**Pratham Trivedi**
**DAIICT**

# CareCompass

## A MERN based web application which aims to help you find nearby Hospitals

# 1. Introduction

The "Care Compass" project was constructed as part of Intel Unnati training program aimed at developing a web application to facilitate seamless connectivity between individuals in need of medical care and registered hospitals. The project stems from a recognized need to streamline the process of hospital search and information retrieval for users.

The '**WHY**'

As the 'information' increases, the filter between corrent and wrong blurs, and with a plethora of misinformation, its hard to find correct recommendations, be it a service or knowledge.

CareCompass tries to solve this issue by providing a collection of Hospitals, filtered on users wishes, from registered hospitals and from Google Api. So the people can find the desired Hospital with a tap of a finger.

The '**HOW**'

The primary objective of the "Care Comfort" project is to develop a robust web application that enables users to easily discover and access detailed information about hospitals. Key goals include implementing secure user authentication via JWT, integrating hospital data from both a local database and the Google Places API, and providing a seamless user experience through intuitive search and filtering functionalities.

The scope of the project encompasses the development of core features such as user registration and login, hospital data integration using Google Places API, search functionality based on location or city, and detailed hospital profiles. Additionally, the project aims to lay the groundwork for future enhancements, including functionality for booking appointments, saving favorite hospitals, and reviewing hospital experiences.

## 2. Framework and Key Concepts:

The theoretical framework for "CareCompass" revolves around user-centered design principles and efficient data integration methodologies. Key concepts include:

Understanding required technologies:

The development of "CareCompass" utilizes key technologies including JavaScript (JS), React for frontend development, and Node.js with Express.js for backend development. These technologies are chosen for their robustness, scalability, and ability to facilitate rapid development and deployment of web applications.

Data Integration :

Utilizes the Google Places API alongside a local database to merge comprehensive hospital information, ensuring accurate and up-to-date data retrieval for users.

Security and Authentication :

Implementing JWT (JSON Web Token) authentication ensures secure access to user and hospital functionalities while safeguarding sensitive data.

Location Based Service:

Leveraging location data (latitude and longitude) to provide personalized search results and enhance user convenience in locating nearby hospitals.

# 3. Requirement Analysis:

## Functional Requirements:

User Management:

Registration**:** Allow users to create accounts with unique usernames and email addresses.

Authentication**:** Provide login/logout functionality using JWT for secure access.

Profile Management: Enable users to update their profiles, including personal information and preferences.

Password Management: Allow users to reset their passwords securely.

Hospital Search and Listing:

Search Hospitals**:** Allow users to search for hospitals based on location (city or current location).

Filtering: Provide options to filter search results by hospital speciality, ratings, and other relevant criteria.

View Hospital Details**:** Display detailed information about each hospital, including contact details, operating hours, and specialties.

Save Favorites(Incomplete):

Save Hospitals: Enable users to save hospitals to their profile for quick access later.

**Non-Functional Requirements:**

Performance:

Response Time: Ensure quick response times for search queries and data retrieval from the database.

Scalability: Design the system to handle increasing numbers of users and hospital listings without significant degradation in performance.

Security:

Data Protection: Implement encryption (e.g., data hashing) to protect user data and secure API interactions.

Authentication**:** Use JWT for secure user authentication and authorization.

Reliability:

Availability**:** Ensure high availability of the application to minimize downtime.

Data Integrity: Implement backup and recovery mechanisms to protect against data loss.

Usability:

User Interface: Design an intuitive and user-friendly interface for easy navigation and interaction.

Responsive**:** Design should be displayed well on any device

# 4. System Design:

### 1. Database and data

a. User Model (Schema):

Attributes:

- `id`: String, unique identifier for each user.
- `email`: String, unique email address for user identification.
- `username`: String, unique username for user identification.
- `name`: String, user's full name.
- `password`: String, hashed password for user authentication.

- `createdAt`: DateTime, timestamp of user registration.

```
model Review {
  id         String   @id @default(auto()) @map("_id") @db.ObjectId
  userId     String   @db.ObjectId
  hospitalId String   @db.ObjectId
  rating     Int
  ratingText String
  user       User     @relation(fields: [userId], references: [id])
  hospital   Hospital  @relation(fields: [hospitalId], references: [id])

  @@index([userId])
  @@index([hospitalId])
}
```

Hospital Model (Schema):

Attributes**:**

- `id`: String, unique identifier for each hospital.
- `name`: String, name of the hospital.
- `images`: Array of Strings, URLs for hospital images.
- `address`: String, address of the hospital.
- `city`: String, city where the hospital is located.

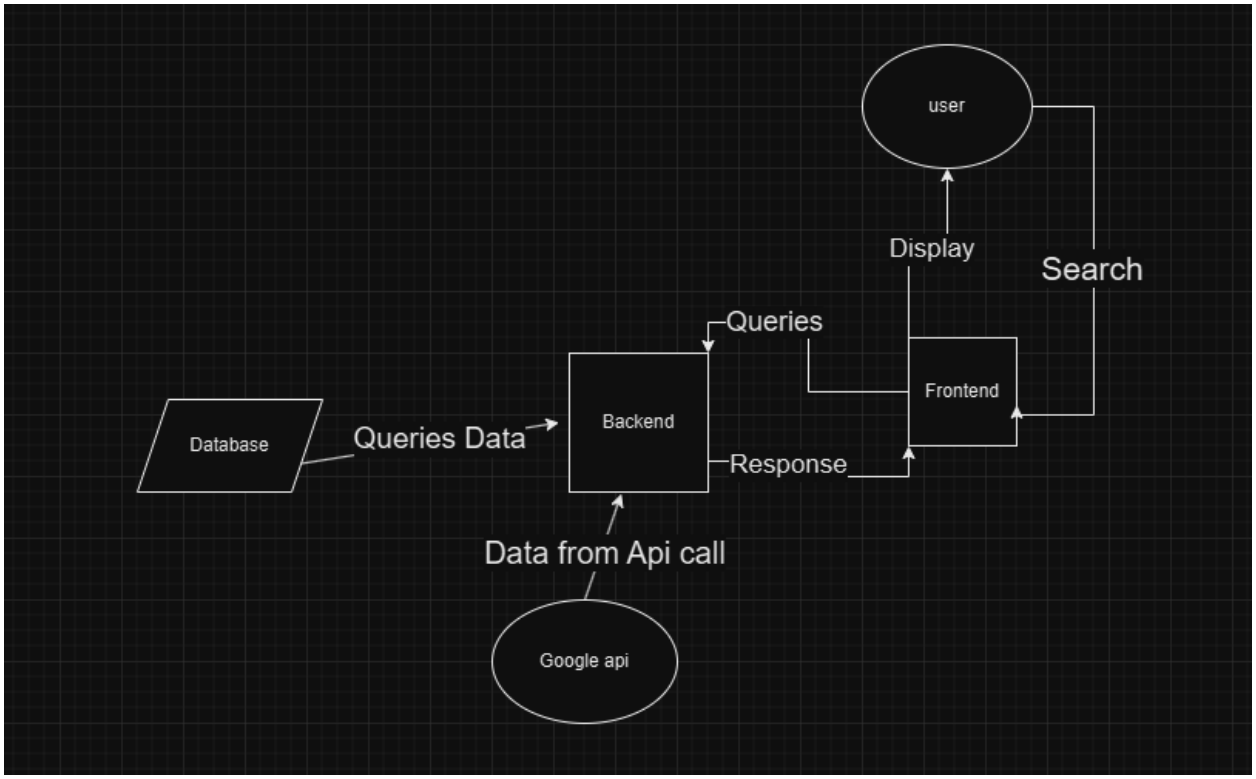
- `latitude`: String, latitude coordinate of the hospital's location.
- `longitude`: String, longitude coordinate of the hospital's location.
- `rating`: Int, average rating of the hospital based on user reviews.
- `email`: String, contact email address for the hospital.
- `phone`: String, contact phone number for the hospital.
- `opentime`: String, opening time of the hospital.
- `closetime`: String, closing time of the hospital.
- `speciality`: Array of Speciality Enum, list of medical specialties provided by the hospital.
- `createdAt`: DateTime, timestamp of hospital registration.

```
model Hospital {
  id          String      @id @default(auto()) @map("_id") @db.ObjectId
  name        String
  images      String[]
  address     String      @unique
  city        String
  latitude    String
  longitude   String
  rating      Int     @default(0)
  email       String      @unique
  phone       String
  opentime    String
  closetime   String
  speciality Speciality[]
  createdAt   DateTime    @default(now())
  reviews     Review[]
}
```

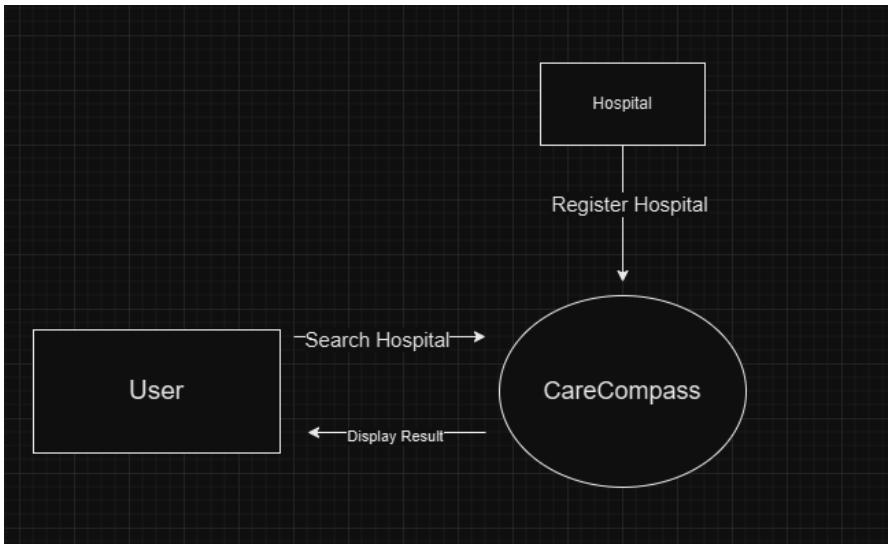
More Data are fetched from Google Places Api[1] which is directly fetched from backend and displayed.

As more planned features are added, the complexity of database will increase.

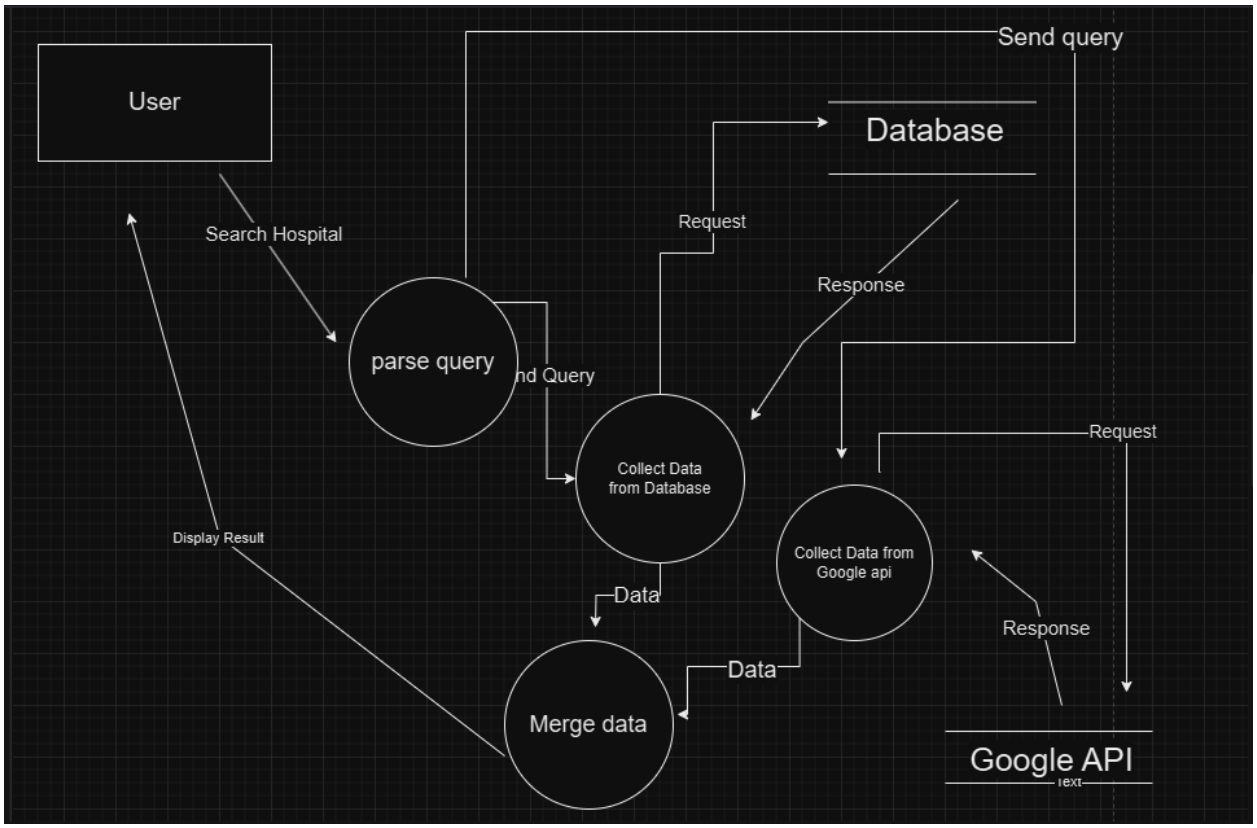
Basic structure of data:



## Level 0 Data Flow Diagram:

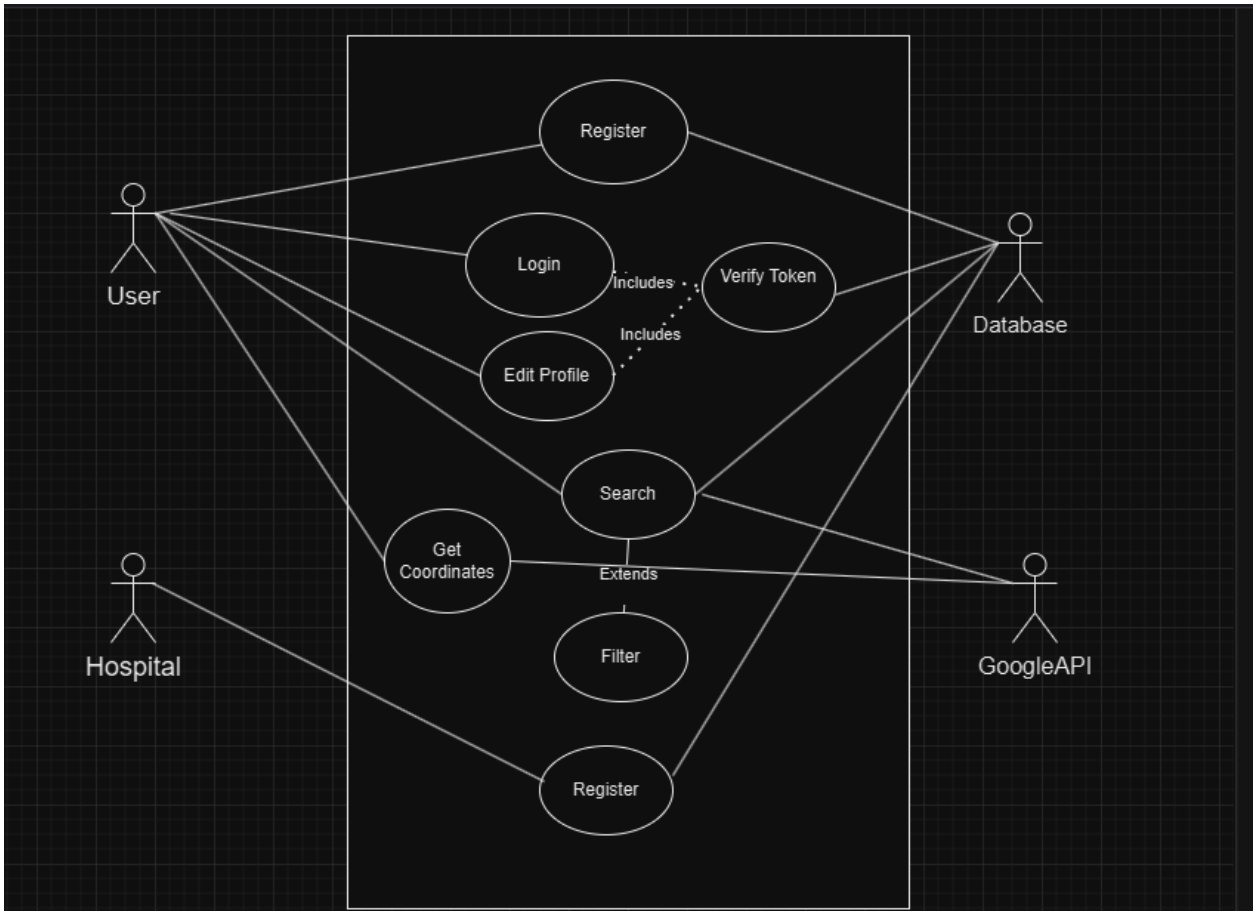



## Level 1 Data Flow Diagram:

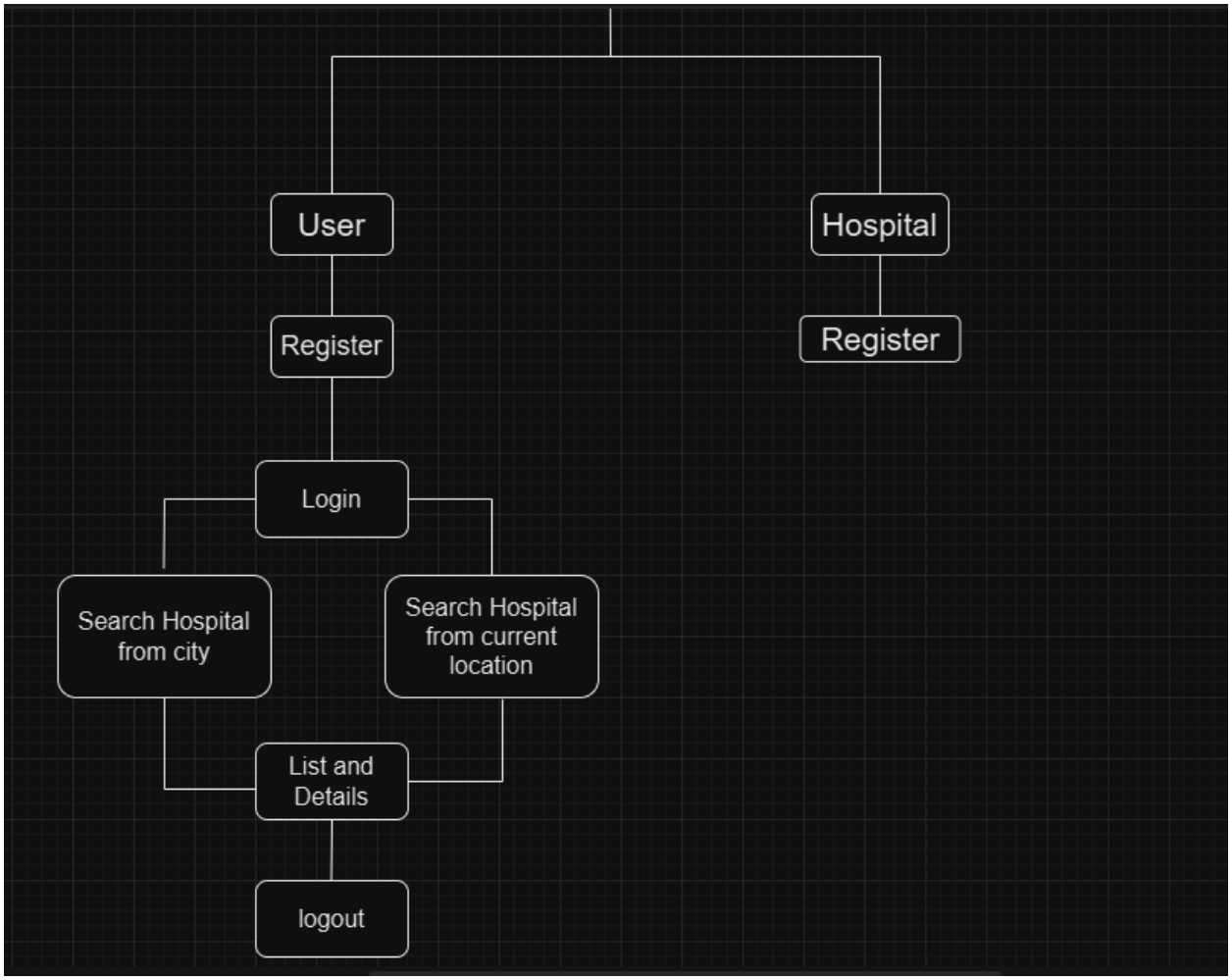


## 2. Flow of the Sytem and components’

UseCase Diagram



Activity Diagram:



# 5. Implementation:

**Technologies:**

Frontend:

**Languages/Frameworks:** JavaScript, React.js

**Libraries:** React Router, React-router-dom(for routing and page protection and to load skeletons), Axios (for api requests)

Backend:

**Languages/Frameworks:** JavaScript (ES6+), Node.js

**Framework:** Express.js (for RESTful API development)

**Database:** Prisma (mongodb model)

**Authentication:** JSON Web Tokens (JWT) for secure authentication

External APIs**:**

Google Places API (for fetching hospital data based on location, getting coordinates and displaying Map and Pins)

**Tools:**

VScode as development environment, Node, Postman(api testing), Cloudify(for storing images in cloud)

**Creating Major Modules:**

Routes:

Routes are implemented using express and http methods (get, push, put etc)

Controllers:

These routes are then bounded to a controller which has the main code.

```
export const register =  async (req, res) =>{

   const {name, username, email, password} = req.body;



   try{

   const hashedPassword = await bcrypt.hash(password, 10);



   const newUser = await prisma.user.create({

      data:{

         name,

         username,

         email,
```

```
            password: hashedPassword

        },

    });

    res.status(201).json({message: "User Created"});

}catch(err){

    console.log(err);

    res.status(500).json({message: "Failed to create user"})

};
```

Controller example, Register controller. As seen Prisma is used for db queries and password are encrypted.

Middleware:

There is a middleware which works to verify tokens and provide user authentication.If there is an error, it stops the request, else it uses next() metho to pass the request forward.

Frontend:

Frontend is divided into two major parts

1) Routes, which uses **BrowserRouter** for routing.
2) Components, which provides desired functionality in the frontend.

Some Important Components are

Search, Filter, Map

# 6. Result and outcome:

## 1. Achievements:

Learned the basics of MERN stack and how to make a fully functional website.

Learned the interaction between each component in a medium scale application.

Learned how to make apis and to use External apis and integrate that with out data.

Created a functioning Search feature for hospitals which supports some filters.

Created and learned successful authentication and authorisation for user.

## 2. Comparison with initial objectives:

The initial idea was to create a website which

1. Allows user to log in and register (done)
2. Allows hospitals to register (done)
3. Allows user to search through hospitals and view detailed information(done)

4. Integrate google api to fetch hospitals from the web with real data (done)
5. Allow user to save hospitals and view them in profile page (UI done, functionality incomplete)
6. Allows user to write reviews on a hospitals and shows other reviews(UI done, functionality incomplete)
7. Allow user to book appointment in desired hospitals(incomplete)

## 3. Challenges faced

Difficulty in integrating and synchronizing data from external APIs like Google Places API.

Learning web developement from HTML/CSS to making full MERN application in a month.

As the components are interdependent changing even a small thing breaks the whole code, which might sometimes require a lot of debugging.

# 7. Future:

There are a lot of bugs and minor adjustment needed on the current project and I plan to implement the incomplete features with time, which increases the scalability of the project.

**Drawbacks**:

1. No authentication for Hospitals
2. After booking appointment, we need a way to confirm it.
3. Can only search by city or latitude/longitude.

Project Developed for : Intel Unnati Training

Developed by : Pratham Trivedi