Navatar – Remote Health Care System

Abstract

In an era where remote work, virtual presence, and telemedicine are transforming healthcare delivery, Navatar offers an innovative solution for remote doctor consultations using a robotic system. Navatar is a telepresence robot that enables doctors to interact with patients and navigate hospital spaces remotely. This system integrates a Django-based backend, a mobile app interface, and a Raspberry Pi-controlled robotic platform equipped with camera, display, and wheels.

Objectives

* To create a secure, real-time platform for doctors to remotely consult patients using a mobile-controlled robot.
* To implement an admin system where hospitals can manage their users and Navatar robots.
* To design and deploy a hardware robot controllable via mobile commands.
* To enable live video streaming for the doctor to see the robot's surroundings.
* To make the doctor’s face visible to patients via a mounted display.

System Architecture

1. Super Admin Interface: Adds hospitals (companies), navatars, and initial hospital admins.

# Super Admin Functionality Specification for Navatar System

## Super Admin Functionality

### Role of Super Admin

* The super admin (e.g., Kavya,Chandrashekar Rao Kuthyar) has the highest level of access.
* Responsibilities include:
  + Adding companies with associated Navatars and admins.
  + Viewing all companies, their Navatars, and admins.
  + Updating company, Navatar, and admin details.
  + Adding more Navatars and admins to existing companies.

### Actions and Workflows

#### Add Companies

* The super admin fills a form to add a new company with the following fields:
  + **Company ID**: Auto-generated (e.g., 2 for Gandhi Hospital).
  + **Company Name**: Compulsory (e.g., "Gandhi Hospital").
  + **Pincode**: Compulsory (e.g., "500003").
  + **Country Code**: Compulsory (e.g., "+91").
  + **Admin Name**: Optional, but if entered, Admin Email is compulsory
  + **Admin Email**: Compulsory if Admin Name is entered (e.g., **Navatar ID**: Optional, auto-generated if entered (e.g., 1 for "Navatar Alpha").
  + **Navatar Name**: Compulsory if Navatar ID is entered (e.g., "Navatar Alpha").
* **Note**: The Navatar location is not provided during creation and is set to NULL. It is updated later in the admin panel.
* **Database Interaction**:
  + Add the company:
  + Assume company\_id: 2 is generated.
  + If a Navatar is provided, add it without a location:
  + INSERT INTO navatar (company\_id, navatar\_name, location)
  + VALUES (2, 'Navatar Alpha', NULL);

Assume navatar\_id: 1 is generated.

* + If an admin is provided, add the admin and link to the Navatar:
  + INSERT INTO navatar\_admin (admin\_name, admin\_email, navatar\_id)

#### View Companies

* The super admin can view all companies, their associated Navatars, and admins.
* **Database Query**:
* SELECT c.company\_id, c.name, c.pincode, c.country,
* n.navatar\_id, n.navatar\_name, n.location,
* a.admin\_email, a.admin\_name
* FROM company c
* LEFT JOIN navatar n ON c.company\_id = n.company\_id
* LEFT JOIN navatar\_admin a ON n.navatar\_id = a.navatar\_id;
* **Example Output**:
  + Company: Gandhi Hospital (company\_id: 2), Pincode: "500003", Country: "+91", Navatar: "Navatar Alpha" (navatar\_id: 1, location: NULL), Admin: "Kavya,Chandrashekar Rao Kuthyar"

#### Update Companies

* The super admin can update the following fields:
  + **Admin Name** (e.g., "Kavya,Chandrashekar Rao Kuthyar" to "Kavya,Chandrashekar Rao Kuthyar ").
  + **Admin Email** (e.g., "[Kavya,Chandrashekar Rao Kuthyar@sangamone.com](mailto:kavya@sangamone.com)" to "[Kavya,Chandrashekar Rao Kuthyar. @sangamone.com](mailto:kavya.sharma@sangamone.com)").
  + **Navatar ID** (e.g., reassign admin to a different Navatar).
  + **Navatar Name** (e.g., "Navatar Alpha" to "Navatar Alpha 2.0").
* **Database Interaction** (example: update admin email and Navatar name):
* UPDATE navatar\_admin
* SET admin\_email = ‘', admin\_name = '
* WHERE admin\_email = '’;
* UPDATE navatar
* SET navatar\_name = 'Navatar Alpha 2.0'
* WHERE navatar\_id = 1;
* **Note**: Updating admin\_email (primary key) requires a delete-and-insert operation:
* DELETE FROM navatar\_admin WHERE admin\_email = 'Kavya,Chandrashekar Rao Kuthyar@sangamone.com';
* INSERT INTO navatar\_admin (admin\_name, admin\_email, navatar\_id)
* VALUES ('Kavya,Chandrashekar Rao Kuthyar ', 'Kavya,Chandrashekar Rao Kuthyar.@sangamone.com', 1);

#### Add More Navatars

* The super admin can add additional Navatars to an existing company.
* **Form Fields**:
  + Company ID: Selected (e.g., 2 for Gandhi Hospital).
  + Navatar Name: Compulsory (e.g., "Navatar Beta").
* **Note**: Location is not provided (set to NULL, updated later in the admin panel).
* **Database Interaction**:
* INSERT INTO navatar (company\_id, navatar\_name, location)
* VALUES (2, 'Navatar Beta', NULL);

Assume navatar\_id: 2 is generated.

#### Add More Admins

* The super admin can add additional admins to a company, with or without Navatar assignment.
* **Form Fields**:
  + Admin Name: Compulsory (e.g., "Manvita").
  + Admin Email: Compulsory (e.g., "[Manvita@gmail.com](mailto:Manvita@gmail.com)").
  + Navatar ID: Optional (e.g., 2 for "Navatar Beta").
* **Database Interaction** (with Navatar assignment):
* INSERT INTO navatar\_admin (admin\_name, admin\_email, navatar\_id)
* VALUES ('Priya', 'priya@sangamone.com', 2);
* **Without Navatar Assignment**:
* INSERT INTO navatar\_admin (admin\_name, admin\_email, navatar\_id)
* VALUES ('Arjun', 'arjun@sangamone.com', NULL);
* **One Admin, Multiple Navatars**:
  + To assign the same admin (e.g., Kavya,Chandrashekar Rao Kuthyar) to multiple Navatars, a new entry with a different email is created:
  + INSERT INTO navatar\_admin (admin\_name, admin\_email, navatar\_id)
  + VALUES ('Kavya,Chandrashekar Rao Kuthyar', 'Kavya,Chandrashekar Rao Kuthyar2@sangamone.com', 2);

#### Admin Panel: Set Navatar Location

* The company admin (e.g., Kavya,Chandrashekar Rao Kuthyar) sets the Navatar location in the admin panel after creation.
* **Example**: Set location for "Navatar Alpha" to "Ward 3".
* **Database Interaction**:
* UPDATE navatar
* SET location = 'Ward 3'
* WHERE navatar\_id = 1;

## 3. Database Schema

### Company Table

* Stores company information.
* **Fields**:
  + company\_id (Primary Key, auto-incremented, e.g., 2 for Gandhi Hospital).
  + name (Compulsory, e.g., "Gandhi Hospital").
  + pincode (Compulsory, e.g., "500003").
  + country (Compulsory, e.g., "+91").
* **Schema**:
* CREATE TABLE company (
* company\_id SERIAL PRIMARY KEY,
* name VARCHAR(100) NOT NULL,
* pincode VARCHAR(10),
* country VARCHAR(50)
* );

### Navatar Table

* Stores Navatar details for each company.
* **Fields**:
  + navatar\_id (Primary Key, auto-incremented, e.g., 1 for "Navatar Alpha").
  + company\_id (Foreign Key, links to company table).
  + navatar\_name (Compulsory, e.g., "Navatar Alpha").
  + location (Nullable, e.g., "Ward 3", set in admin panel).
* **Schema**:
* CREATE TABLE navatar (
* navatar\_id SERIAL PRIMARY KEY,
* company\_id INTEGER NOT NULL,
* navatar\_name VARCHAR(100) NOT NULL,
* location VARCHAR(100),
* FOREIGN KEY (company\_id) REFERENCES company(company\_id) ON DELETE CASCADE
* );

### Navatar Admin Table

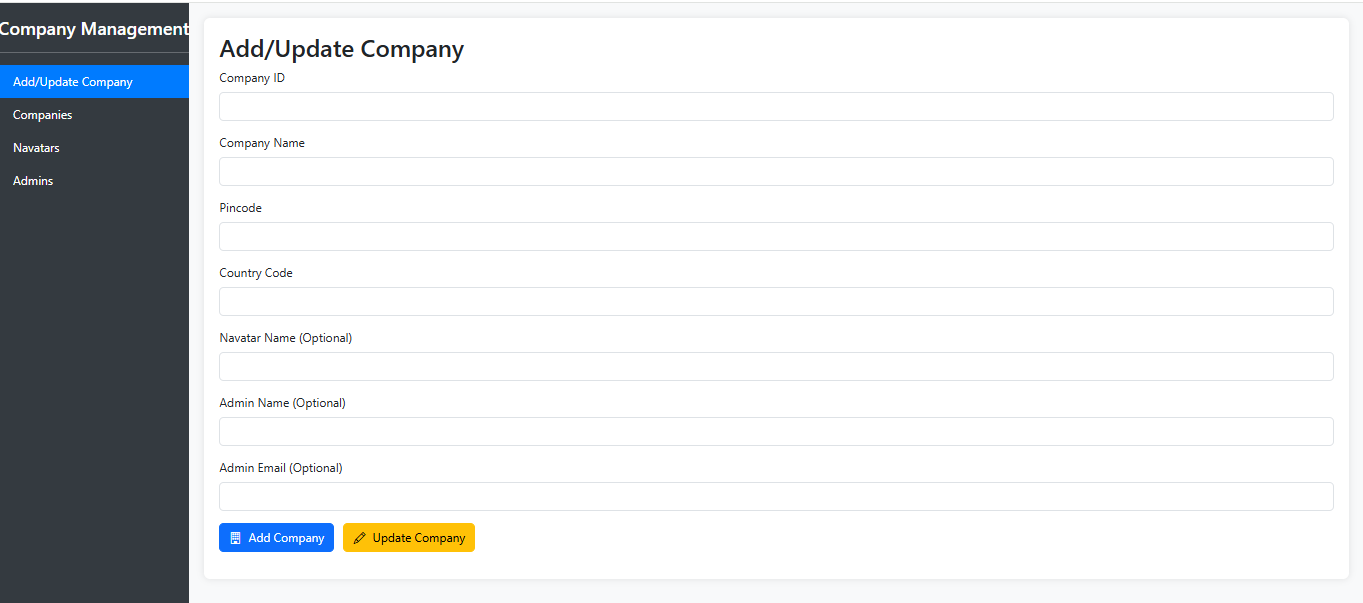
* Stores admin details and their associated Navatar.
* **Fields**:
  + admin\_name (Compulsory, e.g., "Kavya,Chandrashekar Rao Kuthyar").
  + admin\_email (Primary Key, e.g., "[Kavya,Chandrashekar Rao Kuthyar@sangamone.com](mailto:kavya@sangamone.com)").
  + navatar\_id (Nullable, links to navatar table).
* **Schema**:
* CREATE TABLE navatar\_admin (
* admin\_name VARCHAR(100) NOT NULL,
* admin\_email VARCHAR(100) PRIMARY KEY,
* navatar\_id INTEGER,
* FOREIGN KEY (navatar\_id) REFERENCES navatar(navatar\_id) ON DELETE SET NULL

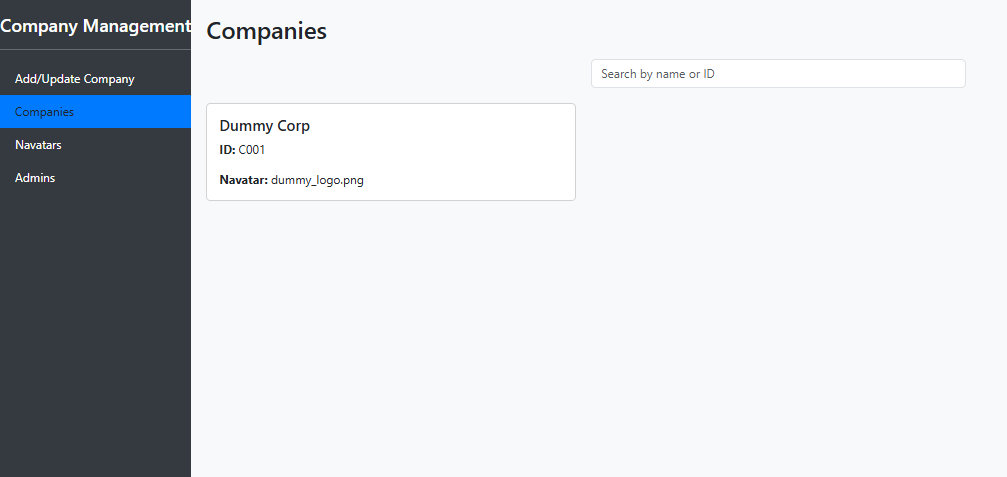
Requirements:

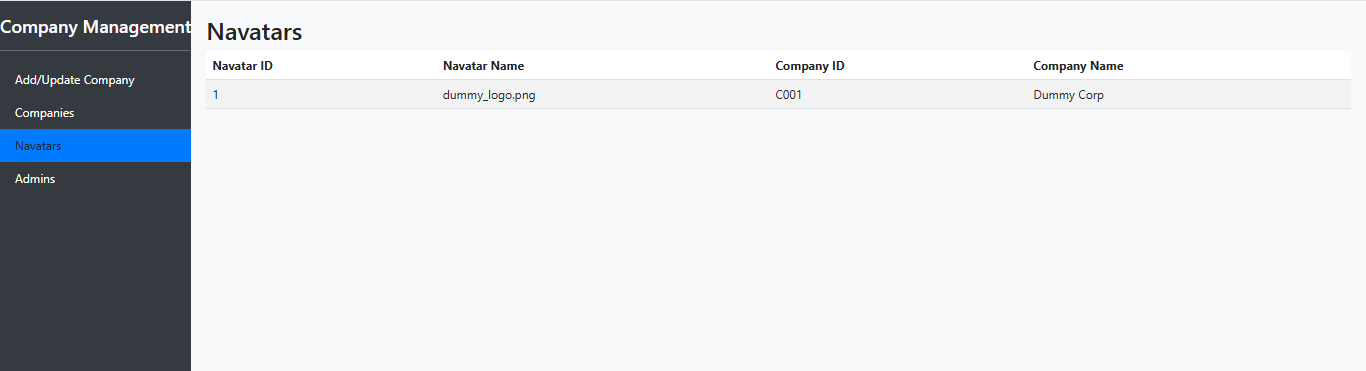
The super admin functionality is a core component of the Navatar system, a web-based telehealth platform. It interacts with the database to manage companies, Navatars, and admins, providing a foundation for remote healthcare consultations. The system integrates with:

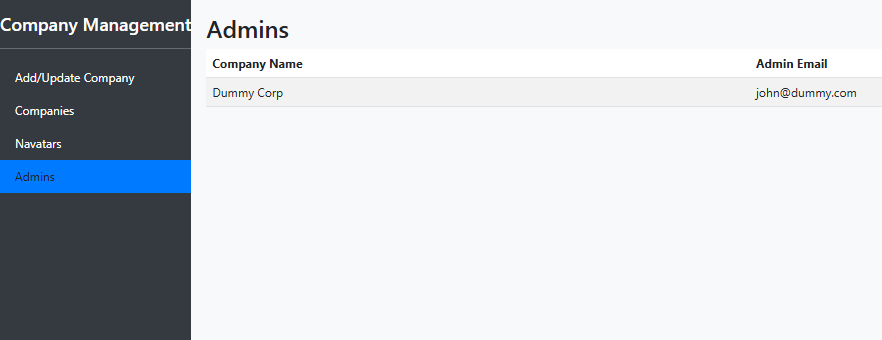
* **Django Backend**: Handles business logic and database operations.
* **MySQL Database**: Stores all data (companies, Navatars, admins).
* **Frontend (CSS/JavaScript)**: Provides a user interface for the super admin.

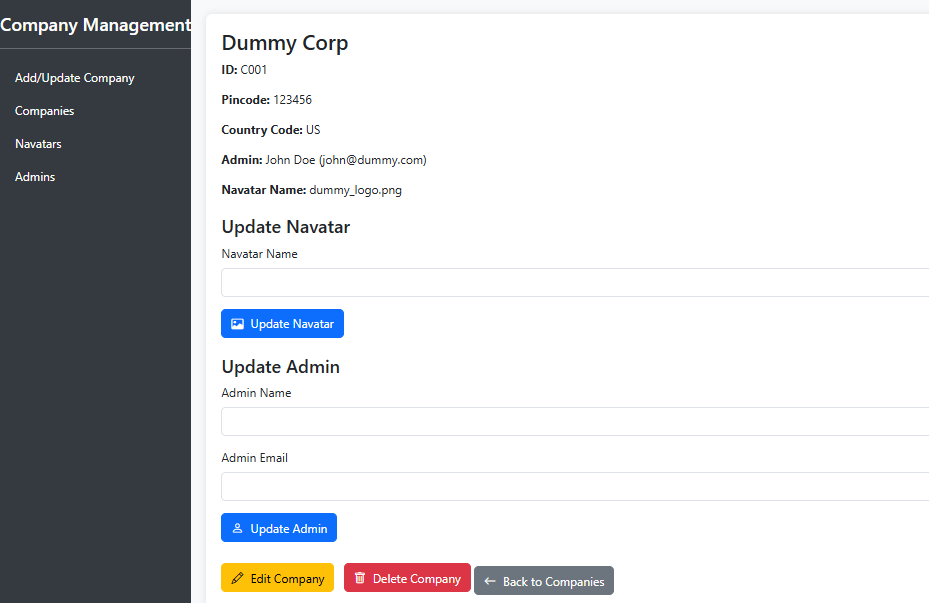
Dummy Frontend Design

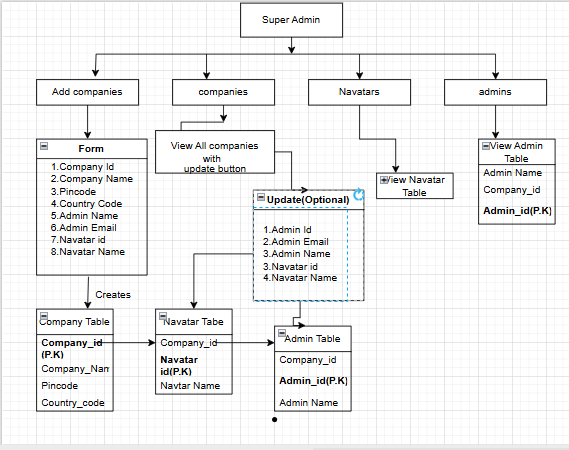












* **Dashboard** → **Add Company** → **Dashboard**
* **Dashboard** → **View Companies** → **Update Company** → **View Companies**
* **Dashboard/View Companies** → **Add More Navatars** → **Dashboard**
* **Dashboard/View Companies** → **Add More Admins** → **Dashboard**
* **Add Navatar** → **Admin Panel (by Company Admin)** → Sets Navatar Location

The Super Admin logs into the system using credentials. This leads to the Super Admin Dashboard. From the dashboard, the Super Admin selects "Add Company." They fill a form with Company Name, Pincode, Country Code, and optional Admin and Navatar details. The system saves the company to the database. If a Navatar is added, its location is set to NULL. If an admin is added, the system assigns an auto-generated admin\_id. This action returns to the dashboard with a success message. From the dashboard, the Super Admin selects "View Companies." The system displays a table of companies, Navatars, and admins. From this table, the Super Admin can click "Edit" on a company. This opens a form to update Admin Name, Admin Email, or Navatar Name. After updating, the system saves changes to the database. This returns to the "View Companies" page. The Super Admin can also select "Add Navatar" from the dashboard. They choose a company and add a Navatar Name, with location as NULL. The system saves the Navatar and returns to the dashboard. Similarly, the Super Admin selects "Add Admin" to add a new admin with a unique email. The system saves the admin and returns to the dashboard. A company admin later sets the Navatar location in the admin panel, updating the database.

Admin Functionality Specification

Role of Admin: Each Admin manages the users, navatars, and bookings for their assigned hospital (company).

Responsibilities Include:

* Adding users (doctor, nurse, personal assistant)
* Viewing and updating user data
* Viewing all bookings made by the users in their hospital
* Viewing and updating Navatar location

Workflow Based on Diagram:

1. Add User:

* Admin fills a form with: user ID, name, email, role, company ID, and mobile number.
* This information is inserted into the User Table.
* The form ensures data is tied to the admin's hospital only.

User Table Schema:

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

user\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

company\_id INTEGER NOT NULL,

role VARCHAR(50),

mobileno VARCHAR(15),

FOREIGN KEY (company\_id) REFERENCES company(company\_id) ON DELETE CASCADE

);

2. View Users:

* Displays all users for the admin’s hospital.
* Admin can edit (update) role or basic info.
* Clicking "Update" opens a prefilled form.
* On submission, updates the User Table entry.

3. View Bookings:

* Admin sees a list of all navatar bookings made by their users.
* Displays: Booking ID, User Name, Booking Date, Start Time, Duration, Navatar ID.

Booking Table Schema:

CREATE TABLE meeting (

booking\_id SERIAL PRIMARY KEY,

user\_id INTEGER NOT NULL,

date DATE NOT NULL,

start\_time TIME NOT NULL,

duration INTEGER NOT NULL,

navatar\_id INTEGER NOT NULL,

FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE,

FOREIGN KEY (navatar\_id) REFERENCES navatar(navatar\_id) ON DELETE CASCADE

);

4. View All Navatars:

* Admin views navatar\_id, navatar\_name, and current location.
* Admin can update the location field for deployment.

Navatar Table Schema:

CREATE TABLE navatar (

navatar\_id SERIAL PRIMARY KEY,

company\_id INTEGER NOT NULL,

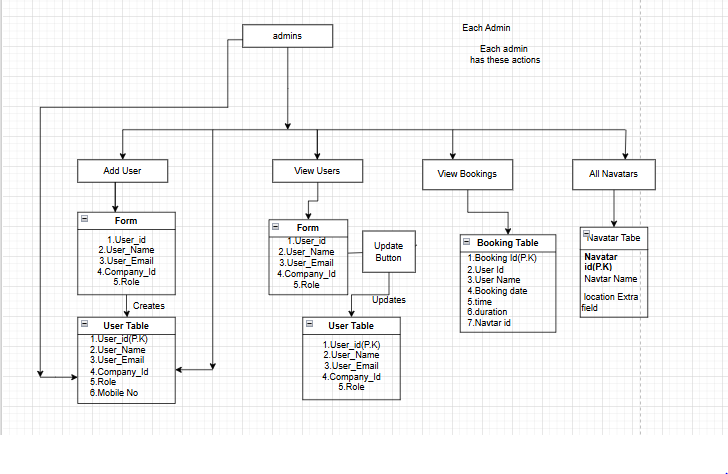
navatar\_name VARCHAR(100) NOT NULL,

location VARCHAR(100),

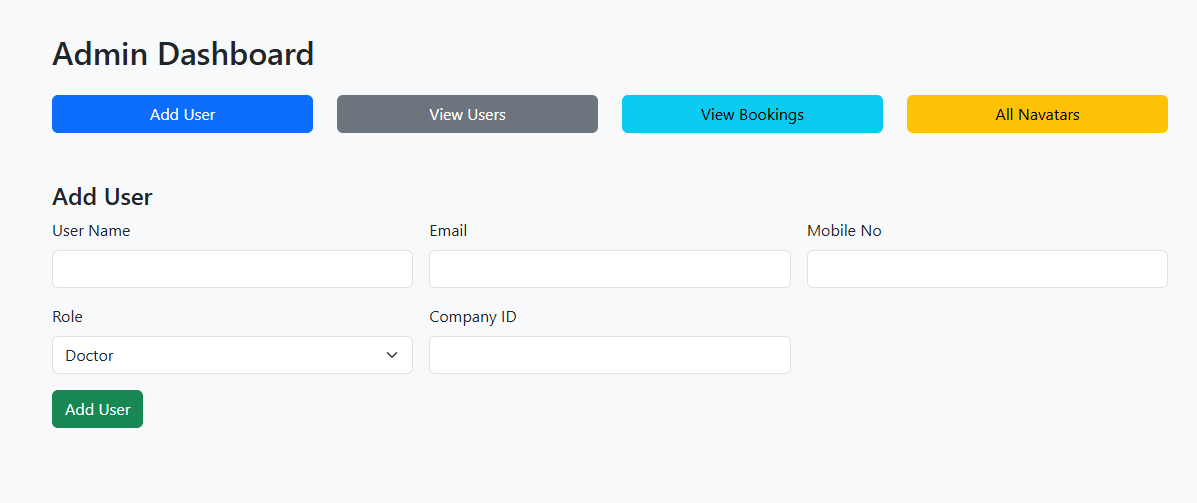
FOREIGN KEY (company\_id) REFERENCES company(company\_id) ON DELETE CASCADE

);

* View users: shows and updates user data.
* View bookings: shows all sessions by doctors and PAs.
* View navatars: shows and updates location.
* Add user: populates user table.



Front End Design:



**User Actions:**

**Booking a Navatar**

* Doctor logs into the mobile app.
* Selects desired date, time, duration, and hospital floor/location.
* Backend checks:
  + Doctor's company\_id
  + Navatars with matching location
  + Status = idle
  + No time conflicts

**🔹 If available:**

* Doctor selects one navatar.
* Booking created in database with status pending.

**Payment**

* Doctor proceeds to payment screen.
* After successful payment:
  + payment\_id, payment\_amount, and payment\_date\_time recorded
  + booking\_status = confirmed
  + meeting\_status = pending

**Session Notification (App Reminder)**

* App regularly checks upcoming sessions.
* If any session is within 30 minutes, a **popup reminder** is shown: "Your session with Navatar starts soon."

**Light Turns ON on Navatar**

* When session starts:
* Raspberry Pi listens to navatar/2/session and turns on GPIO pin for light.

**Doctor Launches Control Panel**

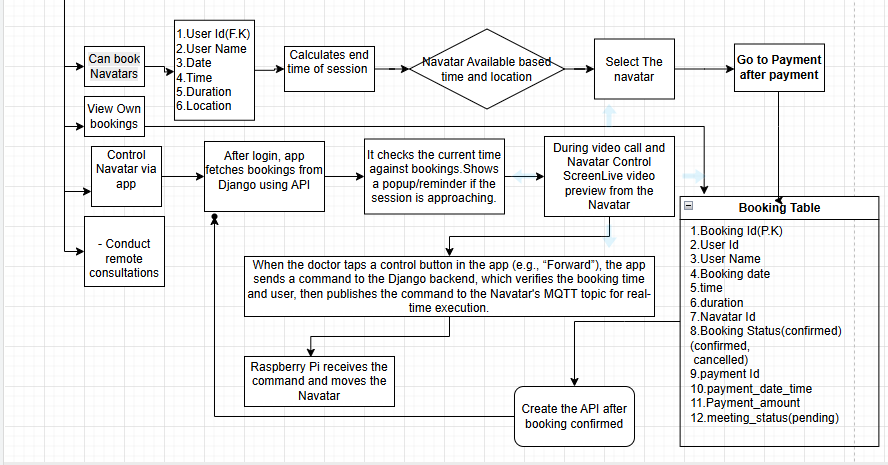
* At session time, doctor clicks "Start Session"
* App shows:
  + **Live video feed** from Navatar camera
  + **Control buttons** (forward, back, left, right)

**Django Verifies Session**

* Django checks:
  + Does booking\_id exist?
  + Is booking confirmed?
  + Is current time between start\_time and end\_time?

**🔹 If valid:**

* Django publishes MQTT message:

****

Complete Flow chart

https://drive.google.com/file/d/1gPBPcXLQYGNyseg36aK9thKHIC1XSfQm/view?usp=drive\_link

This is the complete flow chart link.

Approach 1: Raspberry Pi as Web Server (We Are Going With This)

* The Raspberry Pi itself acts as the web server by running Nginx and a Python web app (like Flask).
* The mobile app (client) sends HTTP requests (forward, stop, left, right) directly to the Pi using its public IP or domain name.
* The Pi receives the command, controls the GPIO pins, and moves the car using the motor driver and wheels.
* This allows the user to control the car from anywhere over the internet, without needing any external cloud database.
* This is the approach we are going with, as you suggested

 Approach 2: Raspberry Pi  and mobile app as Client, Database as Server

* In this method, the mobile app sends commands to a cloud server (like Firebase or a hosted database).
* The Raspberry Pi acts as a client, and it keeps checking the cloud database at regular intervals (e.g., every second).
* If it finds a new command, it executes the appropriate movement using GPIO pins.
* This approach uses the cloud to store and transfer commands but involves more components (cloud + Pi polling logic).

Requiremens:Motor wheels,Battery,Raspberry Pie,nginx server etc

Mobile App (to control the car)

Linode Server (with a public IP)

Raspberry Pi (connected to the car wheels)

Wheels + Motor Driver

Ngrok (to connect your Pi to the internet)

You tap "Forward" in the mobile app from home →

That command goes to the Raspberry Pi in college →

Raspberry Pi makes the car move forward.

Mobile App → Linode Server → Raspberry Pi (via Ngrok) → Motor Driver → Wheels move

🔧 What You Already Have

Raspberry Pi

Linode Server (public IP like 172.232.193.93 )

4 wheels + motor driver

Mobile device

Internet connection

Ngrok (we'll install it)

🟩 Step 1: Set up Raspberry Pi

1. Connect wheels and motor driver to Raspberry Pi using GPIO pins.

2. Install Python and Flask:

sudo apt update

sudo apt install python3 pip

pip3 install flask

3. Create a small Flask server on the Pi (to listen for commands like "forward").

🟨 Step 2: Install Ngrok on Raspberry Pi

1. Go to [https://ngrok.com](https://ngrok.com), create a free account.

2. Download Ngrok on Pi and install:

wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok stable linux arm.zip

unzip ngrok stable linux arm.zip

3. Connect Ngrok to your account:

./ngrok authtoken YOUR\_TOKEN\_HERE

4. Start Ngrok:

./ngrok http 5000

Now Ngrok gives you a public URL (like https://abc123.ngrok.io )

🟧 Step 3: Create API on Linode Server

1. Install Python, Django, and Django REST Framework:

sudo apt update

sudo apt install python3 pip

pip3 install django djangorestframework

2. Create a Django project and add a simple API that:

Receives POST requests from the mobile app

Forwards the command to the Ngrok URL (Raspberry Pi)

🟥 Step 4: Mobile App

1. Build a mobile app using Flutter or Android Studio .

2. The app sends commands (like "forward", "stop", etc.) to your Linode server IP (e.g., http://172.232.193.93/command ).

3. Linode then forwards this command to Raspberry Pi using the Ngrok URL.

1. Mobile app sends:

POST http://172.232.193.93/command

Body: { "command": "forward" }

2. Django on Linode receives it and sends:

POST https://abc123.ngrok.io/move

Body: { "command": "forward" }

3. Raspberry Pi gets it and turns the motors on.