JSPM University Pune

Hackathon – Low-Level Design

On

ORGO-BOT

Problem Statement ID: 3
Problem Statement Title: End-to-End Text Generation Pipeline
Domain: Python, Ai Frameworks, MERN Stack

<u>Team members</u>:

Team Leader Name: Samruddhi Rikesh Patil**.**
Branch (B.Tech) Stream (CSE) Year (III)

Team Member 1 Name: Vaishnavi Umesh Shinde.
Branch (B.Tech) Stream (CSE) Year (III)

Team Member 2 Name: Arfat Mushtaque Haju.
Branch (B.Tech) Stream (CSE) Year (III)

Team Member 3 Name: Kunal Ashok Sapkale.
Branch (B.Tech) Stream (CSE) Year (III)

Table of Contents

1. Introduction

1.1 Scope of the document:

This project's scope involves enhancing an existing organ donation application by integrating a GPT -3 based AI chatbot. The AI-powered bot will be designed to provide users with real-time, accurate, and informative text responses regarding organ donation. The application will feature a user-friendly chat interface, allowing users to interact with the bot seamlessly. The project will be built on the MERN stack (MongoDB, Express.js, React.js, Node.js) and aims to improve user engagement by offering a more interactive and informative experience.

1.2 Intended Audience:

The intended audience for the organ donation application with an AI-powered bot includes:

- **Potential Organ Donors**: Individuals interested in learning about organ donation and considering becoming donors.
- **Medical Professionals**: Healthcare providers looking for reliable information to guide their patients.
- **General Public**: Anyone seeking to understand the organ donation process and its benefits.
- **Developers and AI Enthusiasts**: Individuals interested in AI applications in healthcare and web development.
- **Non-profit Organizations**: Groups focused on promoting organ donation awareness and education.

1.3 System Overview

A system overview of an organ donation application will include various components working together to facilitate the process of organ donation and transplantation:

User Interface (UI) - provides an interface for users to interact with the system.

Medical Records Integration - Ensures healthcare providers have access to up-to-date patient data for informed decision-making.

Backend: Built using Node.js and Express.js for server-side logic and API handling.

Frontend: Developed with React.js, providing a dynamic and interactive user interface.

Database: MongoDB is used to store and manage user data, chatbot interactions, and organ donation information.

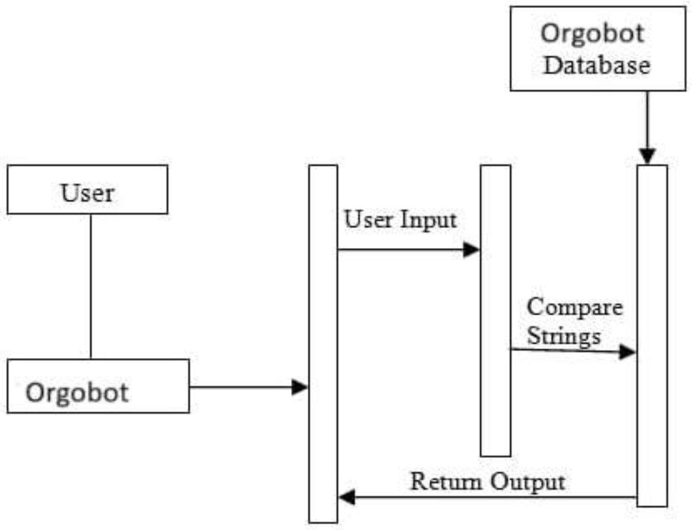
AI Integration: GPT-3 is integrated to power the chatbot, offering real-time, informative text responses related to organ donation.

Deployment: The entire system is deployed on a MERN stack-based platform for seamless performance and scalability.

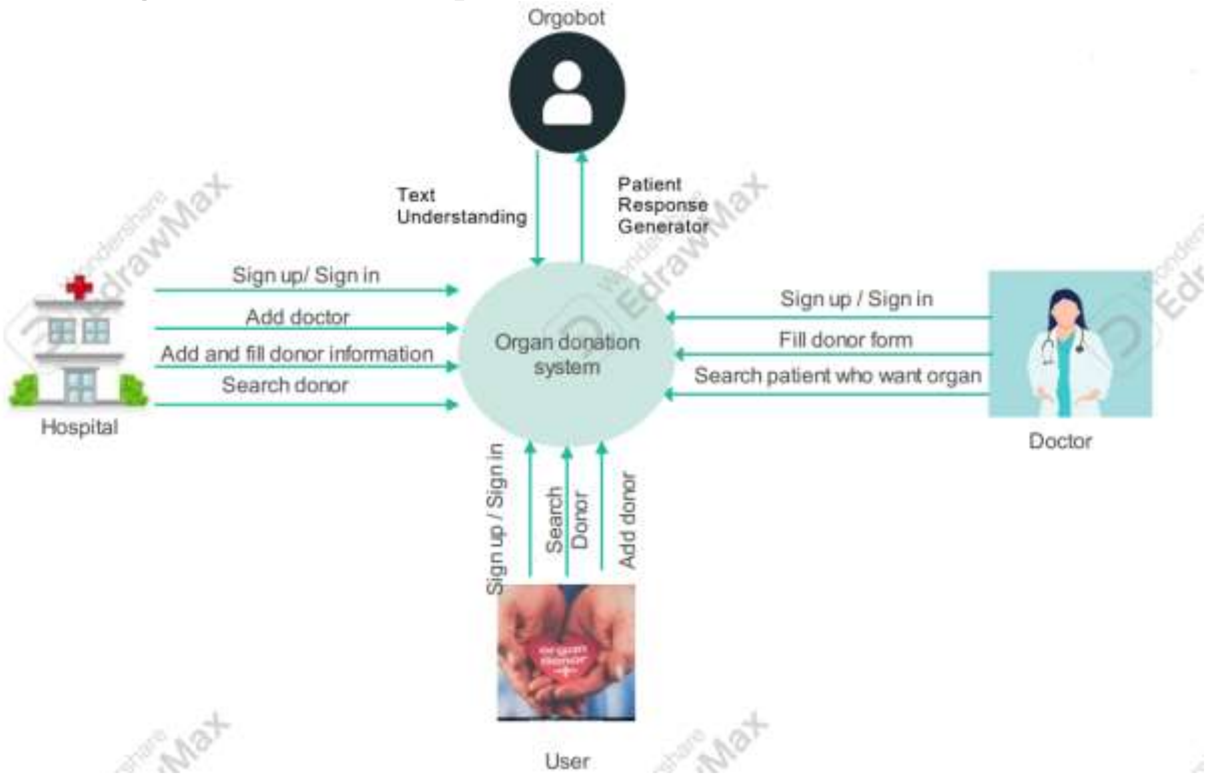
By integrating these components into a cohesive system architecture, an organ donation application can effectively facilitate the matching of donors and recipients, streamline communication among stakeholders, and ultimately save lives by increasing the availability of organs for transplantation.

2. Low Level System Design

2.1 Sequence Diagram



2.2 Navigation Flow/UI Implementation



2.3 Screen Validations, Defaults and Attributes

In an organ donation application, screen validations, defaults, and attributes play a crucial role in ensuring data accuracy, user-friendliness, and adherence to regulatory requirements. Here's how these aspects could be implemented across various screens:

Screen Validations:

- **Input Validation:** Ensure users enter valid data (e.g., correct email format, valid phone numbers) before submitting forms. Real-time validation can notify users of errors as they type.

- **Mandatory Fields:** Critical fields such as name, contact information, and organ type must be filled out before submission. If left blank, users are prompted to complete these fields.

- **Conditional Logic:** Some fields may only appear based on previous selections (e.g., if the user selects "Organ Donor," additional fields related to donation history may appear).

- **Character Limits:** Implement maximum character limits on text fields like name, address, or comments to ensure data consistency and prevent overflow.

Defaults:

- **Pre-filled Fields:** Automatically populate fields such as country or state based on the user's location data, reducing input time.

- **Default Values:** For dropdown menus or radio buttons, set default selections (e.g., "Select Organ Type" defaults to "Kidney") to guide users and streamline the process.

- **Auto-Save:** Enable auto-save for forms, allowing users to return later without losing input data.

Attributes:

- **Field Labels and Instructions:** Clearly labeled input fields with concise descriptions help users understand what information is required. Tooltips or inline help messages can provide additional guidance for complex fields.

- **Responsive Design:** Ensure that all screens are optimized for various devices (desktop, tablet, mobile), maintaining usability and readability across different screen sizes.

- **Accessibility Features:** Incorporate attributes such as ARIA labels for screen readers, high-contrast color schemes, and keyboard navigation support to make the application accessible to all users.

- **Error Messages:** Display clear, user-friendly error messages that specify what went wrong and how to fix it (e.g., "Please enter a valid email address" instead of just "Invalid input").

- **Loading Indicators:** Use loading spinners or progress bars when processing data or interacting with the AI bot to keep users informed and reduce perceived wait times.

These elements together ensure a user-friendly, efficient, and accessible experience for all users of the organ donation application.

2.4 Client-Side Validation Implementation

1. HTML5 Validation:

- Use `required` attributes for key fields (e.g., name, email, organ type).
- Apply `type="email"` for email fields and `type="number"` for numeric inputs like age.

2. JavaScript Validation:

- Implement custom scripts to validate complex fields (e.g., checking if the age is within a valid range).
- Ensure real-time validation with instant feedback, such as green borders for valid inputs.

3. React Validation Libraries:

- Utilize **Formik** or **Yup** in the React framework to manage and validate form state effectively.

4. User Feedback:

- Display validation errors dynamically near the affected fields, with clear instructions for correction.
- Provide tooltips or inline help to guide users on proper input formats.

5. Accessibility:

- Implement ARIA attributes to ensure error messages and validation statuses are accessible to screen readers, providing an inclusive experience.

2.5 Server-Side Validation Implementation

1. **Server Setup:**
   - Utilized Node.js and Express.js to handle server-side logic.
   - Configured body-parser middleware to parse incoming JSON data from HTTP requests.
2. **Input Validation Middleware:**
   - Employed the Joi validation library to enforce input validation rules.
   - Defined a validation schema with the following requirements:
     - **Prompt:** String, minimum length of 10 characters, required.
     - **Model:** Must be "gpt-2" or "gpt-3", required.
     - **Domain:** Must be one of "legal", "healthcare", or "creative", required.
3. **Middleware Application:**
   - Applied the validation middleware to relevant routes (e.g., /generate-text) to ensure input data meets the specified criteria before processing.
4. **Error Handling:**
   - Implemented error handling that returns a 400 Bad Request status with a descriptive message if validation fails.
   - Example error message: "prompt" length must be at least 10 characters long.
5. **Considerations:**
   - **Data Sanitization:** Additional sanitization is recommended to prevent security issues like Cross-Site Scripting (XSS).
   - **Asynchronous Validation:** For scenarios requiring external validation (e.g., checking uniqueness in a database), asynchronous validation techniques can be incorporated.

2.6 Components Design Implementation

. Data Collection Component

- **Purpose**: Gather and preprocess the data required to train the text generation model.
- **Design**:
    - **Data Sources**: Identify data sources (e.g., text corpora, domain-specific datasets).
    - **Data Pipeline**: Implement scripts or services to fetch and preprocess data.
        - **Preprocessing**: Tokenization, removing stop words, handling special characters, and domain-specific processing.
- **Implementation**:
    - Use Python libraries like `Pandas`, `NLTK`, or `spaCy` for preprocessing.
    - Store processed data in a format suitable for model training, such as CSV or JSON.
    - Optionally, use databases like MongoDB to store large datasets.

2. Model Fine-Tuning Component

- **Purpose**: Fine-tune a pre-trained language model like GPT-2 or GPT-3 on the domain-specific data.
- **Design**:
    - **Model Selection**: Choose between GPT-2 or GPT-3 based on the task complexity and resource availability.
    - **Fine-Tuning Process**:
        - Load pre-trained model.
        - Adjust hyperparameters (learning rate, batch size).
        - Implement early stopping to avoid overfitting.
- **Implementation**:
    - Use libraries like `Transformers` by Hugging Face for model fine-tuning.
    - Utilize GPUs for faster training.
    - Save the fine-tuned model for deployment.

3. API & Backend Component (MERN Stack)

- **Purpose**: Create an API that allows users to input text prompts and receive generated text.
- **Design**:
  - **Backend**: Implement a Node.js server using Express.js to handle API requests.
  - **Endpoints**:
    - `POST /generate-text`: Accepts a text prompt and returns generated text.
    - `GET /status`: Check the server's health.
  - **Database**: Use MongoDB to store user input, generated texts, and logs for future reference or analysis.
- **Implementation**:
  - Implement API routes in Express.js.
  - Use Mongoose for MongoDB interaction.
  - Implement input validation and error handling.

4. Frontend Component

- **Purpose**: Provide a user interface (UI) for interacting with the text generation API.
- **Design**:
  - **UI Framework**: Use React for building the frontend.
  - **User Interaction**:
    - Input field for the text prompt.
    - Display generated text results.
    - Optionally, allow users to download the generated text or save it for later use.
- **Implementation**:
  - Implement components in React.
  - Use `Axios` or `Fetch` API for making API calls to the backend.
  - Style the UI using CSS or a framework like Bootstrap.

5. Deployment Component

- **Purpose**: Deploy the application for users to access over the web.
    - **Hosting**: Choose cloud platforms like AWS, Heroku, or Google Cloud for hosting.
    - **CI/CD Pipeline**: Set up continuous integration and deployment pipelines for seamless updates.
    - **Scalability**: Implement auto-scaling and load balancing if necessary.
- **Implementation**:
    - Containerize the application using Docker.
    - Use services like AWS Elastic Beanstalk or Heroku for deployment.
    - Set up a CI/CD pipeline using GitHub Actions, Jenkins, or similar tools.

6. Optimization Component

- **Purpose**: Optimize the model for specific domains (legal, healthcare, creative writing).
- **Design**:
    - **Domain-Specific Tuning**: Adjust the model for the vocabulary, style, and tone of each domain.
    - **Performance Metrics**: Define and measure metrics like coherence, relevance, and fluency.
- **Implementation**:
    - Implement domain-specific preprocessing and fine-tuning scripts.
    - Use tools like TensorBoard for tracking model performance.
    - Optimize model hyperparameters based on feedback from domain experts.

7. Testing and Validation Component

- **Purpose**: Ensure the quality and reliability of the pipeline.
- **Design**:
    - **Unit Testing**: Test individual components (data collection, model, API).
    - **Integration Testing**: Test the end-to-end flow.
    - **User Acceptance Testing (UAT)**: Involve domain experts for validating the model output.
- **Implementation**:

- Use testing frameworks like `Jest` for frontend and `Mocha` for backend.
- Implement automated testing scripts.
- Gather feedback and iteratively improve the model.

8. Logging and Monitoring Component

- **Purpose**: Monitor the application's performance and ensure smooth operation.
- **Design**:
  - **Logging**: Capture logs for API requests, errors, and model performance.
  - **Monitoring**: Set up real-time monitoring and alerts.
- **Implementation**:
  - Use services like AWS CloudWatch or ELK Stack (Elasticsearch, Logstash, Kibana) for monitoring.
  - Implement logging middleware in Express.js.

2.7 Configurations/Settings

- **Hardware Requirement**
  1) Processor: Ryzen 5.
  2) RAM: 8 GB or above.
  3) Hard Disk: Minimum 5GB
  4) Operating System: win 8, win 10, and win 11

- **Software Requirement:**

1. Development Environment

- **Languages**: Python, JavaScript (Node.js, React.js)
- **IDE/Editors**: Visual Studio Code, PyCharm

2. Data Collection & Preprocessing

- **Python Libraries**: Pandas, NLTK/spaCy
- **Database**: MongoDB

3. Model Fine-Tuning

- **Libraries**: Hugging Face Transformers, PyTorch/TensorFlow
- **GPU Support**: CUDA, cuDNN

4. Backend API (MERN Stack)

- **Node.js**, **Express.js**
- **Mongoose** (for MongoDB)
- **JWT** (for authentication)

5. Frontend Development

- **React.js**
- **Axios** (for API calls)
- **Bootstrap**/Material-UI (for styling)

6. Deployment

- **Docker** (for containerization)
- **Cloud Platforms**: AWS, Heroku

- **CI/CD**: GitHub Actions

7. Testing & Validation

- **Testing**: Jest, Mocha, pytest
- **Linting**: ESLint, Pylint

8. Monitoring & Logging

- **Monitoring**: Prometheus, Grafana
- **Logging**: Winston, ELK Stack

9. Version Control

- **Git** and **GitHub/GitLab/Bitbucket**

2.8 Interfaces to other components

1. **Electronic Health Records (EHR):**
   - Connect with hospitals' computer systems to get patients' medical info.

2. **Organ Procurement Organizations (OPOs) and Transplant Centres:**
   - Talk to places that handle organ donations and transplants to share info about donors and recipients.

3. **National or Regional Organ Procurement and Transplantation Networks:**
   - Share data with bigger groups that oversee organ donation and transplants.

4. **Donor Registries and Health Information Exchanges:**
   - Sync info with databases that store details about organ donors and recipients.

5. **Notification Services:**
   - Send messages to users about organ availability, matches, and appointments.

6. **Identity Verification Services:**
   - Make sure users are who they say they are when they sign up or do important stuff.

   - Share data for research and track how well the organ donation system is working.

7. **Public Health Agencies and Registries:**
   - Share data with government groups and other organizations to keep track of organ donation trends and make policies.

3. Data design

3.1 List of Key Schemas/Tables in database

1.Users Collection:	**2.Donors Collection**	**3.Recipents Collection**	**4.Donation Collections**	**5.Messages Collection (for Communication):**	**6.Notification Collection:**
userID	donorID	recipentID	donationID	messageID	notificationID
Email	userID	userID	donorID	senderID	userID
Username	Name	Organ needed	Organ donated	recipientID	notification content
Password	Age	Name, age	Donation status	message content	timestamp
Other user profile information	Blood type	Contact information	Donation date/time	timestamp	notification status (read/unread)
	Medical history			message status (read/unread)	
	Consent status				

7.Settings Collection:	**8.Requests Collection:**	**9.Logs Collection (for Auditing and Analytics):**
userID (reference to Users)	requestID	logID
notification preferences	userID	userID
privacy settings	request type (e.g., organ donation request, transplant request)	action performed
other user-specific settings	request details	timestamp

3.2 Details of access levels on key tables in scope

1. User Table

- **Description**: Stores user information, including authentication credentials, profile details, and user roles.
- **Access Levels**:
  - **Admin**: Full access (CRUD - Create, Read, Update, Delete).
  - **Authenticated User**: Read access to their own profile; Update access to certain fields like password, profile information.
  - **Unauthenticated User**: No access.

2. Prompts Table

- **Description**: Stores the text prompts provided by users for text generation.
- **Access Levels**:
  - **Admin**: Full access (CRUD).
  - **Authenticated User**: Create and Read access for their own prompts; Update/Delete access for their own prompts.
  - **Unauthenticated User**: No access.

3. Generated Text Table

- **Description**: Stores the output generated by the AI model based on the user's prompts.
- **Access Levels**:
  - **Admin**: Full access (CRUD).
  - **Authenticated User**: Read access to their own generated texts; Create access linked to new prompts; Update/Delete access to their own generated texts.
  - **Unauthenticated User**: No access.

4. Model Configurations Table

- **Description**: Stores configurations and hyperparameters used for model fine-tuning and deployment.
- **Access Levels**:
  - **Admin**: Full access (CRUD).
  - **Data Scientist/Engineer**: Read and Update access; no Delete or Create access unless explicitly required.
  - **Authenticated User**: No access.

- **Unauthenticated User**: No access.

5. Logs Table

- **Description**: Stores logs related to API usage, errors, and system events.
- **Access Levels**:
    - **Admin**: Full access (CRUD).
    - **Developer/Support**: Read access to monitor system health and debug issues.
    - **Authenticated User**: No access.
    - **Unauthenticated User**: No access.

6. API Keys Table

- **Description**: Stores API keys for accessing different parts of the system (if applicable).
- **Access Levels**:
    - **Admin**: Full access (CRUD).
    - **Developer**: Read and Create access; no Update/Delete access.
    - **Authenticated User**: No access.
    - **Unauthenticated User**: No access.

7. Access Control Table

- **Description**: Manages user roles and permissions across the system.
- **Access Levels**:
    - **Admin**: Full access (CRUD).
    - **Security Officer**: Read and Update access for managing permissions.
    - **Authenticated User**: No access.
    - **Unauthenticated User**: No access.

8. Feedback/Rating Table

- **Description**: Stores feedback or ratings provided by users regarding the generated text or overall system performance.
- **Access Levels**:
    - **Admin**: Full access (CRUD).
    - **Authenticated User**: Create access to provide feedback; Read/Update/Delete access to their own feedback.
    - **Unauthenticated User**: No access.

9. Domain-Specific Data Table

- **Description**: Stores domain-specific data used for fine-tuning the AI model (e.g., legal texts, healthcare data).
- **Access Levels**:
  - **Admin**: Full access (CRUD).
  - **Data Scientist/Engineer**: Read and Update access.
  - **Authenticated User**: No access.
  - **Unauthenticated User**: No access.

Notes on Access Management:

- **Role-Based Access Control (RBAC)**: Implementing RBAC will help manage access to these tables based on user roles.
- **Logging & Monitoring**: All CRUD operations, especially for sensitive tables like User, API Keys, and Model Configurations, should be logged for audit purposes.
- **Least Privilege Principle**: Users should only have access to the tables and operations necessary for their role.

4. Details of other frameworks being used

4.1 Session management

Session management in an organ donation application involves handling user authentication, maintaining user sessions, and ensuring security throughout the user's interaction with the application. Here's how session management can be implemented in an organ donation application:

1. **User Authentication:**
    - Securely verify user identity during login.
2. **Session Creation:**
    - Generate a unique session ID upon successful authentication.
3. **Session Tracking:**
    - Monitor user activity and manage session lifetimes.
4. **Session Persistence:**
    - Allow users to remain logged in across sessions.
5. **Session Validation:**
    - Verify session data for each request to prevent unauthorized access.
6. **Session Revocation:**
    - Provide logout mechanisms and revoke sessions when necessary.
7. **Security Measures:**
    - Use HTTPS, secure cookies, and regular audits for security.
8. **Session Timeout Handling:**
    - Define session timeout intervals and log out idle users.
9. **Error Handling and Logging:**
    - Handle errors gracefully and log session-related events.

4.2 Caching

1. **Donor and Recipient Profiles:**
    - Cache donor and recipient profiles to reduce database queries when retrieving user information.
    - Update the cache when profiles are created, updated, or deleted to ensure data consistency.

2. **Organ Availability and Matching Data:**
    - Cache information about available organs and potential matches between donors and recipients.
    - Update the cache in real-time as new organ donations are registered and matches are identified.

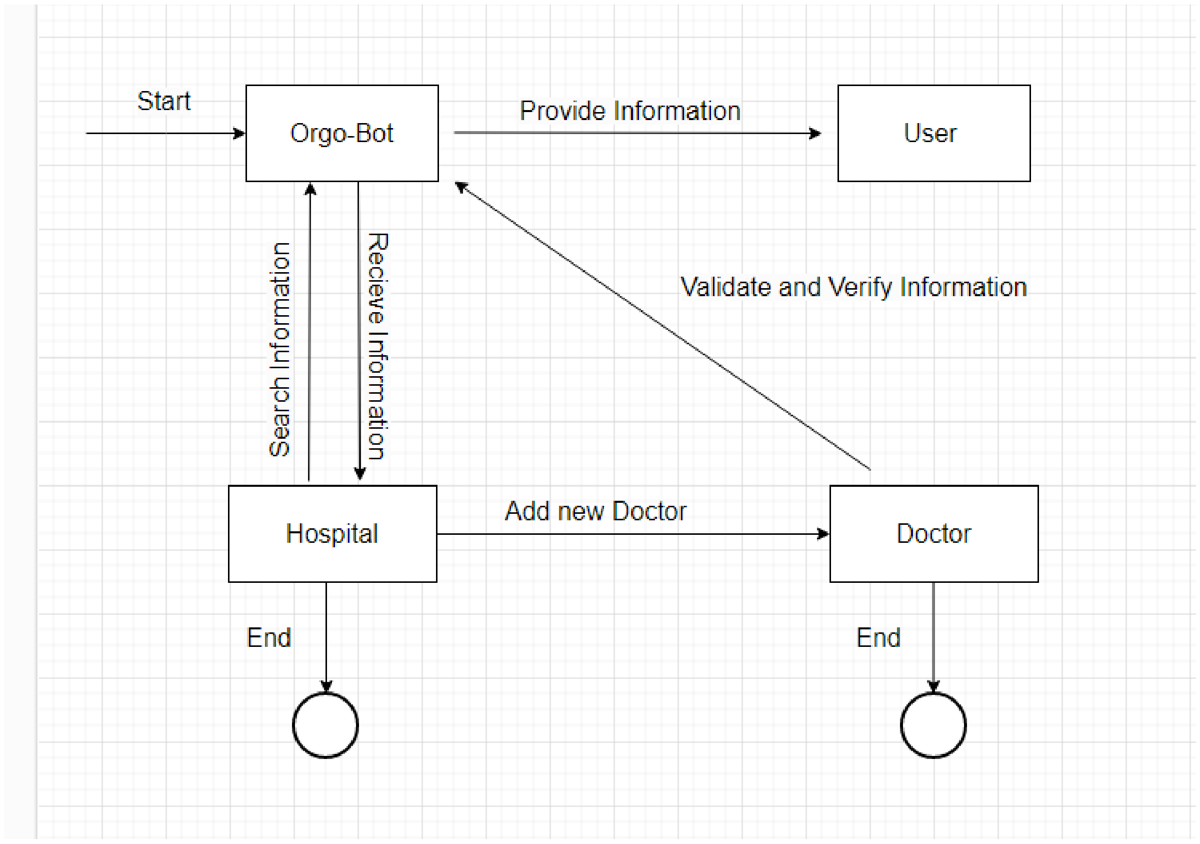
3. **Session Data and User Preferences:**
    - Cache session data and user preferences to personalize user experiences and reduce database queries for user-specific information.
    - Invalidate cached session data when users log out or their preferences are updated.

4. **Error Responses and Exception Handling:**
    - Cache error responses and exception data to minimize the impact of transient errors and improve error handling.

5. **Security Tokens and Authentication Data:**
    - Cache security tokens and authentication data to optimize user authentication and authorization processes.

5. Unit Testing

Unit testing in an organ donation application involves testing individual units or components of the application in isolation to ensure they function correctly. Here's how unit testing can be applied in such an application:



6.Key notes

• **Enhanced User Experience:**
The integration of a GPT-3 AI bot provides real-time, informative, and engaging interactions for users seeking organ donation information.

• **Advanced Validation:**
The app employs robust client-side validation, ensuring accurate and complete user data while offering immediate feedback.

• **Tech Stack:**
Developed using the MERN stack, the application is scalable and adaptable, with seamless AI integration for future upgrades.

• **Accessibility Focus:**
Ensures inclusivity with accessible design and screen reader support, making the app user-friendly for all audiences.

7. References

1. "Donate Life: What you need to know about organ donation in India". [The better India]. 22 November 2016. Retrieved 7 may 2021.
2. "Narayana Nethralaya- Dr. Rajkumar Eye Bank". Having achieved from the original journal.
3. "Strategies and outcome in renal transplant". Vivek B. Kute. Published on 2022 march25.
4. "Health e-living blog: register as an organ donor". [Chester county hospital]. 02 April 2021.
5. "How to register". Transplant Quebec. Retrieved November 26, 2019

Books:

- Head First Java - Kathy Sierra & Bert Bates
- Android Programming for Beginners - John Horton [2nd edition]
- Headfirst Android Development – Dawn Griffiths [1st edition]