GetMyEHR

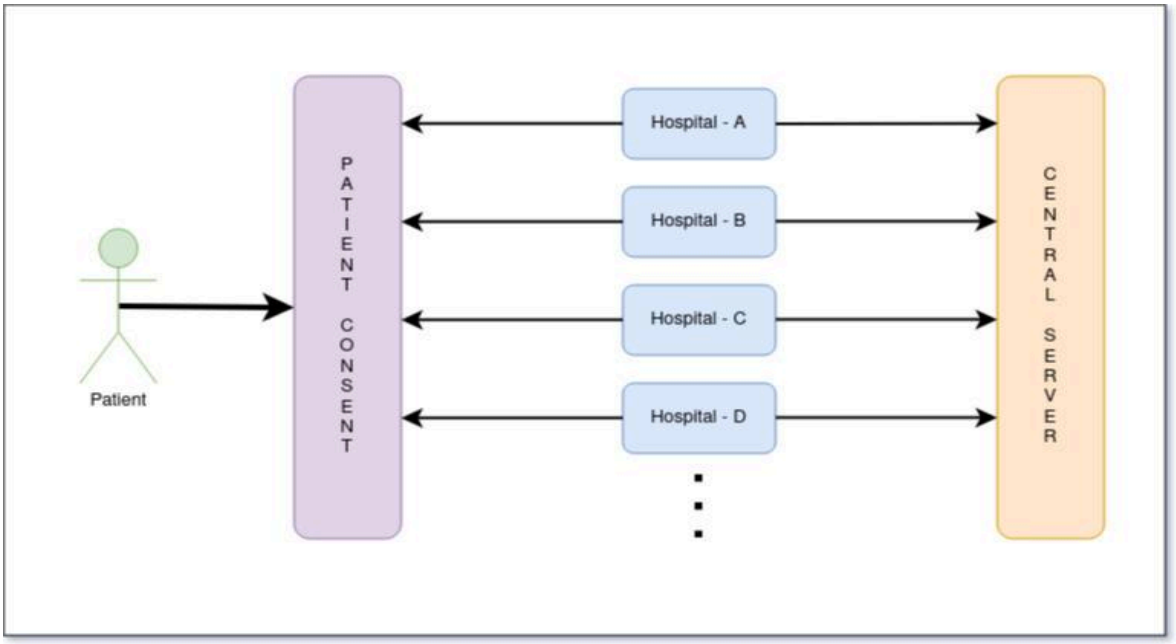
Developer Manual

Made by :

- Susheel Krishna
- Krishna Koushik
- Deekshitha
- Swaroop
- Pratyush

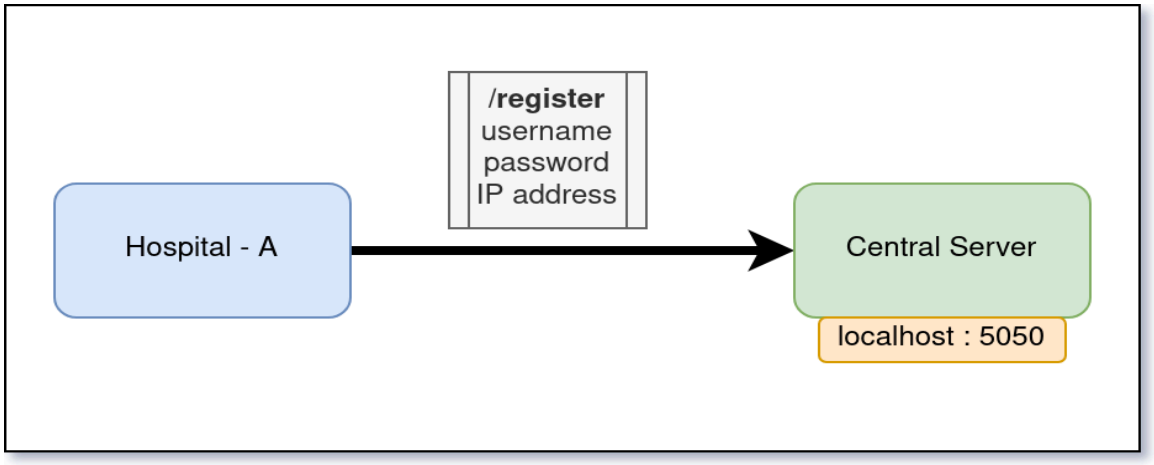
Contents

1. INTRODUCTION



- Initially Hospitals used to communicate Via emails, Messages, Calls etc. and they used to share the Data without Patient Consent.
- Our Approach is to make the communication better, Secure and easy. We are also considering the Patient Consent.
- In Our Approach, We are having 4 servers.
    - FHIR Server (HAPI - JPA Docker)
    - Patient Consent Server
    - Central Server (Get My EHR)
    - Hospital Server
- **FHIR Server** is helpful to store Medical Data in FHIR JSON format. For this, we are using a JPA docker image provided by HAPI. please refer more Here.
- **Patient Consent Server** : is used to get patient consent from patients and provide the patient ID to the Hospital.
- **Central Server** : Establishes Communication between Hospitals
- **Hospital Server** : Stores Hashes of Patients and Consent. Also generates Hash ID.

2. Registration

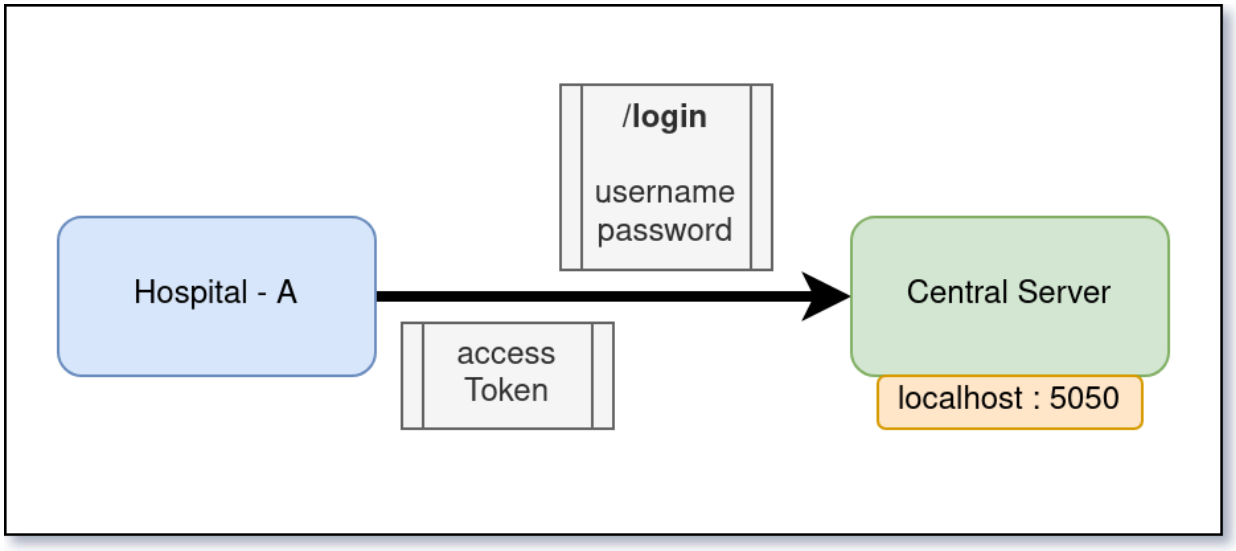


- Hospital Needs to register to the central server if it wants to communicate with other hospitals for Patient's Data.
- Hospitals should provide Username, Password and its IP address as parameters.
- Hospitals can register to the Central Server by sending the HTTP "**POST**"request to the central server with "**/register**" root.

Central_Server.py

```
@app.route('/register', methods=['POST', 'GET'])
def registeration():
    username = request.args.get('username')
    password = request.args.get('password')
    port_number = request.args.get('port')
    ip_address = request.args.get('ip_address')

    user_credentials = {
        "username" : username,
        "password" : password,
        "port" : port_number,
        "last_login" : "NONE",
        "token" : "NONE",
        "IP_Address" : ip_address
        }

    # Reading data
    with open('accounts.json', 'r') as file:
        data = json.load(file)

    # Storing in Memory for faster access.
    hospitals[username] = [port_number, ip_address]

    return "Successfully registered\n"
```

3. Login :

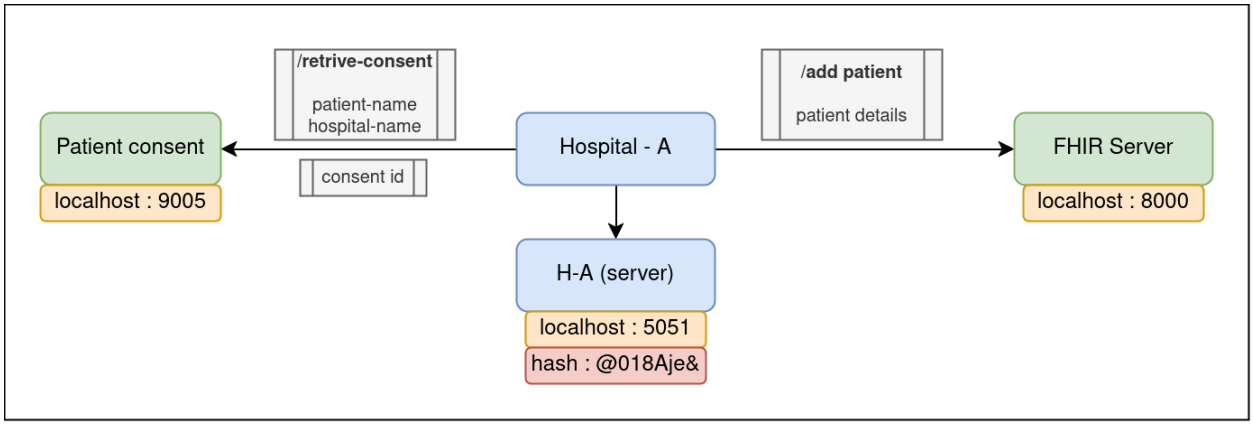


- Each Hospital needs an Access Token to communicate with other Hospitals.
- To get the Access Token, you need to Login to the Central Server with hospital Username and Password that we provided during registration time.
- Central server generates and Access token and stores recently accessed time. This helps in making Access tokens dynamic.
- Hospital should send HTTP “POST” request to the central server with the “**\login**” as root.

central_server.py

```
@app.route('/login', methods=['POST'])
def login():
    headers = request.headers
    username = headers.get('username')
    password = headers.get('password')

    for user in data["accounts"]:
        if(user["username"] == username):
            if(user["password"] == password):
                if(user["last_login"] == "NONE"):   # can add dynamic Tokens here
                    user["last_login"] = datetime.now().strftime("%H:%M:%S")
                    user["token"] = generate_token()
                    with open('accounts.json', 'w') as file:
                        json.dump(data, file)
                return user["token"]
```

4. Add Patient Details :



- For Adding Patient We need to insert Data of the Patient into the Local FHIR server of the hospital. While storing, we need to ask the patient consent from Global patient Consent Server. After getting patient consent, we should get HashID of the Patient using his demographics (fname, lname, dob) from Hospital Server.
- Now we have Patient consent and the HashID of the Patient. We need to Join These two and Store it in our Local Database so that whenever we get a Data request from Central Server, we get an Unique id. We should derive Patient details (fname and lname) from it.
- There are many servers involved Here. :
    - Patient Consent : asks the user for the consent and provides consent ID to the Hospital if user gives permission.
    - FHIR Local Server : Just stores the patient data in FHIR standards.
    - Hospital Server : Generates Hash ID and Stores the {Name : hashID} pair

- First We need to Store Data in FHIR Local Server by sending post request to the FHIR server with Patient root :

add-patient.html

```
const furl = fhir_server_url + "/fhir/Patient?_format=json"
const responseData = await postData(furl, formData);
console.log(responseData);
alert('Patient added successfully!');
```

- Now we need to ask the Patient Consent to the Patient Consent Server by sending GET request with “/share-consent” as root.

add-patient.html

```
const hospital = document.getElementById('hospital').value;
const consentUrl = patient_consent_url + `/share-consent?name=$
{fullname}&hospital=${hospital}`;
const consentResponse = await fetchConsent(consentUrl);
console.log(consentResponse);
```

- Now we need to ask Hash ID for the patient by providing his demographics :

add-patient.html

```
const fir_name = document.getElementById('fname').value;
const las_name = document.getElementById('lname').value;
const fullname = fir_name + las_name
const birth_date = document.getElementById('birthDate').value;
const hashUrl = hospital_server_url + `/give-me-hash?fname=$
{fir_name}&lname=${las_name}&dob=${birth_date}`;
var hash = await generateHash(hashUrl);
console.log('Hash generated successfully.');
```

In the Patient Consent server, we need to Store the request so that the user can see the requests from Hospitals and perform action accordingly. In the patient consent server, we will first store the data with permission zero and ask for the patient to give permission in “**get data()**”. We should wait for some waiting time (30 sec for now) to get patient consent.

patient_consent.py

```python
@app.route('/share-consent', methods=['GET'])
def share_consent():
    name = request.args.get('name')
    hospital = request.args.get('hospital')
    permission = 0
    consent_id = generate_token()
    patient = {
        "name" : name,
        "hospital" : hospital,
        "permission" : permission,
        "consent_id" : consent_id
    }
    store_data(patient)
    data = get_data(name, hospital)   # here we wait for 30 sec
```

patient_consent.py

```python
def get_data_from_json(hash_id, hospital):
    for each_patient in Patients_Data["patients"]:
        if(each_patient["name"] == hash_id):
            each_patient["hospital"] = hospital
    time.sleep(waiting_time)    # in this time patient has to give
access
    msg = "error"
    for each_patient in Patients_Data["patients"]:
        if(each_patient["name"] == hash_id and
each_patient["hospital"] == hospital):
            each_patient["hospital"] = ""
            if(each_patient["permission"] == 0):
                msg = "permission not given"
                break
            else:
                msg = each_patient["consent_id"]
                each_patient["permission"] = 0
        elif(each_patient["name"] == hash_id):
            return "Hospital Name Not Getting Set"
    return msg
```

- Now in the Hospital_server, we need to generate Hash id for the patient with his demographics. We are using SHA256 to generate Hash_id.

patient_consent.py

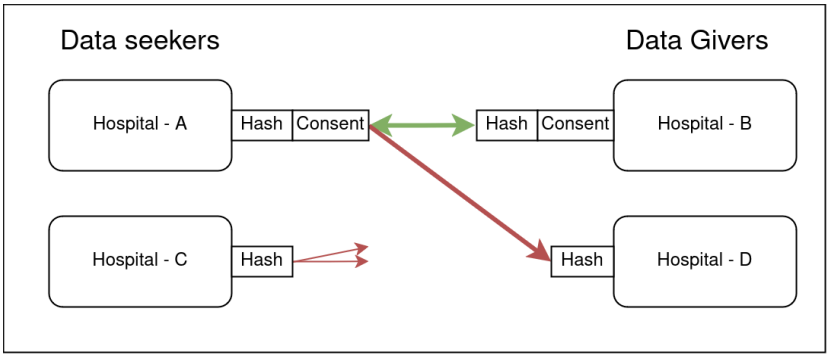
```python
@app.route('/give-me-hash', methods=['GET'])
def retrive_consent():
    fname = request.args.get('fname')
    lname = request.args.get('lname')
    dob = request.args.get('dob')
    data = generate_hash_id(first_name=fname, last_name=lname,
dob=dob)
    return data
```

patient_consent.py

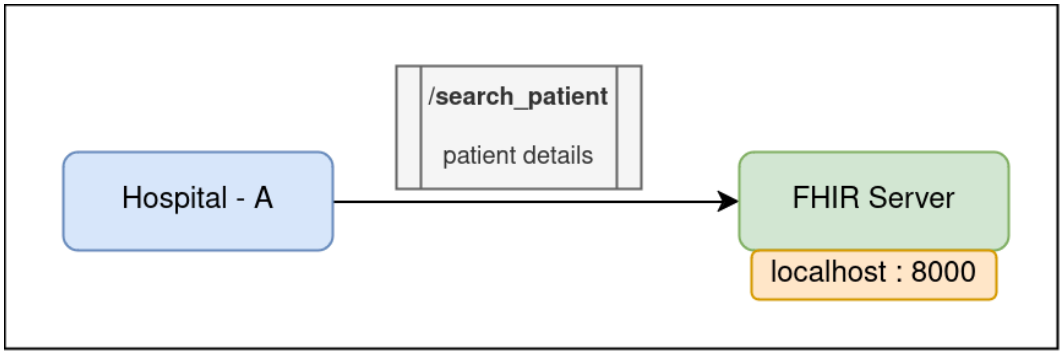
```python
def generate_hash_id(first_name, last_name, dob):
    date, year, month = dob.split("-")  # assuming date DD-MM-YYYY
    input_string = f"{first_name.lower()}{last_name.lower()}{date}
{month}{year}"
    hash_object = hashlib.sha256(input_string.encode())
    hash_id = hash_object.hexdigest()   # Hash.len = 16
    return hash_id
```

Concept of Hash + Consent :

If you could see that we are just appending the Consent ID to the Hash ID for generating a Unique_ID. Here the communication can only happens when you have consent ID. Because, if you don't have consent, then your unique ID will be different. So you cannot Share or Receive information from or to other hospitals.



5. Search Patient Data :

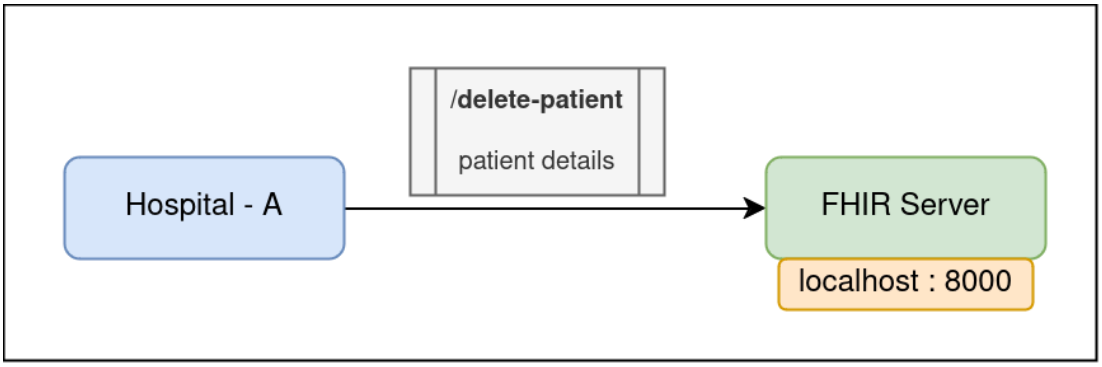


- Here we can search the patient Data in the Local FHIR server. Basically, this doesn't require any consent.
- You can perform this operation even if you don't register to the Central Server (Get My EHR).
- Here we just have to send a GET request to the FHIR server with a Patient query.

view_patient.html

```
function getPatientData() {
            var name = document.getElementById("nameInput").value;
            var url = fhir_server_url + "/fhir/Patient?
name="+name+"&_pretty=true"
            var xhr = new XMLHttpRequest();
            xhr.open("GET", url, true);
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    var response = JSON.parse(xhr.responseText);
                    displayPatientInfo(response);
                }
            };
            xhr.send();
        }
```

6. Delete_Patient :

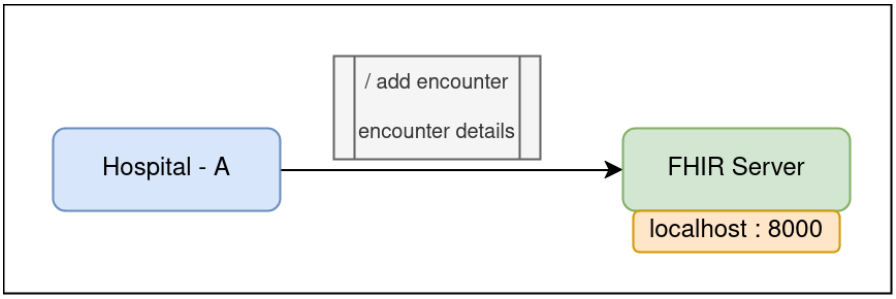


- This is also a local operation as search.
- Here we need to send “DELETE” request with patient query to the local FHIR server.
- We need patient ID to perform this operation.
    - You can get Patient ID while inserting the patient
    - You can get patient ID when you search that patient.
- You need to send that patient Patient id as query to FHIR server.

delete_patient.html

```
function deletePatient() {
    var p_id = document.getElementById("patientId").value;
    var url = fhir_server_url + "/fhir/Patient/" + p_id + "?
_pretty=true";

    var xhr = new XMLHttpRequest();
    xhr.open("DELETE", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4) {
            if (xhr.status === 200) {
                alert("Patient deleted successfully.");
            } else {
                alert("Error deleting patient. Status code: " +
xhr.status);
            }
        }
    };
    xhr.send();
}
```

7. Add Admission :

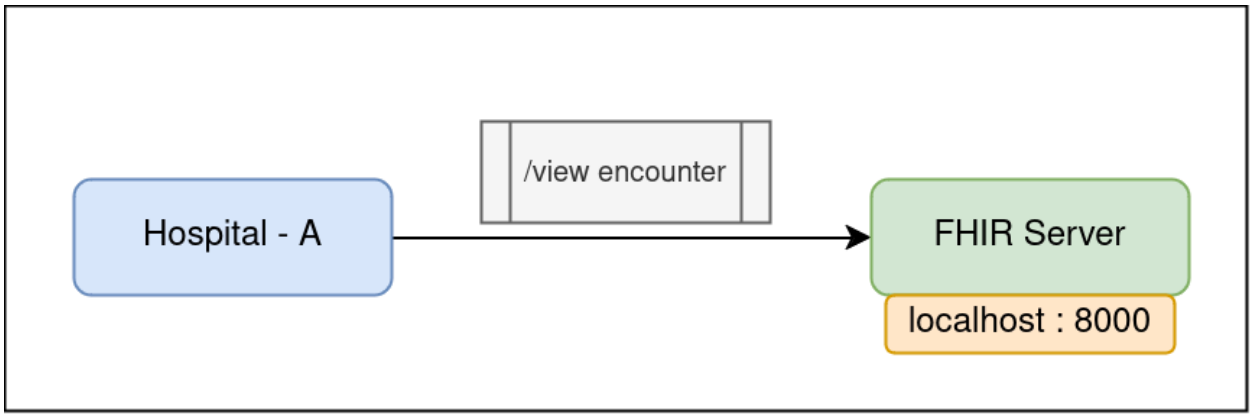


- This operation helps you to understand how to link some other resource data of FHIR to the patient.
- In this project we are taking the example of creating an Encounter and adding the details into it.
- Finally we link it to the patient we want using his ID.
- Basically, you can link the patient by entering his ID as "**Patient/<ID>**" in the referece field of JSON data.

delete_patient.html

```
var payload =
{
  "resourceType": "Encounter",
  "id": "example",
  "subject": {
    "reference": "Patient/" + patient_id
  },
  "period": {
    "start": start_time + "T00:00:00Z",
    "end": end_time + "T00:00:00Z"
  },
  "reasonCode": [
    {"coding": [{
        "system": "http://snomed.info/sct",
        "code": "123456", "display": reason
      }]
    }
  ]
};
const eurl = fhir_server_url + "/fhir/Encounter";
fetch(eurl, {method: "POST",headers: {
    "Content-Type": "application/json"
  },body: payloadString})
```

8. View Admission Details :



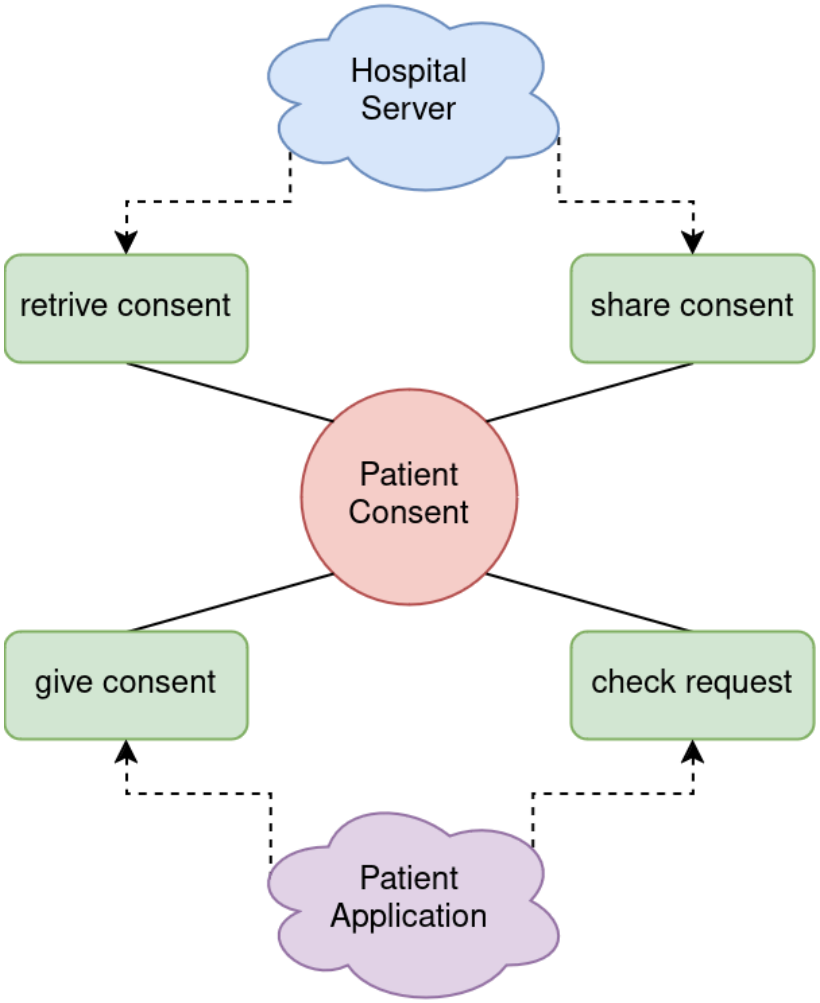
- To view admission details, we should have a patient ID having linked with any of the encounter resource.
- We can just search the encounter ID by sending the GET request to the FHIR with Patient ID as the query.

delete_patient.html

```
function getEncounterData() {
      var patientId = document.getElementById('patientId').value;
      var url = fhir_server_url + '/fhir/Encounter?patient=' +
patientId + '&_pretty=true';

      var xhr = new XMLHttpRequest();
      xhr.open('GET', url, true);
      xhr.onload = function() {
        if (xhr.status >= 200 && xhr.status < 300) {
          var responseData = JSON.parse(xhr.responseText);
          displayEncounterData(responseData);
        } else {
          document.getElementById('encounterData').innerHTML =
'Error fetching data. Status: ' + xhr.status;
        }
      };
      xhr.onerror = function() {
        document.getElementById('encounterData').innerHTML = 'Error
fetching data.';
      };
      xhr.send();
    }
```

9. Patient Consent Working

- Patient consent server as 4 main API calls.
    - **retrieve consent** : mentioned already
    - **Share-consent** : mentioned already
    - **Check-request** : will takes name of the patient as input parameters and returns hospital requests to that patient.
    - **Give-consent** : will takes name and permission as parameters and modifies data so that the hospital which is waiting for the consent will receive consent ID if permission is yes. Else receives “permission not granted”
- Patient server has data that is volatile. So its better to not turn off this server.



Patient_Consent.py

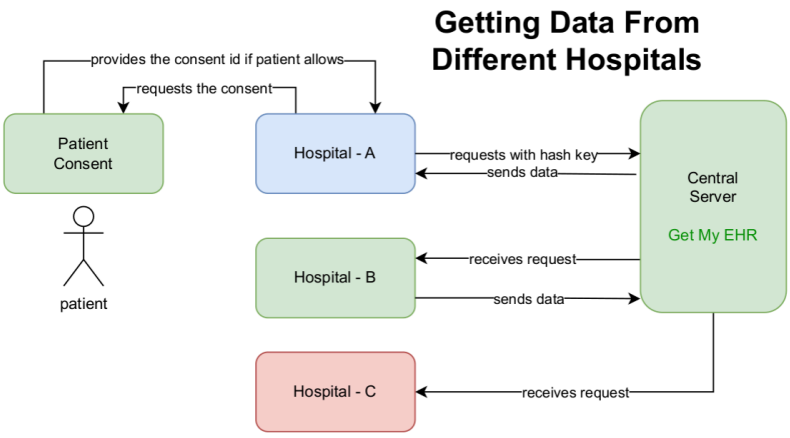
```python
from flask import Flask, jsonify, request

Patients_Data = {"patients" : []}# Data is stored here temporarily

@app.route('/check-request', methods=['GET'])
def check_requests():
    name = request.args.get("name")

    for each_patient in Patients_Data["patients"]:
        if(each_patient["name"] == name):
            return each_patient["hospital"]

    return "No requests"

@app.route('/give-consent', methods=['GET'])
def give_consent():
    name = request.args.get("name")
    permission = request.args.get("permission")
    if(permission == "0"):
        permission = 0
    else:
        permission = 1

    return update_permission(name, permission)

if __name__ == '__main__':
    app.run(port=9005, debug=True)# change here if you want
# to access this server from other system
```

Note : Patient Consent is asked 2 times.

1. When we are adding patient details we ask patient consent so that to confirm whether the hospital **can share data to other hospitals on request of the central server**.
2. When a hospital **wants data of the patient from all the other hospitals.**

Getting Data from Multiple Hospitals

- This is the most important part of this system.
- Firstly, say hospital-X wants data of a patient, then it should have already been registered in to the central server (GetMyEHR)
- hospital-X should first retrieve access token from the central server by logging in and it should also have to get Patient consent and Hash id with patient details.
- Now with access token and patient details (ONLY UNIQUE ID), we should send HTTP “GET” requests to the central server with “**/get-details**” as root.
- Central server then verifies the access token and sends the HTTP “GET” request to all the hospitals in its registry (you can see them in **accounts.json**) with “**/patient-details**” as root.
- Now each of the hospital servers fetches the name of the patient with help of lookup data (**{hash : name}** pairs).
- Now they do manual search in their FHIR servers and retrieve the data of the patient and send that data as a response to the Central server.
- Central server aggregates all the data from each hospital and sends it back as a response to the Hospital-X.



Here hospital C doesn't have consent to share the data but hospital B has.
Thus only hospital B sends data. Hospital - C sends empty message

get-data.html

```
async function fetchConsent(fname, lname, hospitalUsername) {
const purl = patient_consent_url + `/retrive-consent?name=${fname}${lname}&hospital=${hospitalUsername}`;
const response = await fetch(purl, requestOptions);
     const result = await response.text();
     return result
}

async function fetchHash(fname, lname, dob) {
const hurl = hospital_server_url + `give-me-hash?fname=${fname}&lname=${lname}&dob=${dob}`
     const response = await fetch(hurl, requestOptions);
     const result = await response.text();
     return result;
}

async function fetchRecords(dictionary) {
var url = central_server_url + "/get-details"
     url += "?patient=" + dictionary["patient"]
     url += "&id=" + dictionary["id"]
     url += "&username=" + dictionary["username"]
     url += "&access_token=" + dictionary["access_token"]
     const response = await fetch(url, requestOptions);
     const result = await response.json();
     console.log("Data : ", result);
return result;
}
```

patient_consent.py

```
@app.route('/retrive-consent', methods=['GET'])
def retrive_consent():
    name = request.args.get('name')
    hospital = request.args.get('hospital')
    print("----- RETRIVE CONSENT ------")
    print(name, hospital)

    data = get_data(name, hospital)
    return data
```

hospital_server.py

```
@app.route('/patient-details', methods=['GET'])
def patient_details():
    name = request.args.get('name')
    o_name = get_original_name(name) # gives name from Hash by looking up dictionary

    if(o_name == "Not Found"):
        return o_name

    complete_url = url + "Patient?given=" + o_name + "&_include=*&_count=5&_pretty=true"

    list_of_patients = {}
    response = requests.request("GET", complete_url, headers=headers, data=payload)
    if(response.json()["total"] > 0):
        list_of_patients = response.json()["entry"]
    else:
        print("----- No Patient Exists -----")
    return jsonify(list_of_patients)
```

central_server.py

```python
@app.route('/get-details', methods=['GET'])
def handle_request():
    name = request.args.get('patient')
    hospital_id = request.args.get('id')    # IP_ADDRESS
    username = request.args.get('username')
    token = request.args.get('access_token')
    permission_got = 0

    for each_entry in data_f :
        if(username == each_entry["username"] and token == each_entry["token"]):
            permission_got = 1
    # Send requests to all other IDs
    if(permission_got):
        print("Sending the Response :::::::: ")
        responses = send_requests_to_other_ids(hospital_id, name)
        return jsonify(responses)
    else:
        return "No Authorization. Please Login and Paste the correct token"

def send_requests_to_other_ids(current_id, name):
    responses = {}
    count = 0
    for id in hospitals:
        if str(hospitals[id][1]) != current_id:
            url = hospitals[id][1] + "/patient-details"
            try:
                response = requests.get(url, params={'name':{name}})
                responses[count] = response.json()
                count += 1
            except requests.exceptions.RequestException as e:
                responses[count] = str(e)
                count += 1
    return responses
```

IMPORTANT NOTES :

- The UI for the hospitals is present in the webspace folder.
- If you are facing errors while sending requests from the frontend then please check Links in script.js
- Please open the Server_Comm_test folder in vscode and run any html file from that folder only.
- You can see the hospital registration data in accounts.json file.
- To check backend APIs, please use Postman.
- Please make sure the last line of each file has “host=”0.0.0.0” when you are using multiple systems to test this project.
- Do not change any file (especially python files and accounts.json) while servers are running. Because they may reload entire server and servers may loose volatile information stored in them.
- Make sure you import all the libraries which are asked during execution.

Problems with our current version :

- We did not handle printing errors in the front end due to lack of time.
- Our front end doesn’t have styling and designing. We thought of the backend and the proof of concept is the main aim of the project.

Future implementation suggestions :

- There are many ideas to improve/upgrade this project.
- Patient consent can be made dynamic. This can overcome the problem of “immutable consent” where a patient can give share permission only once to a hospital adding his data. He cannot edit the permission again.
- Include more connections to patient data (as encounter).
- Patient Authentication and Hospital Authentication should be more Secure. (2 step auth is better).

Technologies Used

- **Python** : python was used for the entire backend. Whole Logic was implemented in python because python is more easy to understand.
    - **Flask :** flask was used to create and run the server on ports. Central server, patient consent server and Hospital servers were implemented with the help of flask library.
    - **Requests :** requests lib was used to send https requests (GET, POST, DELETE … etc). It was helpful in establishing communication between servers.
    - **JSON :** json lib was used to manage data in json format. JSON load and dump operations are easy when we use JSON lib.
- **Docker :** docker hub containers were used to run the HAPI JPA starter server. We used HAPI FHIR image in implementing the server.
- **Web Development :** we have created web pages for communicating with the backend.
    - **HTML :** HTML5 used to create web pages
    - **CSS :** CSS4 used to design web pages
    - **JavaScript :** JsEs14 used to add functionalities and send requests to the server.
- **App Development :** we created an app for the patient where the patient can give consent to Hospitals.
    - **React Native :** react native CLI with typescripting was used to develop applications.

THANK YOU