELINE
src > modules > organizations > organizations.controller.ts > OrganizationsController > deleteOrganization

```typescript
@UseGuards(RoleGuard(Role.Staff))
export class OrganizationsController {
  private readonly logger = new Logger(OrganizationsService.name);
  constructor(private readonly organizationsService: OrganizationsService) {}

  @Get(':id')
  async getOrganization(@Param('id') id: string) {
    return new ResponseWithOrganization(
      await this.organizationsService.getOrganization(id),
    );
  }

  @Post()
  async createOrganization(
    @Req() request: RequestWithUser,
    @Body() createOrganizationDto: CreateOrganizationDto,
  ) {
    const newOrganization = await this.organizationsService.createOrganization(
      createOrganizationDto,
      request.user,
    );
    this.logger.log(
      `${request.user.email} has created the organization ${newOrganization.name} with id ${newOrganization.id}`,
    );
    return new ResponseWithOrganization(newOrganization);
  }

  @Delete(':id')
  async deleteOrganization(
    @Req() request: RequestWithUser,
    @Param('id') id: string,
  ) {
    const httpStatus: HttpStatus =
      await this.organizationsService.deleteOrganization(id);
    if (httpStatus === HttpStatus.OK)
      this.logger.log(
        `${request.user.email} has deleted the organization  with id ${id}`,
      );
    return httpStatus;
  }

  @Patch(':id')
  async updateOrganization(
    @Param('id') id: string,
    @Req() request: RequestWithUser,
    @Body() updateOrganizationDto: UpdateOrganizationDto,
  ) {
    const updatedOrganization =
```

Figure 18: CRUD Organizations Code