# Vydhya

# Software Design

## CSCI-P465/565 (Software Engineering I)

## Project Team

**<Bhanu Prakash Reddy Gaddam>**

**<Jake Sturgill>**

**<Janmejay Purohit>**

**<Keerthana Sugasi>**

---

# 1. Introduction

We have implemented a object oriented design approach for this sprint, in which the sprint task works around the enitity and their characteristics. This design strategies focuses on entities and its characteristics, class, abstraction and encapsulation around those entities.
The system is divided into smaller working solutions or components. Each component is treated as sub-system and further decomposed. when all the components are put together, it forms the whole working ssystem or application.

### 1.1 System Description

Vydhya is a Patient and Health insurance Management system which acts as a one stop solution for all the needs of doctors, Patients and Insurance providers. This is an effort to bring together all the needs of the three stakeholders and provide services for easy and best management of healthcare systems.

The application serves three different types of users.

1. Doctors
2. Patients
3. Insurance Providers

Functionalities of the system includes:

- The patients should be able to search or filter doctors and book appointments.
- The patients should be able to select health insurance plans and the application helps in providing best recommendations based on patients medical history.
- The patients should able to update their health records and view insurance details
- The doctors should be able to view patient history and appointment schedules
- The insurance providers should be able to provide best plans based on patients health records
- The doctors should be able to extend Covid-19 support to the patients and help track the cases and bed availability
- All the three stakeholders should be able communicate among themselves through chat option

For the first sprint we will be developing the User Registration and Login

### 1.2 Design Evolution

#### 1.2.1 Design Issues

No system design issues were encountered as the application is planned to built on open-source softwares and the application is divided into Backend and Frontend parts separating the User Interface and business logic. As we get into deeper implementation we might start facing issues when the needs and design evolve. The key considerations in this design is to remain open-source friendly and cloud-native.

#### 1.2.2 Candidate Design Solutions

Below are few of the design solutions:

- One of the design solutions we had come up was to adapt **Monolith Architecture Design** where all the major functionalities or components are unified into a single application. So there would a single database and different view templates for each stakeholder. The application is deployed in a server where it is accessed using REST APIs and a frontend template is created which consumes these APIs.

- Another approach was to adapt **Microservice Architecture** design. Microservice Architecture is an method that relies on a series of independent deployable services.Our application can be divided into 3 major services or components, one for each stakeholder. These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service. Also secure communication through appropriate protocols among the microservices needs to be established.

#### 1.2.3 Design Solution Rationale

Based on the requirements of the business and application, the team selected with following design solution for the following reasons:

- If the application can be divided into microservices then there is need to maintain different databases for each service. But it would be a costly affair to maintain the infratructure for 3 components and maintain same information in the three databases. Since same data needs to be used to communicate among the three services, using different databases would be expensive and unnescesary.
- We would be separating the application into backend and frontend layers since the application is going to be data heavy. Backend layers implement the business and provide REST api for the frontend. Frontend layer is separately hosted on a server which serves the requests from the users and requests appropriate API for data.
- Monolith applications are easy to debug and maintain as it is deployed as single application.

Based on the business requirements and necessity for quick delivery, the team decided to go with monolithic application with components under the same umbrella. While keeping in mind the growing business needs and if future requirements might ask for individual component scaling. Hence we have decided to go with what we call as **Microservice Ready Architecture** in which each of the three components are modularized in a way that when business need arises, the application can be converted into a microservice architecture with very minimal changes.

### 1.3 Design Approach

#### 1.3.1 Methods

The application is heavy object oriented in a way that it is divided into classes and objects. The principles of OOP such as abstraction and encapsulation is followed by grouping the related service and data together.This method also helped us integrate different functionalities in a simpler manner.

#### 1.3.2 Standards

- Naming Convention: All the classnames are in snake case and filenames follow sentence All the database names are in lowercase separated by underscores.
- Component hierarchy:The component hierarchy focuses on the component's scope of responsibility, creates meaningful relationships between components, encourages composability, and provides balance between flexibility and structure in our component library.
- Dependency injection: The basic idea is that if an object depends upon having an instance of some other object then the needed object is "injected" into the dependent object; for example, being passed a database connection as an argument to the constructor instead of creating one internally.

#### 1.3.3 Tools

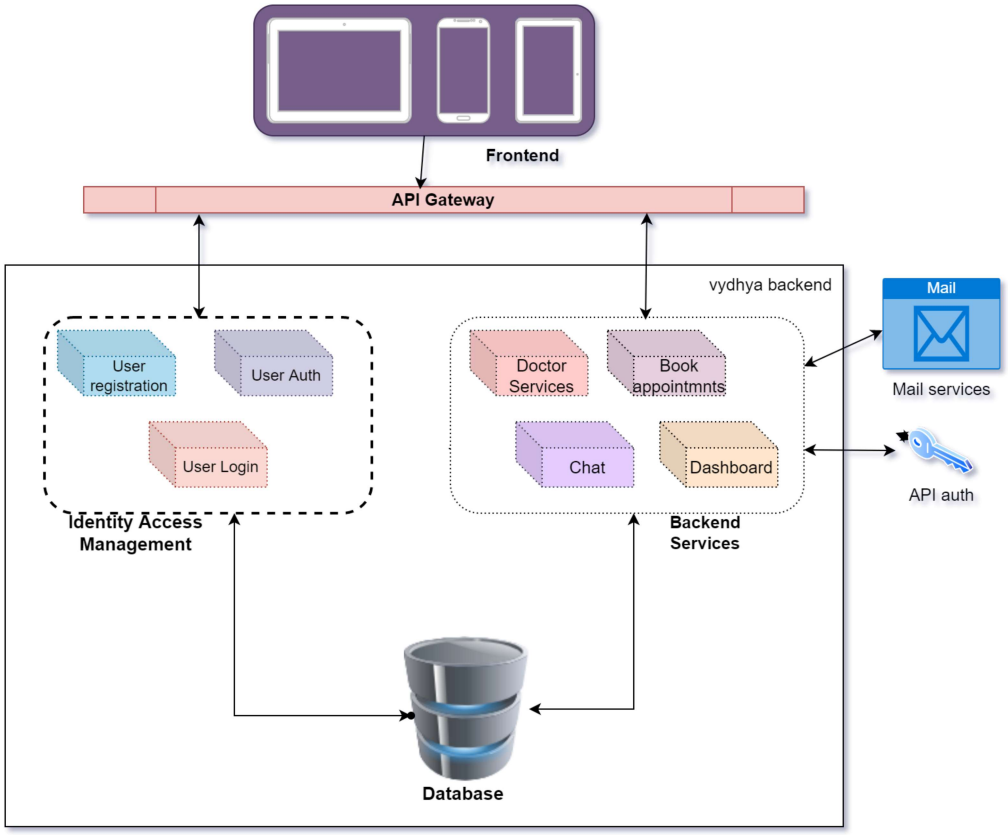
Tools that helped in design devlopment are tools like draw.io and wireframe designing tools which came in handy to visualize the workflows of the system. Docker and Kuberenetes helped in autonomous development and delivery being enviroment agnostic and cloud agnostic.

## 2. System Architecture

### 2.1 System Design

The application follows object-oriented structure where the system is divided into components and developed separately as the sprint progresses.

### High Level Design:




High Level data diagram

The high level diagram shows that there are three main components. Frontend, API Gateway and Backend. As shown in the image below the frontend serves requests from the users and passes it to API gateway wich authenticates the request and decides which component to route the user request.
The backend consists of the following components divided based on the business logic:

- Identity Access Management: This component includes the entities like User Registration, User Login after authentication, Reset Password and talks with database to insert and fetch the User Profile data.
- Database: A shared data for all the needs of the three stakeholders. It is important that the there is various inter-communication that happens between the services for each stakeholder.
- Backend Service: This includes a set of services and classes that part of the activities or flows for each stakeholders. these include Appointment Booking with Doctors, Search and filter doctors, book for insurance plans etc. All the activities are interlinked and developed in modular and extensible fashion such that future requirements can be added easily.

Going forward we might be adding more classes or components as the design evolves.

### Technologies used:

- **Backend :**A python based framework FastAPI, Python
- **Frontend :**ReactJS, HTML, CSS
- **Databases :**Postgres
- **Hosting Environment :**Kubernetes hosted on Google Cloud Platform, Docker
- **External Integrations :**Google Oauth2, Chat, Github, JIRA, Confluence

### 2.2 External Interfaces

There are external systems that work in cooperation with our application.

1. **Google OAuth2 :** A external 2-step verfication for login is integrated with Google OAuth API which is a token based authentication based on the logged in user email. The Google API provides a JSON with JWT token which is used to autheticate the users as well as the APIs.
2. **Gmail Interface :** Interface with SMTP protocol to send auto generated system emails from the applicaton like password reset, reminder emails, appointment confirmation etc.
3. **Machine Learning healthcare recommendation engine:** A machine learning which suggests the best health care plans based on the patients medical history that is fetched from the database and the model performs a simple linear regression on the data to fetch the results.
4. **Docker :** Create docker image of the application and upload to DockerHub. Docker helps to encapsulate each of them in Docker containers. Docker containers are lightweight, resource isolated environments through which you can build, maintain, ship and deploy your application.
5. **Kubernetes :**Kuberenetes helps to orchestrate the docker containers and provide independent deployments that that automatically manages, scales, and deploys the containers. Though our application is not exactly microservice based architect, this was an attempt to make the application independent of the hosting platform and cloud agnostic.
6. **Google Cloud PLatform :** Cloud based hosting environment for the application. Appplication will be hosted on a linux engine with 16GB RAM and 8 core CPU.

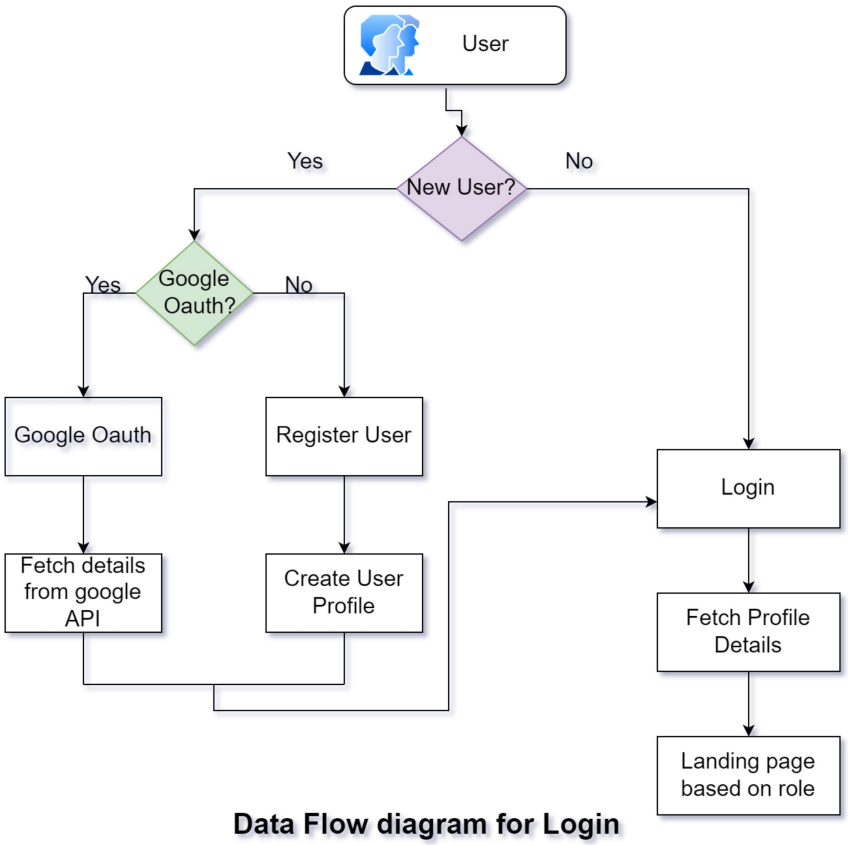
# 3. Component Design

- **Component Name**
  User Registration and Login
- **Component Description**
  There are two major components here:
  1. **User Registration:** The component is for new users. The registration page has a form which requests for details like username, password, Role and create an entry for that user in the database on clicking the submit button. In the backend, the username is an email id and is validated against Email Validation criteria. Also the password is stores as an hashed string in the database. Aftersuccesful registration user is redirected to login.
  2. **User Login:** This components allow authorized access to an registered user into the appli cation. The Login Page is a form which takes in Username and Password and compares it with the data in the database. If a matching user detail is found it logs the user into the system. Also user login maintains an authentication mechanism using JWT tokens to authenticate the user and API requests coming from frontend.

- **Responsible Development Team Member**

  ***Backend and Database*** : Keerthana and Bhanu Prakash
  ***Frontend*** : Jake and Janmejay

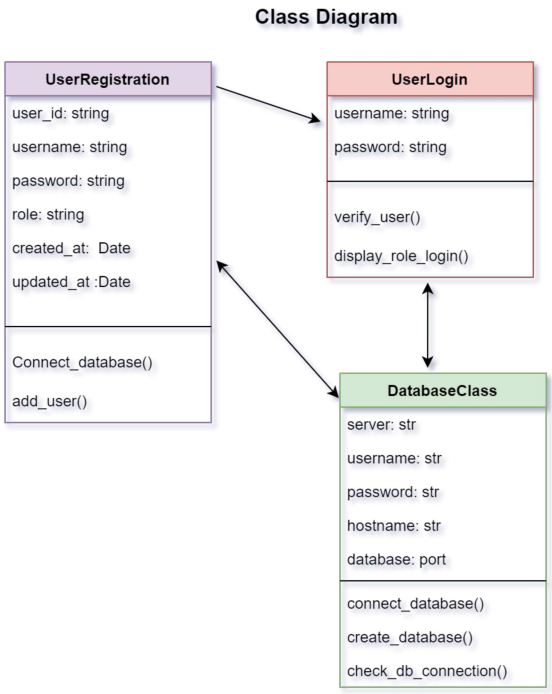
- **Component Diagram**




**Data Flow diagram for Login**

- **Component User Interface**
  The UI has 2 pages:
  - The UI is designed as a React component. User Registration page is opened when user click on New user option in the login. it directs to a form where certain details are collected and submitted to backend.
  - Another page is Login Page where the user enters the username and password. On authentication the user is permitted into the system. The landing page view is decided based on the role of the user.

- **Component Objects**

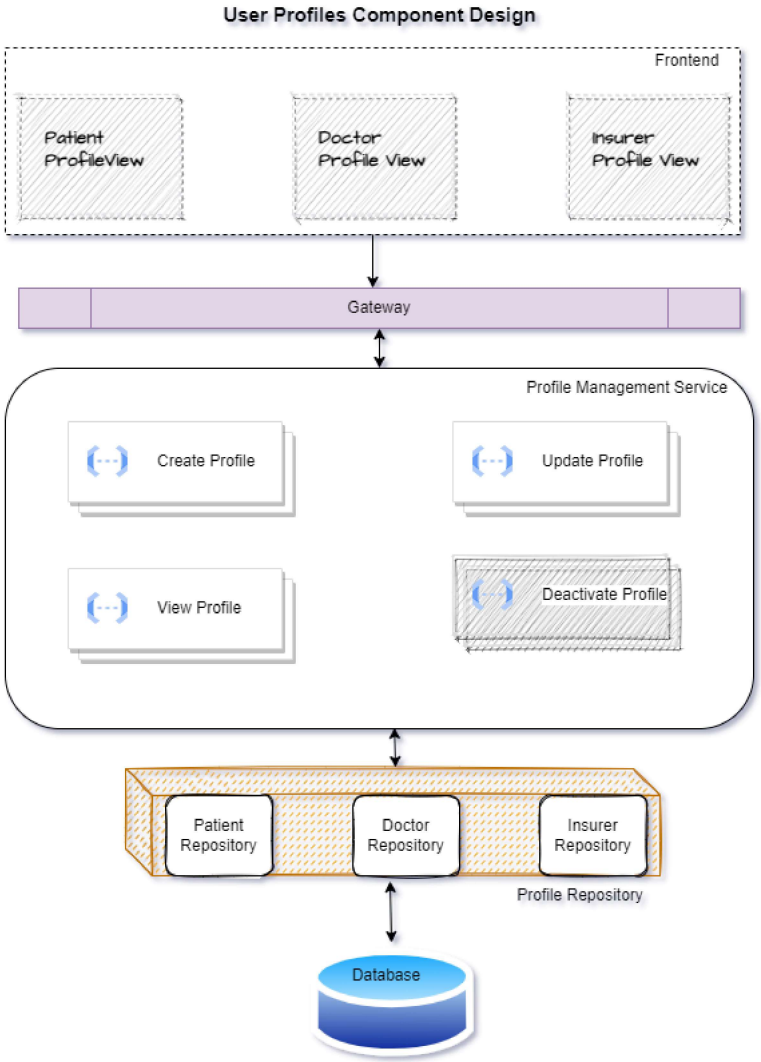



- **Component Interfaces (internal and external)**
  There is no external interfaces for this module as of now. Going forward we are planning on adding an OAuth2 login using Google OAuth2 APIs.
- **Component Error Handling**

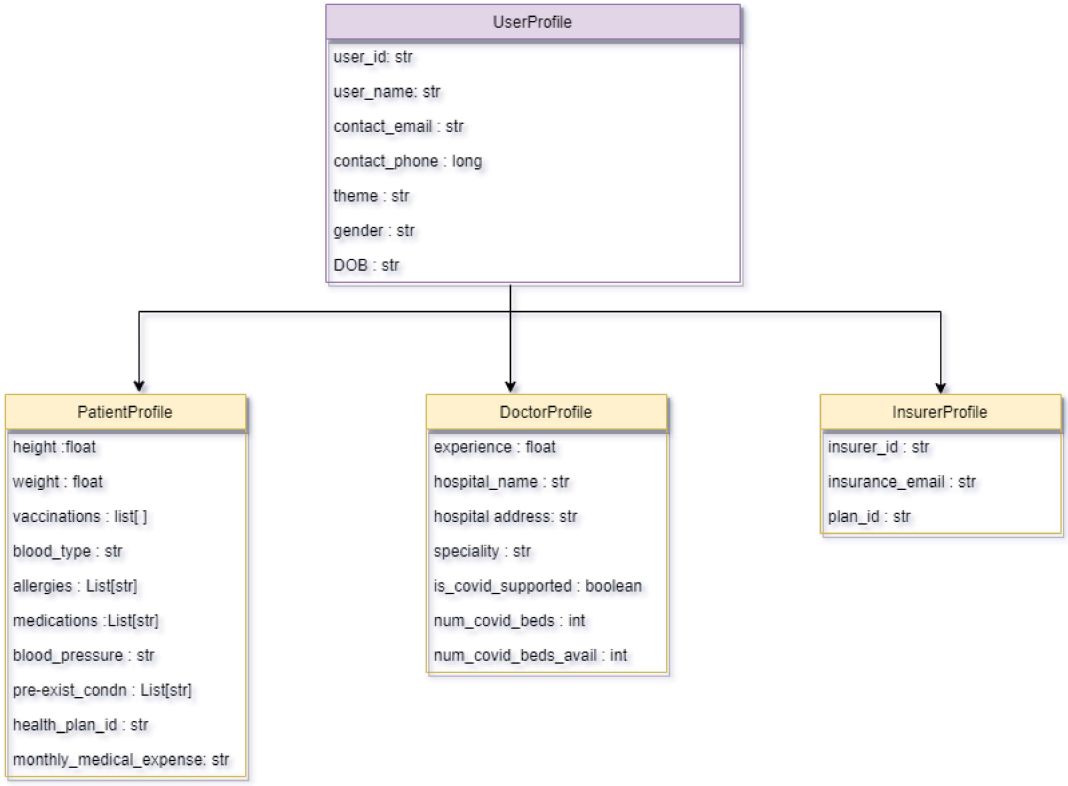
  1. User Registration:
     - Error Case 1: User registration Failed: Missing data or values
     - Error Case 2: Invalid Email / username : Format of email is invalid
     - Error Case 3: User already exists: Creating duplicate username
     - Error Case 4: Empty field: Missing required data
  2. User Login:
     - Error Case 1: Login Failed : Invalid Username or password
     - Error Case 2: Login Failed : User doesnt exist
     - Error Case 4: Empty field: Missing required data

- **Component Name**
  User Profiles Management
- **Component Description**
  This component is supposed to do the following functionalities:
  1. Create user profiles based on user_roles -> update the user profiles based on the user_id and user_role using user registration and login.
  2. View the user profiles requested using user_id and profile
  3. Update the user profiles based on role.
- **Responsible Development Team Member**
  Frontend : Jake Strugill Backend : Keerthana Sugasi

- **Component Diagram**




- **Component User Interface**
  This component has a user interface which has option to view user profile in modal. Also an edit option is provided which is a form which allows the fields to be edited. Also this existed as a common interface for all 3 user dashboards.
- **Component Objects**
  Class Diagram:




**Profile Class Diagram**

**Tasks for create profile:**

- Created a model to update the common details using details obtained from login registration
- Implemented a funcion to insert into appropriate user tables based on the role provided in the user registration service.

**Tasks for View Profile:**

- created necessary data models for 3 user roles: doctor, patient and insurer.
- created a data repository for the above models to interact with the database tables for respective roles.
- created a profile service which retrieves the profile data based on userid and user role
- Return the results in the decided response format

**Request URL:**
https://vydhya-api.herokuapp.com/profile?user_id=patient_1&user_role=patient
**Response Body:**


```
{
  "status": 200,
  "error": null,
  "data": {
    "patient": {
      "user_id": "patient_1",
      "contact_email": "user@example.com",
      "theme": "primary",
      "gender": null,
      "dob": null,
      "height": null,
      "weight": null,
      "vaccinations": null,
      "blood_type": null,
      "allergies": null,
      "medications": null,
      "blood_pressure": null,
      "preexist_conditions": null,
      "health_plan_id": null,
      "monthly_medical_expense": null
    },
    "doctor": null,
    "insurer": null
  }
}
```


**Tasks for Update Profile:**

- created necessary data models for 3 user roles: doctor, patient and insurer.
- created a data repository for the above models to interact with the database tables for respective roles.
- created a profile service which parses the request body and updates the data with the new or modified information only. It filters out the empty data from the request body.
- Make a POST call to the function above.
- Return the results in the decided response format

**Sample Request URL:**
POST: https://vydhya-api.herokuapp.com/profile?user_id=patient_1&user_role=patient
**Sample Request Body:**

```
{
  "patient": {
    "contact_email": "string",
    "theme": "string",
    "gender": "string",
    "dob": "string",
    "height": 0,
    "weight": 0,
    "vaccinations": [
      "string"
    ],
    "blood_type": "string",
    "allergies": [
      "string"
    ],
    "medications": [
      "string"
    ],
    "blood_pressure": "string",
    "preexist_conditions": [
      "string"
    ],
    "health_plan_id": "string",
    "monthly_medical_expense": "string"
  }
}
```

**Sample Response Body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "patient": {
      "user_id": "patient_1",
      "contact_email": "string",
      "theme": "string",
      "gender": "string",
      "dob": "string",
      "height": 0,
      "weight": 0,
      "vaccinations": [
        "string"
      ],
      "blood_type": "string",
      "allergies": [
        "string"
      ],
      "medications": [
        "string"
      ],
      "blood_pressure": "string",
      "preexist_conditions": [
        "string"
      ],
      "health_plan_id": "string",
      "monthly_medical_expense": "string"
    },
    "doctor": null,
    "insurer": null
  }
}
```

- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
    This component is in-house built and does not use any external 3rd party functionality or integrations.
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Invalid User_id**
     The error response format:

     ```
     {
         "status": 500,
         "error": "error while authenticating user patient_: unable to find user patient_",
         "data": null
     }
     ```

  2. **Error 2 : Invalid User Role**

     The error response format:

     ```
     {
         "status": 500,
         "error": "unsupported user_role: patient_123",
         "data": null
     }
     ```

  3. **Error 3: Wrong fields in the Request Body**

The error response format:

```
{
  "status": 500,
  "error": "field doesnt exists in request body : field_name",
  "data": null
}
```

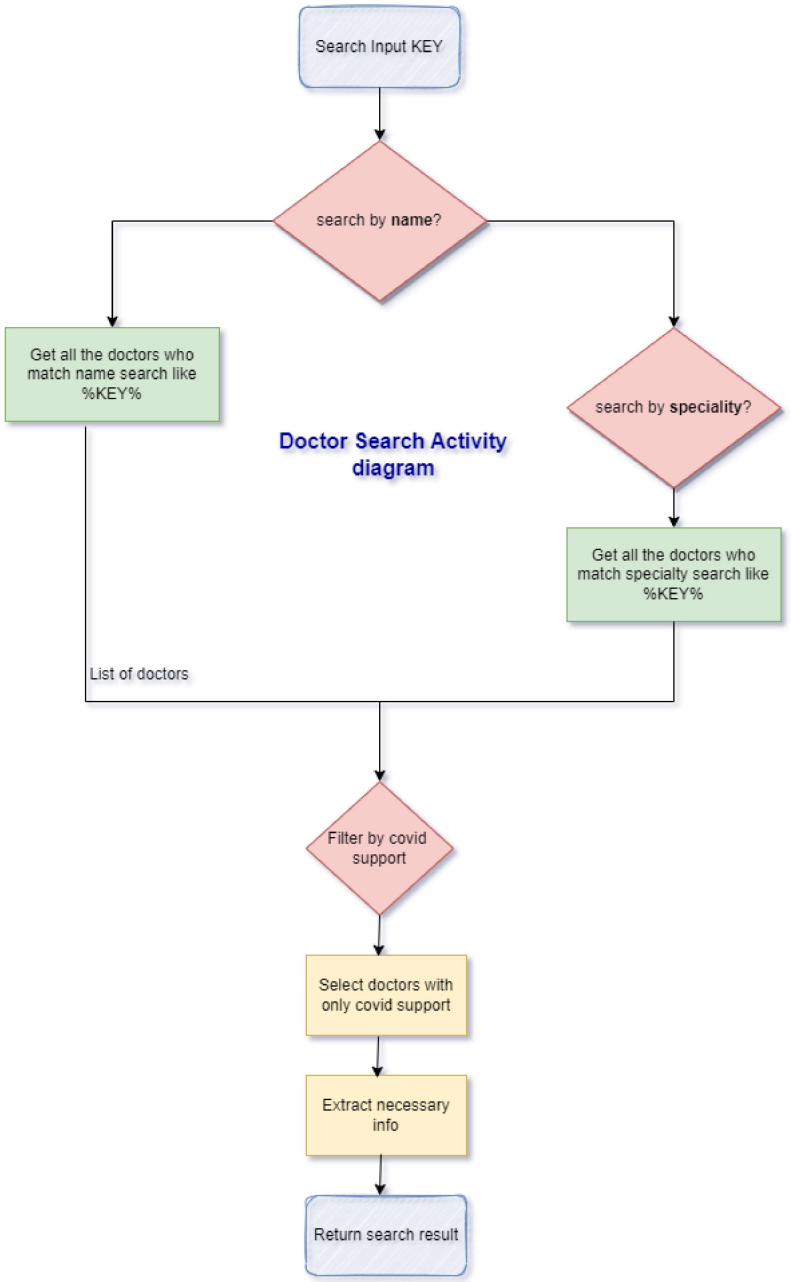
4. **Error 4: Datatype mismatch for updation of Profile**

   The error response format:

```
{
  "status": 500,
  "error": "Datatype mismatch for updation of Profile : field_name",
  "data": null
}
```

- **Component Name**
  Doctor Search and Filter
- **Component Description**
  ***Doctor Search API***
  1. Implemented search for doctors based on speciality and doctor name criteria
  2. Filters the result of the above search with covid support
  3. Return list of doctors' profiles and number of doctor profiles returned as a response
- **Responsible Development Team Member**
  Frontend: Janmejay Purohit
  Backend: Keerthana Sugasi

- **Component Diagram**




- **Component User Interface**
  This component has an User Interface which is a react hook. In the UI there will be a search bar alongside a dropdown search criteria which is by default searches by name. Currently only 2 search criterias : Doctor name and speciality is supported. On selecting the criteria, and entering the search keyword we get a list of doctors whether they are covid supported or not. If the covid support is checked then the result is filtered only with doctors working in covid supported hospitals.
- **Component Objects**
  **Tasks for Doctor search:**
  - Created data models for doctorSearchRequest and DoctorSearchResponse
  - Make a POST call with request body of search_by, search_criteria and covid_support.
  - Created a function for searching doctors based on search criteria which retrives the data using LIKE behaviour in database.
  - Return the response in predicided format which is a list of doctor profiles and total number of doctors

  **Sample Request URL:**
  POST: https://vydhya-api.herokuapp.com/doctor/search
  **Sample Request Body:**

```
{
  "search_by": "speciality",
  "search_string": "cardio",
  "covid_support": false
}
```

  **Sample Response Body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "doctor_details": [
      {
```

```
            "name": "Doctor2",
            "is_hosp_covid_supported": "0",
            "contact_email": "doctor2@iu.edu",
            "contact_phone": "435654",
            "experience": 5,
            "hospital_name": "abc hospitals",
            "hospital_address": "Bloomington Indiana",
            "speciality": "Cardiology",
            "insurance_accepted": "0",
            "gender": "Male"
          },
          {
            "name": "Doctor4",
            "is_hosp_covid_supported": "0",
            "contact_email": "doctor4@gmail.com",
            "contact_phone": "4356005413",
            "experience": 15,
            "hospital_name": "San Jose hospitals",
            "hospital_address": "San Jose, California",
            "speciality": "Interventional Cardiology",
            "insurance_accepted": "0",
            "gender": "Male"
          }
        ],
        "num_doctors": 2
      }
    }
```


- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting

  This component is in-house built and does not use any external 3rd party functionality or integrations.
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Invalid Search Criteria**
     The error response format:

     
     ```
     {
             "status": 500,
             "error": "error while searching doctors: unsupported search filter: ame",
             "data": null
     }
     ```
     

  2. **Error 2: Wrong Search word/ query**
     The return response format:

     
     ```
     {
             "status": 200,
             "error": null,
             "data": {
               "doctor_details": [],
               "num_doctors": 0
     }
     ```
     

     Returns a empty list of doctors when any doctors profiles doesnt satisfy search criteria.

- **Component Name**
  Doctor Schedule Table
- **Component Description**
  The component is supposed to be a table of availablity time of each doctor registered. This allows the patient to choose the time slot for appointment and book appointments accordingly. The table gets updated as the and shows unavailable for the times that has been booked for that doctor. Similarly when an appointment gets cancelled, the slot frees up.
- **Responsible Development Team Member**
  Backend : Bhanu Prakash
- **Component User Interface**
  The UI for this component would be to click on the tab which includes the 2 fields and a add button and a delete button. The user enter the date of availability and time frame of availability. As he clicks on add the slots for working hours of a doctor gets updated. this is shown for the current week.
- **Component Objects**
  The component consists of 3 major API:
  1. **Adding of schedule for the week** This is POST API which takes the below body and updates the database by creating a row for each line item in the json body.
  2. **Updating the schedule for the week:** This is a POST API which is used to change the start time or endtime of the slots
  3. **Deleting a entry or avilabitlity for that day:** This would again be a GET call which takes the date as input and deletes the corresponding record from the database.
  4. **Display the schedule**: This is GET call which fetches the uploaded schedule by the doctor id.
- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  Database Postgres running on Kunbernetes in Google Cloud Platform
  Swagger UI for API documentation
  Heroku for hosting
  This component is in-house built and does not use any external 3rd party functionality or integrations.
- **Component Error Handling**
  Describe the steps taken in the design to incorporate fault tolerance, data corruption prevention, and incorrect operation avoidance. Please organize these into error cases.

**Example:**
The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.

1. **Error 1: Schedule for the date doesnt exist**
   If the selected date for the delete comand doesnt exist
   The error response format:

```
{
        "status": 500,
        "error": "Delete Schedule: requested date doesnt exist for the doctor id",
        "data": null
}
```

2. **Error 2: Duplicate time updation**
   The error response format:

```
{
  "status": 500,
  "error": "Update Doctor Schedule: schedule for the slot already exists: Kindly delete and add the schedule for the day. ",
  "data": null
}
```

- **Component Name**
  Insurance Plans Upload
- **Component Description**
  The component is provides an option to the insurance providers to create their healthcare plans on the insurer view. The insurer user should be able to create a list of healthcare plans which is later chosen by the patients. The insurance company should also be able to update the plan or even delete the plan when necessary.
- **Responsible Development Team Member**
  Backend : Keerthana Sugasi
- **Component User Interface**
  The UI component opens up a modal where all the details for the new healthcare plan is added and clicked on save button. Also there is a edit button on the top right corner which allows to edit the field for that plans. And again on save button it updates the table. Similarly the insurer can delete the plan entirely by clicking on the delete button. The planid has a foreign key relationship with the patients plan_id field in patient profile.
- **Component Objects**
  The component consists of 3 major API:
  1. **Adding of plan** This is POST API which takes the below body and updates the database by creating an entry in database for that plan.
     **Request URL:**
     https://vydhya-api.herokuapp.com/insurer/plans
     **Sample Request Body:**

```
{
  "insurer_id": "insurer_5",
  "plan_name": "new_plan_1",
  "plan_display_name": "New Plan 1",
  "plan_description": "This is a new plan",
  "plan_exceptions": [
    "exception 1"
  ],
  "premium": 0,
  "coverage": 0,
  "duration_years": 0,
  "deductible_amt": 0,
  "is_monthly": true
}
```

     **Sample Response Body:**
     Returns New Plan ID generated

```
{ "status": 200,
"error": null,
"data": {
  "plan_id": "plan_insurer_5_5"
}
}
```

  2. **Updating the plan for that plain id:** This is a POST API which is used to change any of the editable fields of the plan details **Request URL:**
     https://vydhya-api.herokuapp.com/insurer/plans/update?insurer_id=insurer_5&plan_name=new_plan_10
     **Sample Request body:**

```
{
  "plan_display_name": "new plans updated 1"

}
Sample Response body:


  {
    "status": 200,
    "error": null,
    "data": {
      "plan_id": "plan_insurer_5_5",
      "insurer_id": "insurer_5",
      "plan_name": "new_plan_10",
      "plan_display_name": "new plans updated- heartcare",
      "plan_description": null,
      "plan_exceptions": [
        "exception 1"
      ],
```

```
        "premium": 0,
        "coverage": 0,
        "duration_years": 0,
        "deductible_amt": 0,
        "is_monthly": true
      }
    }
    }
```


3. **Deleting a plan:** This would again be a GET call which takes the plan id or plan name as input and deletes the plan from the table. **Request URL:** https://vydhya-api.herokuapp.com/insurer/plans?insurer_id=insurer_5&plan_name=new_plan_15
   **Sample Response Body:**


```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "successfully deleted plan new_plan_15 for insurer insurer_5"
  }
}
```


4. **Display all the plan for that insurer**: This is GET call which fetches the uploaded plans associated with that insurer id. **Request URL:** https://vydhya-api.herokuapp.com/insurer/plans?insurer_id=insurer_5
   **Sample Response Body:**


```
  {
    "status": 200,
    "error": null,
    "data": {
      "num_plans": 3,
      "message": "found 3 plans for insurer insurer_5",
      "plans": [
        {
          "plan_id": "plan_insurer_5_1",
          "insurer_id": "insurer_5",
          "plan_name": "HeartCare Plan-1",
          "plan_display_name": "Heart Care Plan",
          "plan_description": "for heart patients",
          "plan_exceptions": [
            "exception1"
          ],
          "premium": 5000,
          "coverage": 1000,
          "duration_years": 1,
          "deductible_amt": 500,
          "is_monthly": true
        },
        {
          "plan_id": "plan_insurer_5_4",
          "insurer_id": "insurer_5",
          "plan_name": "new_plan_2",
          "plan_display_name": "new display name 1",
          "plan_description": null,
          "plan_exceptions": [
            "exception 2"
          ],
          "premium": 1000,
          "coverage": 0,
          "duration_years": 0,
          "deductible_amt": 0,
          "is_monthly": true
        },
        {
          "plan_id": "plan_insurer_5_5",
          "insurer_id": "insurer_5",
          "plan_name": "new_plan_10",
          "plan_display_name": "New Plan 10",
          "plan_description": null,
          "plan_exceptions": [
            "exception 1"
          ],
          "premium": 0,
          "coverage": 0,
          "duration_years": 0,
          "deductible_amt": 0,
          "is_monthly": true
        }
      ]
    }
  }
```


- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  Database Postgres running on Kunbernetes in Google Cloud Platform
  Swagger UI for API documentation
  Heroku for hosting
  This component is in-house built and does not use any external 3rd party functionality or integrations.
- **Component Error Handling**
  Describe the steps taken in the design to incorporate fault tolerance, data corruption prevention, and incorrect operation avoidance. Please organize these into error cases.

**Example:**
The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.

1. **Error 1: Insurer id doesnt exist**
   The error response format:

```
{
         "status": 500,
         "error": "Invalid Insurer id",
         "data": null
 }
```

2. **Error 1: Plan for the date doesnt exist**
   The error response format:

```
{
        "status": 500,
        "error": "Delete Plan: requested plan doesnt exist for the insurer id",
        "data": null
}
```

3. **Error 2: Duplicate Plan exists**
   The error response format:

```
{
  "status": 500,
  "error": "Duplicate Plan: plan already exists",
  "data": null
}
```

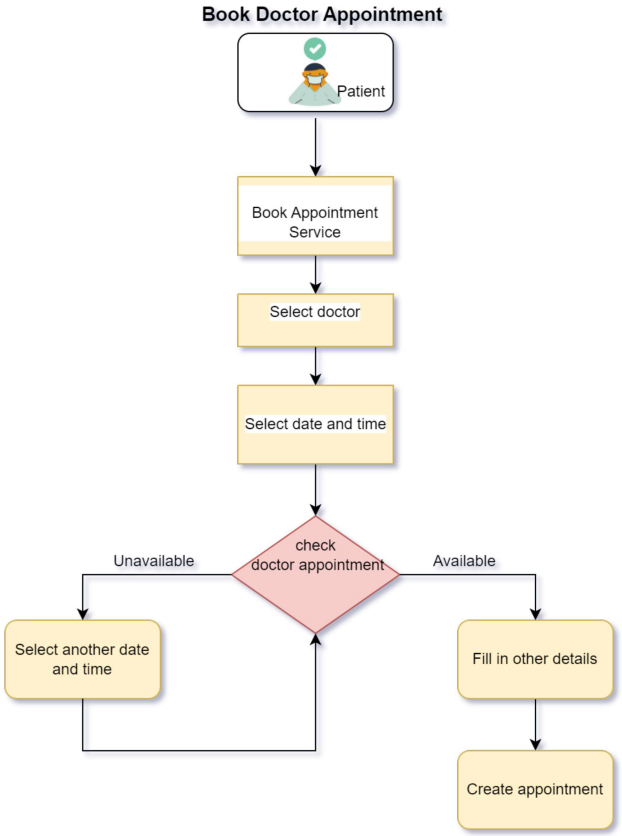
4. **Error 2: Updation failed due to insufficient data**
   The error response format:

```
{
  "status": 500,
  "error": "Updation Failed: Incorrect field or incorrect number of fields",
  "data": null
}
```

- **Component Name**
  Figma Designs for frontend
- **Component Description**
  Created various figma design for visualization of the dashboard.
  Here are some of the designs:
  **https://www.figma.com/file/t0KtF42pxxul40cceco44e/Style-Guide-%7C-Ui-Kit-(Community)?node-id=4220%3A1810**

- **Component Name**
  Doctor Appointment
- **Component Description**
  This component is a feature development which allows to create appointment for specific doctor. The user(patient) has options to add, edit and delete appointments with the doctor. The patient can book appointment between the posted schedule of the doctor on a specified day. The appointment then shows up for both doctor and patient as upcoming appointment. The patient or doctor can cancel the appointment.
- **Responsible Development Team Member**
  **Backend API** : Bhanu Prakash
  **Frontend** : Jake or Janmejay

- **Component Diagram**




- **Component User Interface**
  This component has UI which is a form to fill in patient name, time for appointment, and reason for appointment and submit the form to create an appointment. There is an option to cancel the existing appointment.
  There different views for patient and doctor profile logins. The doctor view has cards for Upcoming appointment and deails of the patient who has booked appointment. The patient view also shows the a card which has upcoming appointment along with the doctors name.
- **Component Objects**
  The following are the APIs for the Doctor appointment:
  **Add Appoinment**
  create an appoinment for a doctor by a patient.
    - **Sample Request URL:**
      https://vydhya-api.herokuapp.com/add_appointment
    - **Sample Request body:**

```
{
  "appointement_id": "string",
  "doctor_id": "doctor_1",
  "patient_id": "string",
  "appointment_start_time": "string",
  "duration": "30",
  "feedback": "string",
  "rating": 0,
  "appointment_attended": false
}
```

    - **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Appointment Added Successfully"
  }
```

  **Update Appoinment**
  Change the appointment time for the appointment by the patient
    - **Sample Request URL:**
      https://vydhya-api.herokuapp.com/update_appointment
    - **Sample Request body:**

```
{
  "doctor_id": "string",
  "patient_id": "string",
  "old_time": "string",
  "new_time": "string"
}
```

    - **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Appointment Updated Successfully"
  }
}
```

**Delete Appoinment**
Delete the appointment time.

- **Sample Request URL:**
  https://vydhya-api.herokuapp.com/delete_appointment
- **Sample Request body:**

```
{
  "doctor_id": "string",
  "patient_id": "string",
  "appointment_time": "string"
}
```

- **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Appointment Deleted Successfully"
  }
}
```

- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Doctor id doesnt exist**
     The error response format:

```
{
  "status": 500,
  "error": "Invalid doctor id",
  "data": null
}
```

  2. **Error 2: Invalid patient id**
     The error response format:

```
{
  "status": 500,
  "error": "Invalid patient id",
  "data": null
}
```

  3. **Error 3: Can book appointments only for future dates**
     The error response format:

```
{
  "status": 500,
  "error": "Failed to Add Appointment Failed to Add Appointment time data ''
 does not match format '%Y-%m-%d %H:%M'",
  "data": null
}
```

  4. **Error 3: Delete appointment: Apppntment doesnt exist**
     The error response format:

```
{
  "status": 500,
  "error": "Delete appointment: Appointment doesnt exist"
  "data": null
}
```

- **Component Name**
  Reset Password
- **Component Description**
  The feature allows the user(patient, doctor, insurer) to reset their password.
- **Responsible Development Team Member**
  **Backend API:** Keerthana
  **Frontend:** jake or janmejay
- **Component User Interface**
  This feature has Ui component which is on Login page. If the reset password is clicked, then it routes to page with send code and fields reset code and new password. there will a reset password button which will update the the userid with the new hashed password. A email is sent to the registered email address on clicking the send code button.
- **Component Objects**
  Following are the API for reset password
  1. **Send reset code:**

This is an GET API which send unique 6 digit reset code to the registered email. the reset code is valid for 15 min.

- **Sample Request** : GET : https://vydhya-api.herokuapp.com/sendresetcode?user_id=doctor_1
- **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Reset code sent to registered email keerthana271995@gmail.com successfully"
  }
}
```

2. **Update / Reset password API:**
   This is an POST API which takes the reset code and checks with the reset code stored in database. if matches it takes the new password and updates the hashed password in the user login table.
   - **Sample Request: POST:**https://vydhya-api.herokuapp.com/resetpassword
   - **Sample request body:**

```
{
  "reset_code": "string",
  "user_password": "string",
  "updated_at": "string"
}
```

   - **Sample response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Password reset successful"
  }
}
```

- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
  - Email integration using GMAIL
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Error sending reset code: Email doesnt exist or invalid email address**
     The error response format:

```
{
            "status": 500,
            "error": "Error sending reset code: Email doesnt exist or invalid email address",
            "data": null
           }
```

  2. **Error 2: Reset Code expired**
     The error response format:

```
{
       "status": 500,
       "error": "Reset code: Your reset code has expired. Please generate a new one",
       "data": null
      }
```

  3. **Error 3: Pasword reset failed**
     The error response format:

```
{
  "status": 500,
  "error": "Reset password:Your password reset has failed",
  "data": null
}
```

  4. **Error 4: Pasword reset email failed**
     The error response format:

```
{
  "status": 500,
  "error": "Reset password:unable to send email",
  "data": null
}
```

- **Component Name**
  Custom Dashboards
- **Component Description**
  1. Doctors that work in hospitals supporting COVID-19 patient care should have the provision to see if there are beds available to admit a patient who has tested positive for COVID-19 and is in a critical condition
  2. Covid Questionnaire
- **Responsible Development Team Member**
  **Backend API** : Bhanu Prakash
  **Frontend** : Jake or Janmejay

- **Component Diagram**
  Graphically depict the design of the component in terms of interfaces with other components and external interfaces. Also, consider including a diagram depicting the internal operations and/or class relationships in the component.
- **Component User Interface**
  This component has UI which is used to identify and priortize covid patients and provide immediate care for them. This component provides a form which takes in covid patient details, covid symptoms, covid test details and any reports previously taken.
- **Component Objects**
  The following are the APIs for the Covid Questionnaire
  **Add Covid Questionnaire:**
  POST request which updates the covid details for a patient.
  - **Sample Request URL:**
    https://vydhya-api.herokuapp.com/covid_questionnaire
  - **Sample Request body:**

```
{
  "user_id": "string",
  "name": "string",
  "email": "string",
  "age": 0,
  "has_cold": 0,
  "has_fever": 0,
  "has_cough": 0,
  "has_weakness": 0,
  "has_sour_throat": 0,
  "has_body_pains": 0,
  "other_symptoms": "string",
  "covid_test": 0,
  "updated_at": "string"
}
```

  - **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Covid questionnaire updated successfully!"
  }
```

  **Fetch covid details for a patient:**
  Fetch covid details for a patient if exists.
  - **Sample Request URL:**
    https://vydhya-api.herokuapp.com/get_covid_details?user_id=patient1
    **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": {
      "has_cold": 0,
      "name": "string",
      "email": "string",
      "has_cough": 0,
      "has_sour_throat": 0,
      "other_symptoms": "string",
      "updated_at": "2022-11-15 03:23:19.248255",
      "age": 0,
      "user_id": "doctor_1",
      "has_fever": 1,
      "has_weakness": 0,
      "has_body_pains": 0,
      "covid_test": 0
    }
  }
}
```

- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: update Covid questionnaire: Wrong data value**
     The error response format:

```
{
            "status": 500,
            "error": " Update Covid questionnaire: The field has diffrent datatype. Wrong data given",
            "data": null
           }
```

  2. **Error 2: Covid Details for the userid doesnt exist**
     The error response format:

```
{
  "status": 500,
  "error": "Covid Questionnaire: Covid Details for the userid doesnt exist",
```

```
  "data": null
}
```

- **Component Name**
  Recommendation and Statistics
- **Component Description**
  1. The insurance providers should have the option to provide various insurance packages to the patients
  2. A default package for general patients. More advanced packages for patients spending more than usual on medical expenses.
  3. Machine learning can be used to provide recommendations to the users based on their past data.
  4. Patients should be able to subscribe to an insurance provider that would inform the patient about new/discounted insurance packages as the insurance provider publishes.
- **Responsible Development Team Member**
  **Backend API** : Keerthana
  **Frontend** : Jake or Janmejay
- **Component Diagram**
- **Component User Interface**
  This component has UI which has a list of healthcare plans displayed . The user can click on the enroll button to enroll into the plan. This POST request updates the plan id in the patient profile. The result can be fetched using get patient profile request.
- **Component Objects**
  The following are the APIs for the Enroll Plan
  **Enroll healthcare plan for user:**
  A POST request which enroll a patient into healthcare plan.
    - **Sample Request URL:**
      https://vydhya-api.herokuapp.com/enroll_health_plan?patient_id=patient_1&plan_id=1000
    - **Sample Response body:**

```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Enrolled in plan 1000 successfully"
  }
}
```

- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
  - Netlify for hosting frontend
  - Gmail for sending emails to users
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Error while enrolling into plan : The user id**
     The error response format:

```
{
"status": 500,
"error": " Error while enrolling into plan : Plan id doesnt exist ",
"data": null
}
```

- **Component Name**
  Patient Feedback for doctors
- **Component Description**
  1. Patients who have booked an appointment with doctor should be able to give feedback to the doctor and service. It will take comments and rating for the doctor.
  2. A general feedback form that allows patients to post feedback comments and ratings for doctors in general with or without appoinments.
- **Responsible Development Team Member**
  **Backend API** : Keerthana
  **Frontend** : Jake or Janmejay
- **Component User Interface**
  This component is part of Appointment scheduling feature. After attending an appoinment a patient can click on Give feedback button to add comments and rating for the doctor or rather the appointment. Also there's a list of comments in the feedbacks section displayed which is general feedback.
- **Component Objects**
  The following are the APIs for the Feedback section
  **add feedback comment and rating for an appointment:**
  A POST request which adds feedback.
    - **Sample Request URL:**
      https://vydhya-api.herokuapp.com/update_feedback_by_appointment?appoinment_id=ecd4dbfd-de6d-43e2-b6fe-5ed00cd3b9cc
    - **Sample Request body:**

```
{
  "feedback": "Excellent service",
  "rating": 1
}
```

    - **Sample Response body:**

```
{
  "status": 200,
```

```
  "error": null,
  "data": {
    "message": "updated feedback for doctor appointment id =ecd4dbfd-de6d-43e2-b6fe-5ed00cd3b9cc"
  }
}
```


General feedback APIS:

- **API for adding feedbacks to doctors:**
  Adding general feedback for doctors with or without appointment.
- **Sample API Request** :
  https://vydhya-api.herokuapp.com/addFeedback?doctor_id=doctor_1&patient_id=patient_2
- **Sample Request Body:**


```
{
  "feedback": "Good service",
  "rating": 4.9
}
```


- **Sample Response Body:**


```
{
  "status": 200,
  "error": null,
  "data": {
    "message": "Feedback for doctor doctor_1 added successfully"
  }
}
```


**Get request to fetch all the feedbacks by doctor_id**

- **Sample API request:**

```
https://vydhya-api.herokuapp.com/get_feedback_by_doctor?doctor_id=doctor_1
```

- **Sample Response body:**


```
{
  "status": 200,
  "error": null,
  "data": {
    "doctor_1": [
      {
        "patient_id": "patient_2",
        "feedback": "Excellent service",
        "rating": 3.5,
        "submitted_at": "22-11-26 05:16:06"
      },
      {
        "patient_id": "patient_3",
        "feedback": "good",
        "rating": 5,
        "submitted_at": "22-11-26 05:44:26"
      },
      {
        "patient_id": "patient_2",
        "feedback": "Good service",
        "rating": 4.9,
        "submitted_at": "22-11-29 09:37:08"
      }
    ],
    "total rating ": 4.466666666666667
  }
```


- **Component Interfaces (internal and external)**
  This component is interfaced with the following services:
  - Database Postgres running on Kunbernetes in Google Cloud Platform
  - Swagger UI for API documentation
  - Heroku for hosting
  - Netlify for hosting frontend
  - Gmail for sending emails to users
- **Component Error Handling**
  The services in the component will raise appropriate HTTP error responses if any unnatural behavior is encountered.
  1. **Error 1: Error while adding feedback: The appointment was not attended to provide feedback**
     The error response format:


```
{
"status": 500,
"error": " Error while adding feedback: The appointment was not attended to provide feedback ",
"data": null
}
```

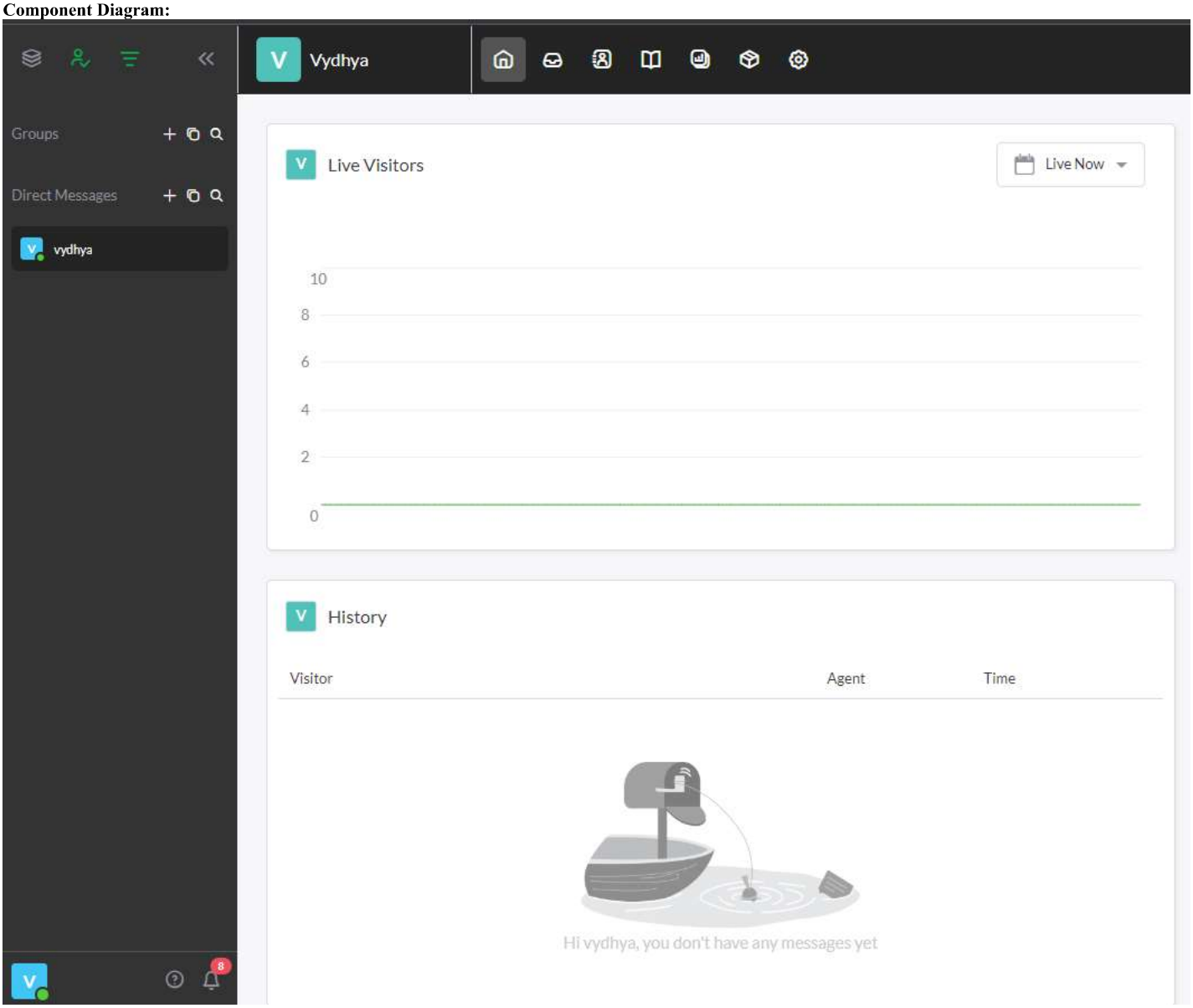

  2. **Error 2: Error while adding feedback: Mismatch datatype in the data entry**
     The error response format:

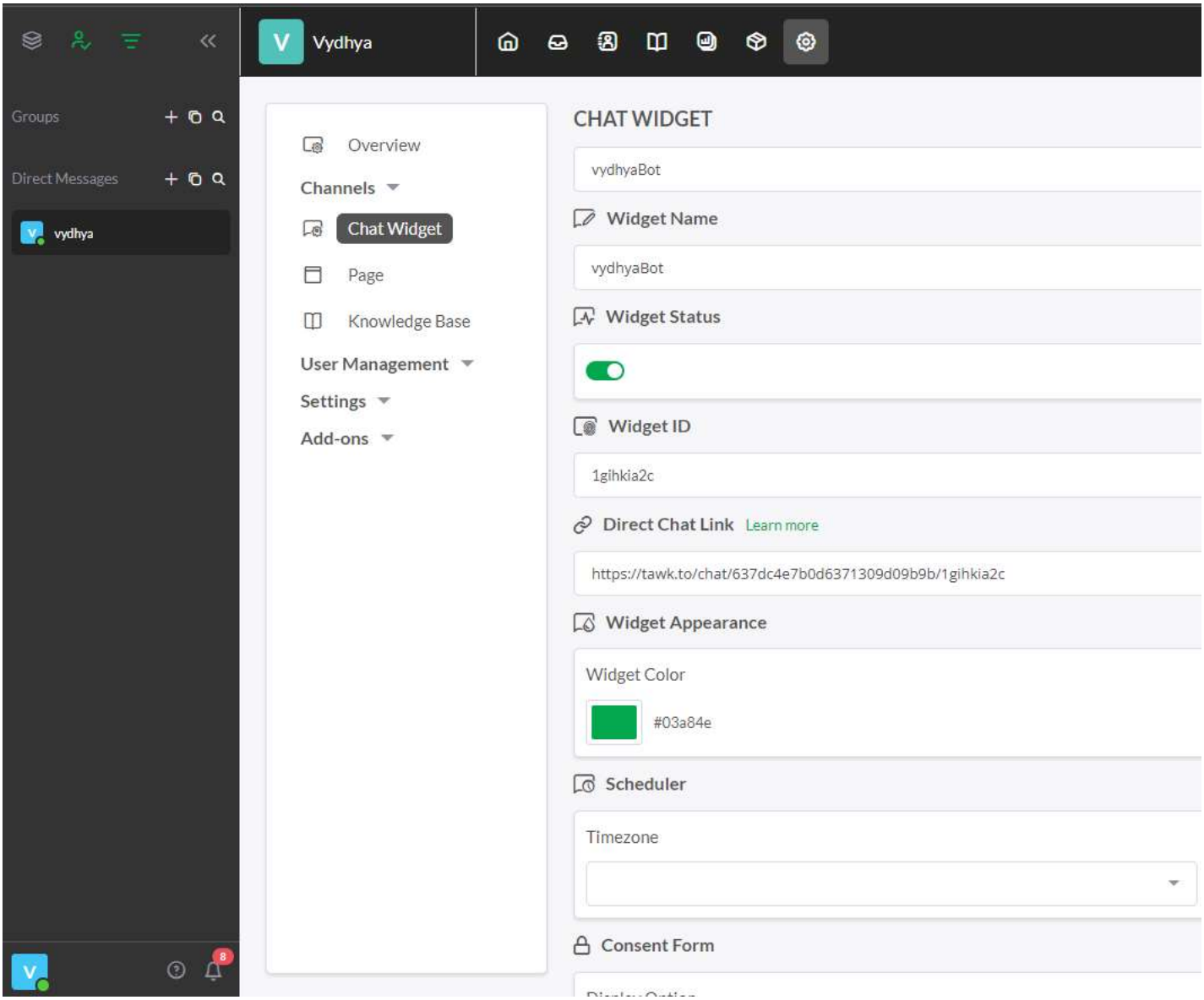

```
{
"status": 500,
"error": " Error while adding feedback: Mismatch datatype in the data entry",
"data": null
}
```

3. **Error 3: Error while adding feedback: When wrong doctor id is given. it returns empty list**
   The error response format:

```
{
  "status": 200,
  "error": null,
  "data": {
    "doctor_id": [],
    "total rating ": null
  }
}
```

- **Component Name**
  Chat
- **Component Description**
  1. The patients can chat with any insurance provider to get details about the insurance plan.
  2. There should be an option for group chat between all doctors.
  3. The chat should have the following features:
     i. Private chat feature
     ii. Online/offline status.
     iii. Message status, delivered/undelivered.
     iv. Display â€œtypingâ€ when the other user is typing
- **Component Diagram:**






- **Component Interfaces Integration:**
  For this feature we have integrated tawk.io chat widget whcih is a chat widget added to the frontend. It can be added by embedding a javascript code snippet. Different group can be added and patient and doctors can converse with each other even after appointment ends.

## Revision History

| Revision | Date | Change Description |
|---|---|---|
| User Login Component | 10/18/2022 | Added Reset Password functionality |
| Profile Management | 10/18/2022 | Added create, view and update user profiles for all 3 user roles |
| Doctor Search | 10/18/2022 | Feature to search the doctors based on speciality and name. Filtered out based on Covid support by doctor and hospital. |
| Figma Designs | 11/1/2022 | Created Figma designs |
| Doctor Schedule | 11/1/2022 | Creating the schedule for doctors |
| Insurer Plan management | 11/1/2022 | CRUD operation for healthcare plans |
| Reset Password | 11/15/2022 | Reset password with email feature |
| Doctor Appointment | 11/15/2022 | Create, update and delete doctor appointments |
| Covid Questionnaire | 11/15/2022 | Form to provide special care to covid 19 patients |
| Feedback for doctors | 11/29/2022 | Added feedback capturing feature for doctors |
| Recommednation and Statistics | 11/29/2022 | Added statistics display of bills, amount claimed,choose best health care plans |
| Enroll in plans by patients | 11/29/2022 | Allows patients to choose the best healthcare plan and enroll in them |
| Chat | 11/29/2022 | Chat integration with tawk.io |
| Additional Features | 11/29/2022 | Added covid bed availability and booking them. |
| | | |

---

**Page Author/Creator:** Keerthana Sugasi

**Last Modified:11/29/2022**