CSC406 - Technical Specifications: fEMR Central REST API Construction

Team: Chain Gang
Authors: Alex Burke, Jayant Devkar, Steven Ngo, Yuya Shimbori
Reviewers: BJ Klingenberg, Team fEMR Representatives

Created On: November 14, 2021
Last Updated: June 7, 2022

Jira Board:
https://platinum.cscaws.com:8443/secure/RapidBoard.jspa?rapidView=63
Project Repositories:
https://github.com/FEMR/femr,
https://github.com/CPSECapstone/super-femr,
https://github.com/FEMR/fEMR-OnChain-Core,
https://github.com/CPSECapstone/ChainGang

Table of Contents

Introduction

Overview

Team fEMR (Fast Electronic Medical Records) has created an electronic medical record system (EMR) designed for mobile teams of medics setting up “pop-up” clinics at remote refugee locations around the world. Still, their Legacy fEMR kits currently do not effectively communicate with one another to establish synchronized data. Their fEMR-OnChain EMR, a blockchain-based architecture, also does not currently synchronize with Legacy fEMR kits, so there is a need for a centralized data repository that can support multiple data schemas. Our project will address this concern and build off of fEMR-OnChain EMR with guidance and feedback from Team fEMR representatives, including Sean Batzel, Sarah Draugelis, and Andy Mastie. We aim to create a set of REST API endpoints surrounding a centralized data repository, fEMR Central. We envision it to have the capability to receive data from fEMR-OnChain and Legacy fEMR kits of varying schema, format new data and support data merges, and export data in HL7-FHIR format to external medical collaborators.

Glossary

- Legacy fEMR = Team fEMR’s self-contained hardware and software kit that can provide its local WiFi network to enable access to Team fEMR’s electronic medical record system. Legacy fEMR predates fEMR-OnChain.
- fEMR-OnChain = Team fEMR’s newest solution enables communication and synchronization between Legacy fEMR kits and a centralized database, which leverages blockchain technology (Amazon’s Quantum Ledger Database, Fennel Lab’s Fennel Protocol, and Substrate).
- Personally Identifiable Information (PII) = Data that ties to a single individual (either as a single attribute or as a combination), including national identification numbers, patient identification numbers, date of birth, ethnicity information, etc. PII also includes protected health information (PHI). Team fEMR and our solution are responsible for protecting patient data that fall under this category, out of data privacy concerns and law requirements (across multiple countries).
- HL7-FHIR = Health Level Seven International (HL7), a non-profit, ANSI-accredited health care data standards development organization, maintains an API for electronic health record exchange, known as Fast Healthcare Interoperability Resources (FHIR). Team fEMR would like their data to be translatable to formats and resources from the HL7-FHIR standard to facilitate easier external healthcare collaboration.

Background

Especially with the ongoing COVID-19 pandemic, there is a critical need for an electronic medical record system with constantly-synced data even at remote locations around the world so that mobile medical teams have all the information necessary to make the right decisions regarding patient treatment. Up-to-date and accurate patient data records are vital for determining what medical resources (e.g., medicine, specialized tools) should be kept on hand or delivered as soon as possible by, for example, arriving medical teams.

Fortunately, Team fEMR already has an open-source system supporting the work of medical professionals and the lives of thousands of refugee patients. The Legacy fEMR kits and fEMR-OnChain are currently independent and require a solution that allows the two to communicate with one another. Team fEMR ultimately wants to save more lives, especially in the case of continuity of care and treatment as refugees arrive at different mobile medical sites. Solving the above problem enables medical professionals to make appropriately informed decisions.

Product and Technical Requirements

Product Requirements

- As a user, I should be able to update and store information about my health on a centralized and secure database so that the medical professionals know what they need to know about me and I am aware of what information they have on me.
- As a doctor, I should be able to retrieve updated health information on a specific user/patient so that I can provide proper care depending on the patient's needs and data.
- As a developer, I should be able to change the system without having access to sensitive user information, so user privacy is maintained and the system maintains a separation of privileges.
- As a developer, I can merge data of varying schema from both fEMR-OnChain and Legacy fEMR kits without any conflicts (in the centralized schema).
- As an external medical collaborator, I can retrieve patient information in HL7-FHIR standardized format so that I can easily recognize all the fields I need for efficient and proper treatment.
- As a doctor, I should be able to access medical records that are accurate and synchronized between all locations/tents so that if I am seeing a patient that visited a different tent sometime before, I have all the information needed to provide proper care.

Technical Requirements

- The system shall be compatible with Chrome, Firefox, and Safari.

- The system will hold our user's data securely and retrieve it swiftly.
- The system will only allow certified admins and the user itself to retrieve sensitive information on the user.
- The system should be able to detect duplication of data so that there aren't two profiles for the same user.
- The system will allow users to sign in seamlessly with a singular pair of usernames and passwords.
- The system should be able to update the database model when connected to the internet without any loss of data.
- The system should be able to perform software upgrades when connecting to the internet and an update is available.
- The system should be able to work with kits at different database model levels.

Out-of-Scope Goals

- Optimizing the size of the kit used by doctors in rural areas to record patient data.
- Reducing the technical tasks doctors need to perform to use the data synchronization API.

Future Goals

- Creating an interface/dashboard utilizing and aggregating non-PII data collected from users of our fEMR Central API and the data in fEMR Central (e.g., tracking number of patients, medicine/equipment needed).

Assumptions

- Access to the Relational Database Service (RDS) table that already exists in AWS.
- OnChain repo can communicate with fEMR Central for database synchronization purposes.

Solutions

Current Solution and Design

Team fEMR already has an open-source electronic medical records system that can be deployed as a self-contained kit, but there is no capability to synchronize data between the legacy kits. Below are database models for Legacy fEMR and fEMR-OnChain based on the data and variables in the different classes from their respective code repositories.

fEMR-OnChain is Team fEMR's newest EMR version that relies on blockchain technology (Amazon's Quantum Ledger Database, Fennel Lab's Fennel Protocol, and Substrate). There is currently no communication between fEMR-OnChain and the Legacy fEMR kits.

fEMR-OnChain Database Model:
https://lucid.app/lucidchart/18c2fe9d-a66b-4cdc-be55-e70c9c2b778f/edit?viewport_loc=-2935%2C17%2C9271%2C4505%2C0_0&invitationId=inv_8ecf1c73-e56c-4218-867e-d77e81f14ffb

Legacy fEMR Database Model:
https://lucid.app/lucidchart/89f466fc-95d5-468c-847a-3febb8241369/edit?invitationId=inv_5dfc0f7f-c447-4e16-8895-4d8d0690f500

There currently exists a set of REST API endpoints that allow for interfacing with fEMR-OnChain. It includes GET, POST, PUT, PATCH, and DELETE methods for the different custom data types pertaining to health information collected by Team fEMR, such as Administration Schedule, Encounter, Instance, InventoryCategory, Patient, etc.

To provide an example of the different data types/models currently available above is a data model of Patient. The rest of the API documentation can be accessed at https://chain.teamfemr.org/swagger/.

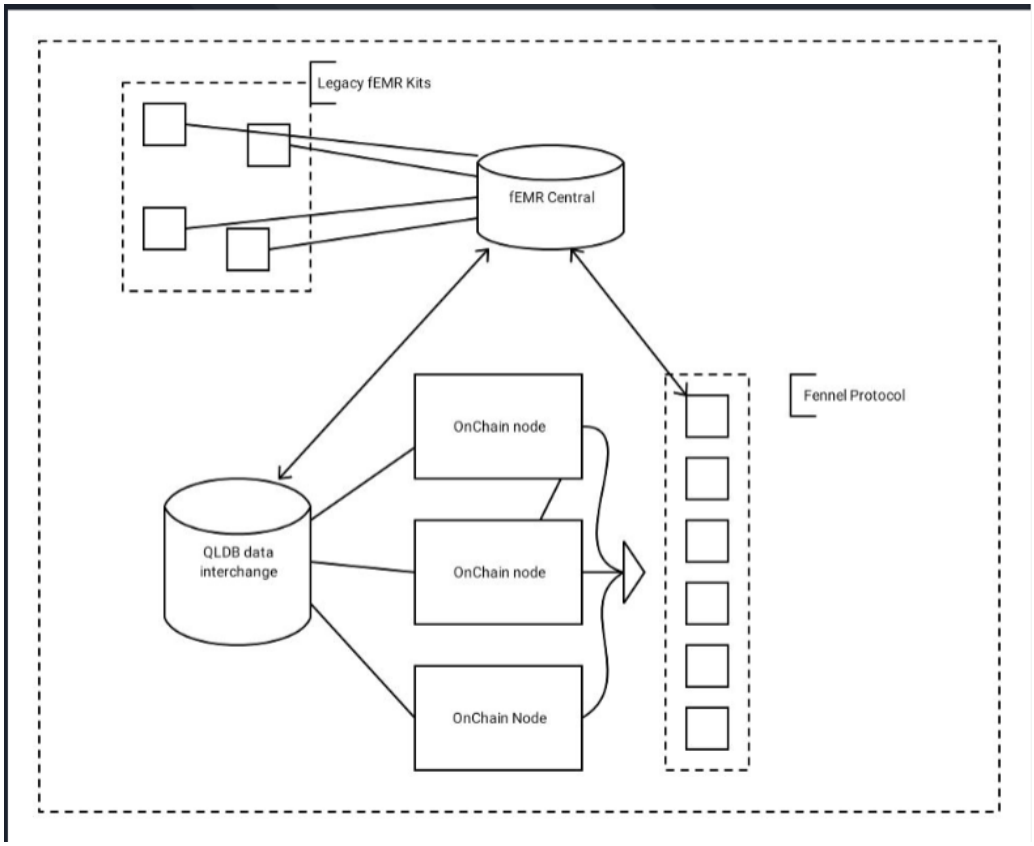
For additional references, below are GitHub repositories for the different kits:
2020-21 Cal Poly SE Capstone's super-femr: https://github.com/CPSECapstone/super-femr
fEMR-OnChain: https://github.com/FEMR/fEMR-OnChain-Core
Legacy fEMR: https://github.com/FEMR/femr

Proposed Solution and Design

We aim to create a set of REST API endpoints surrounding a centralized data repository, fEMR Central, with the capability to receive data from fEMR-OnChain and Legacy fEMR kits of varying schema, format new data and support data merges, and export data in HL7-FHIR format to external medical collaborators. Utilizing the current database models of Legacy fEMR and fEMR-OnChain, we will have to design a way for the data to "sync up" despite the differences in the data model and system needs.

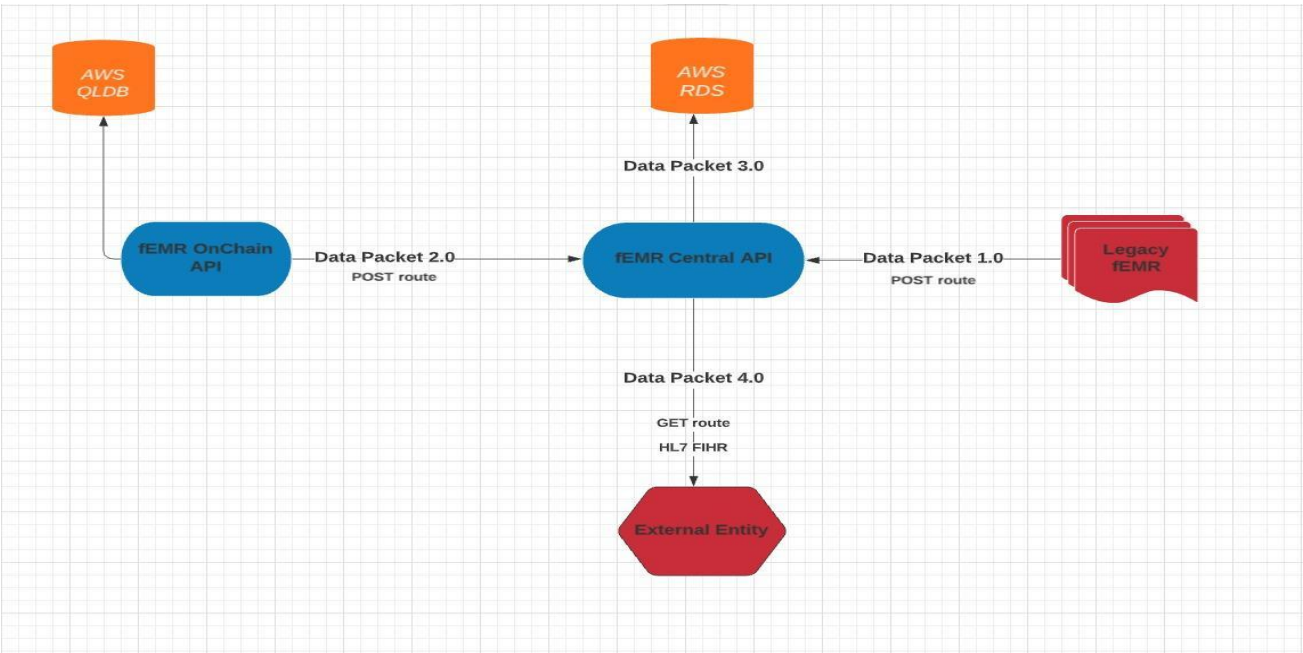
We are adopting the existing tech stack that Team fEMR developers are already using. The following breakdown is a summary of the tech stack:

- Django for fEMR-OnChain
- Java and Play Framework for Legacy fEMR
- Amazon Quantum Ledger Database (QLDB) for backend blockchain
- PostSQL for backend legacy database.



The diagram above, created by Sean Batzel, serves as a visual representation of the intended communication between the Legacy fEMR kits, fEMR Central, Amazon QLDB, and fEMR-OnChain (alongside Fennel Protocol).

There is currently no presentation layer apart from the interfaces of the kits themselves, but we will need to consider this further when planning an interface/dashboard that utilizes our data synchronization API.



The diagram above shows our intended fEMR Central API implementation, with data packets coming from fEMR-OnChain and Legacy fEMR kits and eventually allowing for data packets to be sent to external healthcare entities. Data packets from fEMR-OnChain and Legacy fEMR kits will utilize the Central API's POST routes, sending data as JSON that will be converted and processed accordingly (based on what data it is), and the data will be sent to fEMR Central's AWS RDS. External healthcare entities will then be able to utilize the Central API's GET routes to receive data in HL7-FHIR standard format as needed.

We are using Django to implement the fEMR Central API, and our POST and GET routes currently focus on patient data. We currently plan on documenting the process for Legacy fEMR integration into fEMR Central.

Test Plan

We will be using JUnit for the legacy fEMR repository and Pytest for the fEMR-OnChain repository as our unit testing tools, though we have not yet established a minimum bar for code coverage. Sean Batzel mentioned the current unit tests for fEMR-OnChain range from around 20-40% depending on the machine being used, but we will continue to discuss with Sean what our goal should be for coverage.

We are currently using Pytest for our separate ChainGang repository, primarily for unit testing of fEMR data to HL7-FHIR JSON format functions.

Scala will be used for continuous integration (CI) in legacy fEMR and GitHub Actions will be used for CI in fEMR-OnChain. We will also consider using Cypress for end-to-end testing.

Alternate Solutions

Considering Team fEMR's main problem currently is with data synchronization, there are two alternative solutions as opposed to our set of REST API endpoints:

- Using satellite internet for constant access with all of the legacy fEMR kits, but this would be expensive to maintain (especially in the context of having it at constant availability/ always online).
- Installing fEMR-OnChain on all legacy fEMR kits, but this would also be costly and not scalable (consider if Team fEMR wanted to continually expand with the number of mobile teams and prepare more technical equipment).

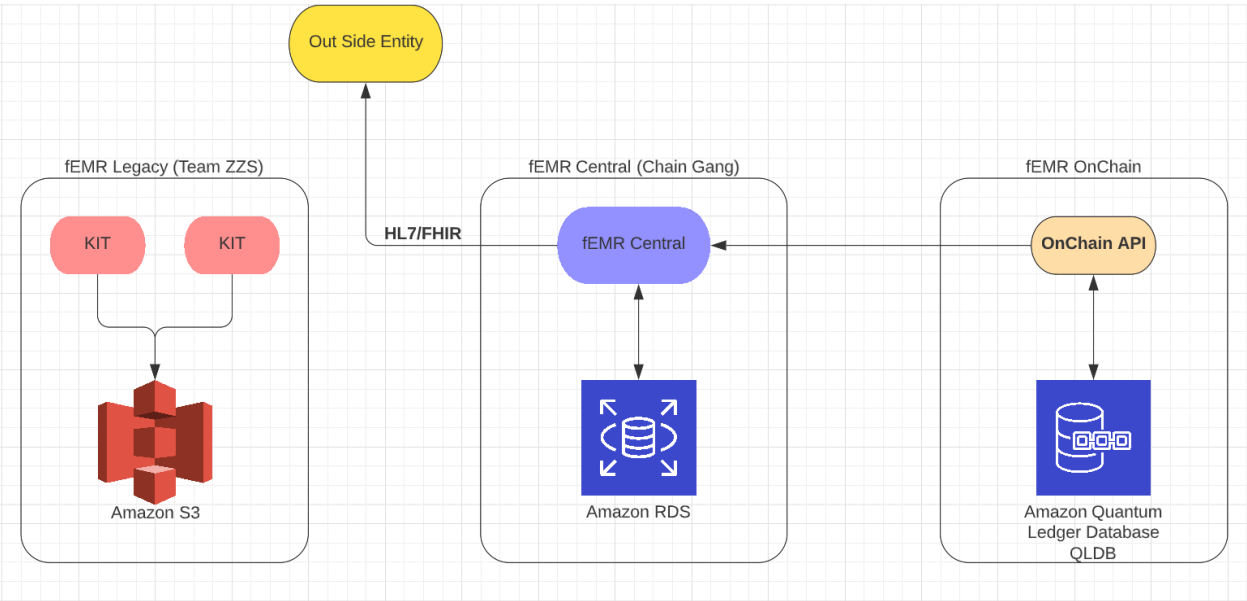
By developing a data synchronization API that allows for communication when the legacy fEMR kits do have internet access, without the need for satellite internet, we address the issues of cost and scalability.

Future Project Recommendation

To make fEMR Central the centralized database for Team fEMR, this project will need to be further developed and expanded to include integration with the Legacy fEMR kit deployments. This will require creating a workflow from the Legacy fEMR kits to Central RDS, in addition to needing to handle duplicate patient records and matching incoming record updates with existing ones.

Current Architecture

For the 2021-22 academic school year, we focused on building up the foundations of the fEMR Central API, starting with the integration of the fEMR-OnChain deployment.



- Team ZZS has written the code for kits to backup data to an AWS S3 bucket. They also have code for duplicate patient/data-checking.
- ChainGang has created an API that centralizes all the fEMR data in Central RDS, in addition to implementing the handshake between the fEMR Central API and the OnChain API. It also has code to convert patient data into HL7/FHIR format.
- The fEMR Central API's schema is designed to allocate both OnChain and Legacy fEMR data into a universal fEMR Central schema. (look at the bottom section for more info).

Integration With Legacy fEMR

To create a handshake between Legacy fEMR kits and Central RDS, we will need to use ChainGang's fEMR Central API and the code from Team ZZS, a 2020-2021 capstone team, to integrate the complete workflow. After discussing with Team fEMR and researching possible solutions, below is a summary of our recommendations.

Proposed Solution and Design

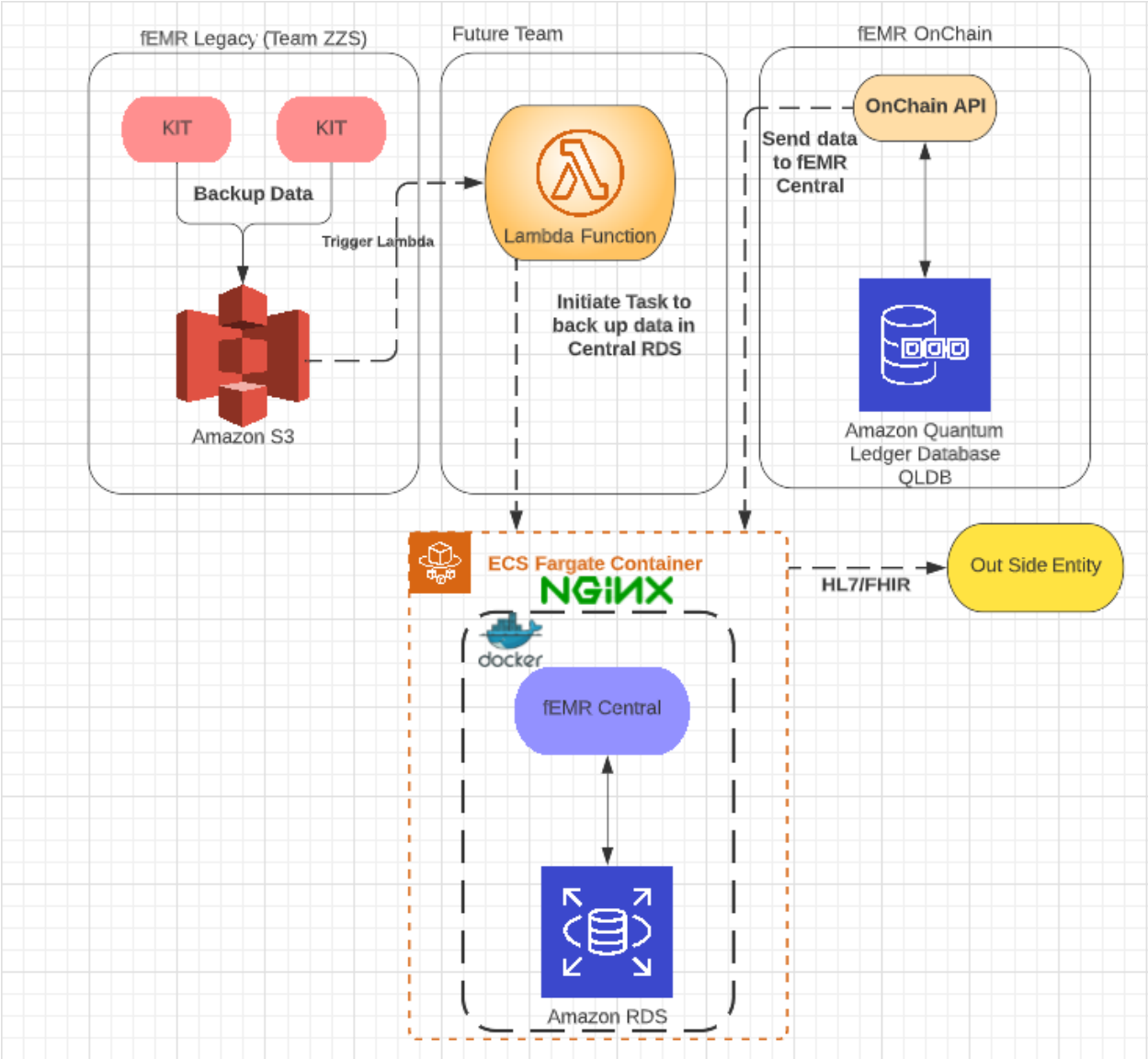
Data from Legacy fEMR kits is being backed up to AWS S3, so now we need to add a *lambda trigger* that will invoke a *lambda function* whenever the data was backed up to s3.
This lambda function will then ping *fEMR Central* to process the data and store it in *Central RDS*.

Below is a rough outline of possible steps for implementation of the solution:

1. Read Team ZZS' code for backing up data from the local SQL server in kits to an S3 bucket.
2. Create an S3 bucket in fEMR's AWS console.
3. Create a trigger for that S3 bucket and invoke a lambda for every upload.
4. Create a lambda function that monitors the S3 bucket by using S3 triggers.
5. Set up Amazon CloudWatch so that whenever the kits upload to S3, the upload event is recorded in CloudWatch.
    a. The event will then be forwarded to the lambda where the event will get processed.
6. Host fEMR Central so, the lambda can ping it with the backed-up data from the Legacy kits.
7. Two possible options on how to approach the lambda processing:
    a. Check the data packet in lambda itself, and have the duplicate-checking function run by allowing a lambda connection to Central RDS.
        - Then ping Central API with the data that's already checked and ready to be inserted.
    b. Ping the Central API with the data without checking for duplicacy
        - In this method, we would need to check for duplicates in the Central API before adding it to the Central RDS.
8. Create a logging system so that all data writes and updates are maintained for reporting and debugging purposes.

Proposed Architecture

This is a visualized recommendation of how data should flow between Legacy fEMR kits → S3 → Central RDS



In this architecture, we will be relying on the internal service ecosystem of AWS to complete the flow of data from fEMR Legacy Kits to fEMR Central (which is supposed to be the centralized database for all the fEMR services). The uploading of data from kits to s3 should trigger a lambda function, which then initiates a task in ECS Fargate to process the data by giving instructions to docker image fEMR Central. Fargate will then run and store data in Central RDS.

We can also use the URL generated by AWS Fargate thus exposing our fEMR Central to get pinged by data from fEMR OnChain our let outside entities request HL7/FIHR format patient.

To implement the complete workflow described above, we recommend dividing the project into three major parts:

1. Setting up Lambda Function to get invoked as soon as kits back-up in s3.
2. Modifying fEMR Central to process and handle data from kits.
3. Hosting fEMR Central and sending data to it.

1. S3 Bucket → AWS Lambda

Below are steps on how to build out the data flow from the S3 bucket to AWS Lambda, plus some details on the different AWS services proposed.

- ***AWS S3 Bucket:***
  Team ZZS uses S3 to backup data from the kits.
  Steps to configure:
  1. Create an S3 bucket in fEMR's AWS console, so that the kits can back-up data into an S3 bucket using Team ZZS' code.
     a. Reach out to Team fEMR to get AWS console access.
  2. Configure S3 Event Notification.
     a. S3 lets us forward an event to AWS lambda, so we need to configure the S3 event notification to invoke the lambda function we will be writing.
     b. We will need to create an IAM policy and role with correct permissions to be able to forward events to lambda.
        i. Reach out to Team fEMR to discuss the IAM role/policy generation.

- ***AWS Lambda:***
  We need a function that takes data from S3 and pings the route created for fEMR Legacy kit data in the central API. We need AWS Lambda to host this function so that every time data was deposited in S3, it will run this function.
  Steps to configure:
  1. Create an AWS lambda function to handle the S3 event.
     a. Use the blueprint option to create an S3 lambda function template
  2. Set the lambda trigger as the S3 trigger.
     a. Choose the S3 bucket you created.

2. Modifying fEMR Central

After the completion of the 1st half of the architecture, we will have a way of knowing when new data has arrived in S3 with the help of the lambda function. Once data is uploaded to s3 it will trigger the lambda function and from there we want lambda to be able to send that data to the Central API so that it can be processed and stored in RDS.

- ***fEMR Central:***
  ChainGang has created Django REST API to be able to centralize the patient records in RDS (from OnChain and Legacy fEMR). Currently, the Central API has routes to take in data from onChain and convert it to the new schema that Chain Gang has created and store it in the Central RDS. We need to do the same for taking in data from Legacy fEMR. So, to do that we need to take the following steps:

  1. Create a post route: We need to create a post route to take in data uploaded by fEMR kits. Convert that incoming data from the Legacy fEMR schema to the central schema.

  2. Maintain data transfer logs: Team ZZS' code currently *dumps* the data directly into s3. That means it doesn't maintain logs of data uploaded and it *uploads the whole* database from kits to s3. The data transfer logs between OnChain to Central are handled by OnChain; it would be best practice to keep logs of it in Central as well.

     We need to create a logging system to maintain the data integrity across fEMR -OnChain, Legacy fEMR, and fEMR Central. As fEMR Central acts as a junction to centralize data for fEMR, it would make sense to keep the log in Central.

     To be able to store the logging data in Central RDS, the future team would have to come up with a table with variables that will store the details each time new data was sent to fEMR Central from fEMR OnChain & fEMR Legacy Kits.

  3. Check For Duplicate Data: Even with maintaining logs, there are many reasons why duplicate data might end up in our Central RDS. Team ZZS has written a function that can check if a similar patient already exists by ranking matching patients. The function needs to be modified and implemented in fEMR Central's codebase to maintain data integrity while storing data in central RDS.

2. AWS Lambda → fEMR Central

- ***Hosting fEMR Central:***
  We need to host fEMR Central so that lambda can ping with new data from the kits that are backed up in S3.

Currently, only Cal Poly fEMR kits can back up patient data automatically to S3. Only Cal Poly kits using the fEMR Central will be used for testing, so it makes sense to come up with a short-term solution for now to reduce the usage of computing resources.

One of the short-term solutions will be to *containerize* fEMR Central so that it can be invoked by running a task from lambda with instructions in Dockerfile.

Steps to containerize fEMR Central:

1. Create a Docker Image.
    a. This image should have instructions on installing the dependencies.
    b. It should have instructions to start the Django server with Nginx configuration.
    c. Read the articles which are referred to at the end of this section for detailed instructions.
2. Test the image.
    a. You can test the image by building the image locally.
3. Host the image.
    a. The image needs to be hosted in a registry (server for storing images) so that our lambda function can access it.
    b. AWS ECR is a container registry by amazon. Hosting it on ECR will make it easy for lambda to access it.

- ***Setting up AWS Fargate:***

We need servers to run the dockerized fEMR Central so we will be using AWS Fargate which is nothing but evolved version of lambda which runs whole app/ long tasks "server-less" instead of running a function or small tasks like lambda.

Steps to send data to fEMR Central to process:

1. Install AWS and ECS CLI
2. Configure an ECS Cluster by using ECS CLI
3. Create Execution Roles and Security Group.
    a. Reach out to Team fEMR or capstone instructor for additional information for role permissions.
4. Configure Docker-yml files to host the app.
5. Setup configuration file for Nginx
6. Setup configuration file for ECS.

The article linked in the resource section gives detailed instructions on how to do the steps above. In the end, we will have a URL for the fEMR Central where we will be able to ping it from lambda with the s3 image.

Duplicate Record Matching

Source Code:
https://github.com/CPSECapstone/zzs-femr/blob/4f778a29bb8429ff780f0090986c4b0c863dd7d9/public/js/triage/triage.js#L678

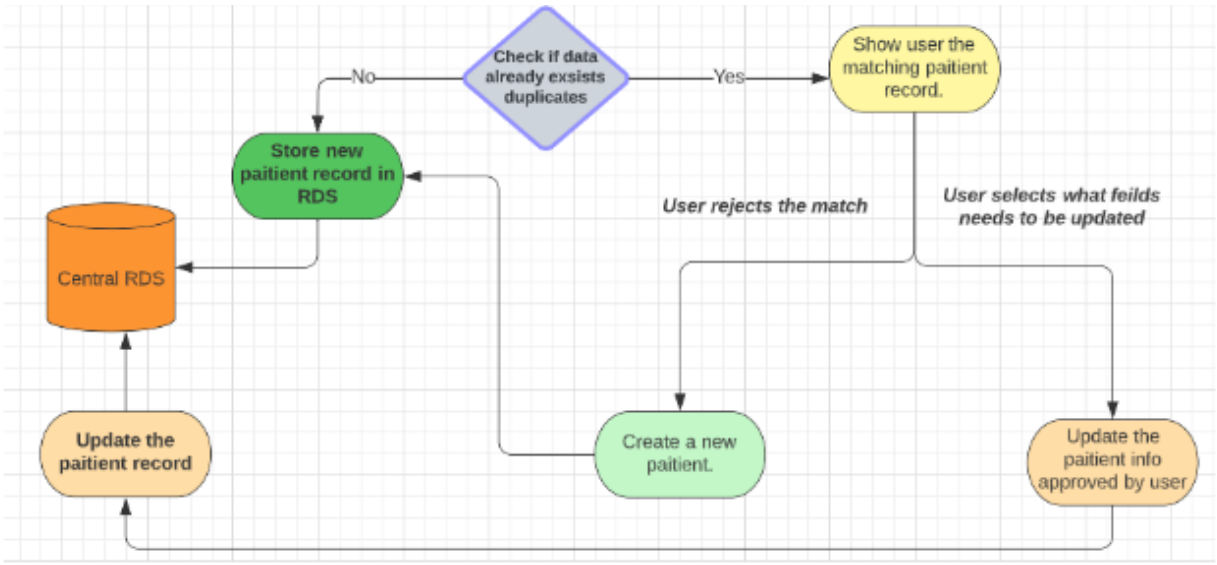
Team ZZS has already done a great job at creating a function that matches the patient records and sees if it already exists in the database.

The function uses the following variables to check for duplicate records:

- Name
- Phone
- Address
- Age
- Gender
- City

Currently, the function is meant for the patient schema in Legacy fEMR. To use this function in Central, we would need to make changes according to the patient schema in Central API. The solution to this will be trying to implement a duplicate_paitient_matcher() in Central API by updating the duplicate_matcher() from Team ZZS.

We need to update function so that it can use the combinations of variables listed above to find if a close match to the data that is being uploaded already exists in Central RDS.
Inorder to do that we would have to come up with scoring system which will score the patient data that is simillar to the new data being uploaded. The paitient with a score above certain level will confirm that the patient records are simillar enough to consider as the same patient. In this case we will update/add info of the old patient with any new info that came in.
We need to also log the transactions in which the old patients were updated with new info.



Saving Data into the Central API Database Schema

Once we have the data through the front of the central API, we need to save the data in an updated database model. The current database model (should) include all of the following tables in the synchronized master schema found here. The current implementation also includes API views for some of the OnChain integration, such as Patient. To get the remaining routes set up, we need to modify the codebase in a few key files:

1. Views.py: Create a new View class for the table you are adding. For now, we will need a POST and GET route (these are just methods in Django).
    a. For a POST route, we first need to parse the received JSON object into variables using data.get('attribute_name'). Next, we need to make another data object to hold all of the variables. Within this object, assign all of the variables to another variable with the same name as the attribute in the model as defined in models.py. Then we can simply make a new model using the data object, and save() it to the RDS database.
    b. For a GET route, we need to make an empty array to hold the returned data. Then we loop over every attribute in the table and add it to the array. Then we return the array in a data object.
2. Urls.py: We need to add the View as a path on the API by adding a new line to urlpatterns in the following manner: path(r'^tablename/', TablenameView.as_view()), Then we can add documentation for the view by importing the table's viewset from api_views.py and registering the viewset to the router below.
3. Serializers.py: Import the model name from ./model.py. Then make a ModelSerializer by adding a new class with the name: ModelnameSerializer, and copying the format of the other serializers.
4. Api_views.py: Import the model name from ./model.py and import the corresponding serializer name from ./serializers.py. Then make a new viewset with the name ModelnameViewSet and fill in the rest of the viewset in the same format as the other viewsets.

Services to be used:

- AWS S3
- AWS Lambda
- AWS Fargate
- Docker
- Ngnix
- fEMR Central
- fEMR Legacy Kits

Recommended Resources

- https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html - Tutorial on using an Amazon S3 trigger to invoke a Lambda function.
- https://www.youtube.com/watch?v=Gjnup-PuquQ - Video on a summary of Docker and containerization in general.
- https://medium.com/@vinodkv2511/build-and-deploy-a-rest-api-with-django-and-docker-part-4-19d55ca1053e - Article on building and deploying a REST API with Django and Docker.
- https://faun.pub/deploying-django-application-on-aws-fargate-in-8-minutes-f04373880e0a - Article on deploying Django app in AWS Fargate.

Some key points to be aware of:

- Team ZZS Warnings:
    - fEMR is not meant to diagnose, treat, cure or prevent disease.
    - fEMR may attempt to establish a secure remote connection when internet access becomes available. This behavior is configurable and turned off by default.
- ChainGang:
    - The handshake between OnChain and fEMR Central doesn't consider encryption.
    - The handshake between OnChain and fEMR Central currently relies on OnChain to do the checking data for duplicates part. But if the duplicate check is to be implemented in Central API from kits, then Central can also check OnChain data for duplicates.

Further Considerations

Security Considerations

Due to the nature of the medical patient data Team fEMR works with, security is a critical component that should not be disregarded. The Open Web Application Security Project (OWASP) has an entire project and documentation dedicated to API security (https://owasp.org/www-project-api-security/), and the top 10 list featured on the site can act as a checklist when analyzing our solution for any vulnerabilities. Some of the primary ones to look out for include excessive data exposure, broken user authentication, lack of rate-limiting, lack of input sanitization, and security misconfigurations. These flaws allow for denial of service attacks, injection attacks, and authentication attacks.

The above can be solved by following various security policies and principles throughout development that include but are not limited to:

- Open design = Always assume your design and architecture is available to the public, so security mechanisms should not rely on components being kept obscured/as a secret.

- Economy of mechanism = Do not overcomplicate API returns and only provide API calls with the minimal amount of information necessary to perform its described functions.
- Input sanitization = Always sanitize any user input by explicitly allowing necessary characters and denying any entries that do not follow set input guidelines.

We also would have to ensure any security solutions do not come at a great cost to performance and usability, as authorized users (i.e., medical professionals) should always be able to get what they need without going through massive security hurdles.

Privacy Considerations

Considering our solution would work directly with medical patient data and Team fEMR appears to be based in the United States (while deploying mobile medical teams all across the world), we would need to be compliant with the Health Insurance Portability and Accountability Act (HIPAA). HIPAA requires us to protect all patients' personally identifiable information (PII) and protected health information (PHI) while allowing patients to request and access the data Team fEMR may have on them. Though, while this may be more of a consideration for Team fEMR, we would have to make sure our solution is compliant with any privacy laws in other countries where medical teams utilize Team fEMR's electronic medical record system and our solution.

We would need to ensure medical patient data gets transmitted securely between the legacy fEMR kit and fEMR-OnChain; this may require an encryption protocol, but this should be discussed further with Team fEMR stakeholders. According to Team fEMR's website, they are HIPAA-compliant, and we will need to outline what our solution should have in terms of security and privacy functionality through additional discussion with Team fEMR. Fortunately, we will be provided with example/mock anonymized data when designing and testing our solution.

Deliberation

Discussion

None as of the time of writing.

Open Questions

None at the time of writing.

Previous Questions

- Team fEMR's project proposal from the Canvas page for CSC 402 mentions the goal of having fEMR be able to upgrade the software on top of data migration and synchronization when a kit connects to the internet. Is this still the case or has this already been accomplished by the previous capstone class? While there were mentions of this prior throughout the quarter, we would like to verify.
    - 11/29/21: Prof. Klingenberg said this is still a requirement and we plan on doing a deeper dive into last year's capstones at the start of 405 and their technical specifications.
- What kind of security and privacy mechanisms will we need to include in our project for the scope of the capstone sequence? For example, will we need to include some kind of encryption algorithm for medical data transmission, hashing algorithm when storing data in databases, and/or rate limitation on API calls?

References

- https://canvas.calpoly.edu/courses/63284/files/5578255?module_item_id=1397895 The template for this document, adapted by Dr. da Silva and Prof. Klingenberg.
    - https://stackoverflow.blog/2020/04/06/a-practical-guide-to-writing-technical-specs/ A reference to the above link.
- https://canvas.calpoly.edu/courses/63284/files/4842407?module_item_id=1397948 Team fEMR's project proposal from the CSC 402 Canvas page, "SLDM - Saving Lives through Data Merging".
- https://teamfemr.org/ Team fEMR's website.
- https://owasp.org/www-project-api-security/ OWASP's documentation on API security.
- https://www.hhs.gov/hipaa/index.html US Dept. of Health & Human Services' site on health information privacy.
- http://hl7.org/fhir/ HL7-FHIR (standard for health care data exchange) documentation home page.

Acknowledgments

- Thank you to Prof. BJ Klingenberg for his guidance and advice throughout CSC 402/405/406 and to Team fEMR representatives Sean Batzel, Sarah Draugelis, and Andy Mastie, for their feedback and demonstrations related to the project itself.
    - Sean Batzel's fEMR OnChain architecture diagram from his presentation on fEMR OnChain on October 14, 2021.