Introduction to Clinic Management System

A **Clinic Management System** is a digital platform designed to streamline clinic operations by managing appointments, patient records, billing, staff schedules, and more. It leverages technology to enhance administrative efficiency and improve patient care by offering real-time data access and seamless workflow integration.

Purpose:

1. **Efficient Operations:** Streamlines daily administrative tasks, reducing manual workload.
2. **Enhanced Patient Care:** Centralized management of patient records and history improves diagnosis accuracy.
3. **Appointment Optimization:** Real-time booking and rescheduling minimize patient wait times.
4. **Data Security:** Ensures the confidentiality of sensitive patient and clinic data.
5. **Resource Management:** Tracks inventory for medical supplies and drugs.
6. **Billing Simplification:** Generates accurate invoices and supports multiple payment methods.

Scope of Clinic Management System Development Project

1. Functional Scope:

- **User Management:**
    - Registration and authentication for patients, doctors, and staff.
    - Role-based access for users (admin, doctors, staff).
- **Appointment Scheduling:**
    - Real-time booking and cancellation of appointments.
    - Notifications for upcoming appointments.
- **Patient Records Management:**
    - Maintain comprehensive patient medical history.
    - Integration with diagnostic reports and prescriptions.
- **Billing and Payments:**
    - Automated billing based on services provided.
    - Integration with online payment gateways.
- **Inventory Management:**
    - Real-time tracking of medical supplies and drugs.

        - Alerts for low stock and expiry.

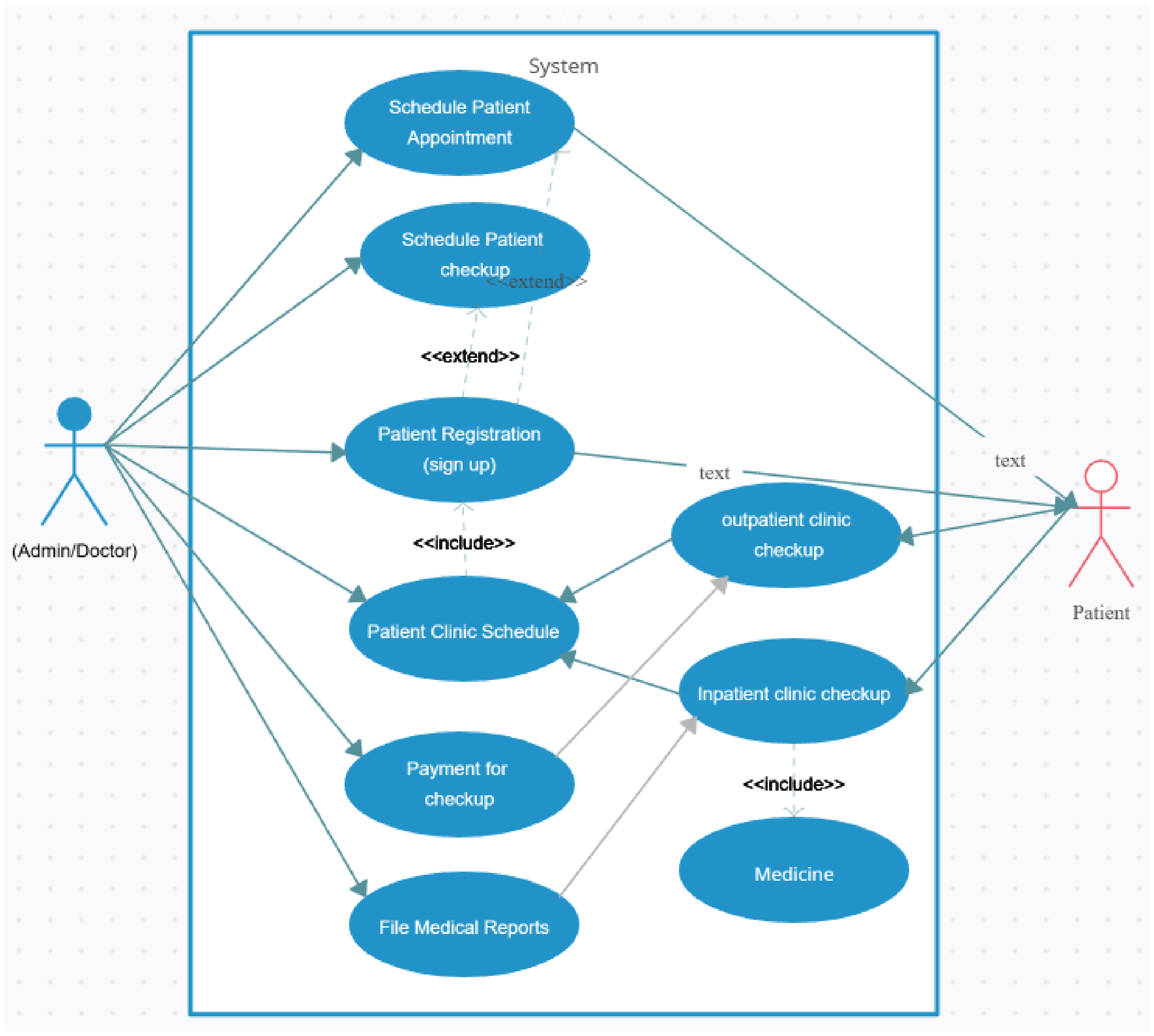
2. Technical Scope:

- **Platform Compatibility:**
    - Mobile apps for iOS and Android.
    - Web-based interface for desktop access.
- **Technology Stack:**
    - Backend: Django REST Framework.
    - Frontend: React.js or Angular for the user interface.
    - Database: PostgreSQL for structured data storage.
- **Development and Deployment Scope:**
    - **Phased Development:**
        - Initial release with core features like appointments and patient management.
        - Gradual rollout of advanced features such as analytics and predictive models.
    - **Testing and Quality Assurance:**
        - Functional and performance testing for all modules.
    - **Post-Launch Support:**
        - Regular updates and maintenance services.

Definitions, Acronyms, and Abbreviations:

Term	Full Form	Description
CMS	Clinic Management System	Platform to streamline clinic operations.
EMR	Electronic Medical Records	Digital repository of patient health data.
API	Application Programming Interface	Enables third-party integration.
RBAC	Role-Based Access Control	User access control based on roles.

User case diagram



Product Function

The Clinic Management System is designed to support various operational, administrative, and clinical workflows within a clinic. The key functions of the system are organized into the following categories:

---

1. Patient Management

- **Registration:**
    - Enables new patients to register with their basic details, medical history, and contact information.

- **Medical Records:**
    - Stores and retrieves patient health information, prescriptions, and diagnostic reports.
- **Patient Portal:**
    - Provides patients access to their medical history, invoices, and upcoming appointments.

---

2. Appointment Management

- **Scheduling:**
    - Allows patients to book, reschedule, or cancel appointments with doctors.
- **Real-Time Availability:**
    - Displays available time slots for doctors to optimize scheduling.
- **Reminders:**
    - Sends automated notifications and reminders for upcoming appointments.

---

3. Doctor and Staff Management

- **Doctor Profiles:**
    - Stores details like qualifications, specialties, and schedules.
- **Staff Tasks:**
    - Assigns and tracks tasks for administrative and medical staff.
- **Shift Management:**
    - Maintains a roster for doctors and staff to manage clinic operations efficiently.

---

4. Billing and Payments

- **Invoice Generation:**
    - Automatically generates bills based on the services rendered during appointments.
- **Payment Processing:**
    - Integrates with multiple payment gateways to support online and on-site transactions.
- **Billing History:**
    - Maintains a record of all past invoices for reference and compliance.

5. Notifications and Alerts

- **Patient Alerts:**
  - Sends reminders for appointments, medications, and follow-ups.
- **Staff Alerts:**
  - Notifies staff of task assignments, critical inventory levels, or system updates.

---

6. Security and Compliance

- **Data Protection:**
  - Ensures sensitive patient data is encrypted and secure.
- **Regulatory Compliance:**
  - Adheres to standards like HIPAA and GDPR to maintain legal compliance.

---

7. User Management

- **Role-Based Access Control (RBAC):**
  - Provides access to system features based on user roles (e.g., admin, doctor, patient).
- **Profile Management:**
  - Allows users to update personal details and preferences.

User Requirements

User Requirements for Clinic Management System

The system will cater to multiple user types, each with distinct needs and functionalities. These user requirements are categorized based on roles to ensure comprehensive coverage.

---

1. Patient Requirements:

- **Registration and Login:**
  - Ability to create an account with basic details (name, contact, address, medical history).
  - Login using email/phone number and password.

- **Appointment Management:**
    - View available time slots for doctors and book appointments.
    - Modify or cancel appointments.
- **Access to Medical Records:**
    - View prescriptions, diagnostic reports, and medical history.
    - Download and share medical documents.
- **Notifications:**
    - Receive reminders for upcoming appointments, medication schedules, and follow-ups.
- **Billing and Payments:**
    - View invoices and payment history.
    - Pay bills online using multiple payment methods.
- **Feedback and Support:**
    - Provide feedback on services.
    - Access customer support for assistance.

---

2. Doctor Requirements:

- **Account Management:**
    - Register and maintain a profile with qualifications and specialties.
    - Update availability schedules.
- **Patient Interaction:**
    - Access patient details, medical history, and diagnostic reports.
    - Add notes, prescriptions, and recommendations.
- **Appointment Management:**
    - View daily schedules and manage appointment queues.
- **Analytics and Reports:**
    - Access patient visit summaries and treatment outcomes for reference.
- **Notifications:**
    - Receive alerts for new appointments or emergencies.

---

3. Admin Requirements:

- **User Management:**
  - Register and manage patients, doctors, and staff accounts.
  - Assign roles and access permissions.
- **Appointment Oversight:**
  - Monitor overall appointment scheduling and availability of doctors.
- **Billing and Finance:**
  - Track payments, generate invoices, and handle refunds.
  - Manage clinic revenue and financial reports.
- **Inventory Management:**
  - Monitor and update stock of medical supplies and drugs.
  - Generate alerts for low inventory levels.
- **System Maintenance:**
  - Perform backups and monitor system health.
  - Resolve system errors and technical issues.

---

4. Staff Requirements:

- **Patient Management:**
  - Assist in patient registration and record updates.
  - Schedule and manage appointments for walk-in patients.
- **Operational Support:**
  - Coordinate with doctors for patient flow and emergencies.
  - Manage administrative tasks like report printing and inventory updates.
- **Notifications and Alerts:**
  - Receive updates on tasks assigned by the admin or doctors.

---

5. Additional User Requirements:

- **Scalability:**
  - Support for adding new users without affecting system performance.

- **Accessibility:**
    - Ensure user-friendly interfaces for all user types, including support for older patients or individuals with disabilities.
- **Data Security:**
    - Protect sensitive information with encryption and secure access protocols.

Clinic Management System (CMS) A digital platform designed to streamline clinic operations such as appointments, billing, and patient record management.

Electronic Medical Records (EMR)A digital repository of patient health data, including history, diagnoses, and treatments.

Role-Based Access Control (RBAC) A security mechanism that restricts access to system features based on the user's role.

Appointment Queue A sequential list of patients scheduled for consultations or treatments.

Invoice A document summarizing the services provided and their associated costs.

Payment Gateway A service that facilitates secure online payments between patients and the clinic.

Patient Portal An online interface where patients can access their medical records, appointments, and bills.

Doctor Dashboard A dedicated interface for doctors to view schedules, patient details, and medical records.

Admin Panel The central interface for managing users, appointments, billing, and system configurations.

Inventory Management The process of tracking and managing medical supplies and drug stocks in the clinic.

Encryption A method of protecting sensitive data by converting it into an unreadable format without a key.

Two-Factor Authentication (2FA) An additional layer of security requiring two forms of identification for access.

Data Backup A copy of data stored securely to prevent loss in case of system failure.

User Interface (UI) The visual and interactive components through which users interact with the system.

Database A structured collection of data stored electronically, such as PostgreSQL or MySQL.

Cloud Hosting A type of hosting where data and applications are stored on remote servers accessible via the internet.

IoT Devices Internet-connected devices such as biometric scanners or smart printers used in clinic operations.

API (Application Programming Interface) A set of rules that allow different software applications to communicate.

Notifications Alerts or reminders sent to users about appointments, payments, or system updates.

Operating Environment for a Clinic Management System

The operating environment outlines the hardware, software, and network requirements to ensure the smooth functioning of the clinic Management System App. It also specifies compatibility with various devices and platforms to enhance user experience.

1. Hardware Requirements:

Client-Side:

- **User Devices:**
  - Desktops, laptops, smartphones, or tablets.
  - Minimum configuration:
    - Processor: Intel Core i3 or equivalent.
    - RAM: 4 GB or higher.
    - Storage: 50 MB free disk space for the application.
  - Recommended configuration:
    - Processor: Intel Core i5 or equivalent.
    - RAM: 8 GB or higher.
    -

Server-Side:

- **Servers:**
  - Processor: Intel Xeon or AMD EPYC series.
  - RAM: Minimum 16 GB (expandable based on scale).

    - Storage: SSD with at least 500 GB of space.
    - Network Interface: Gigabit Ethernet or faster.

Peripherals:

- Printers for invoice and prescription generation.
- Barcode scanners (for pharmacy inventory).
- Biometric devices for secure staff/patient authentication (optional).

2. Software Requirements:

Client-Side:

- Operating Systems:
    - Windows 10 or later, macOS 10.13 (High Sierra) or later, Android 8.0 or later, iOS 13 or later.
- Browsers (for web access):
    - Google Chrome (latest version), Mozilla Firefox, Microsoft Edge, Safari.

Server-Side:

- Operating System:
    - Linux (Ubuntu 22.04 LTS or equivalent preferred).
    - Windows Server 2019 or later (if using Windows).
- Database:
    - PostgreSQL 13+ (preferred) or MySQL 8.0+.

Third-Party Integrations:

- APIs for online payment gateways (e.g., Stripe, Razorpay).
- SMS and email notification services (e.g., Twilio, SendGrid).

Network Requirements:

- **Connectivity:**
    - Reliable internet connection (minimum 5 Mbps for clients, 100 Mbps for the server).

- **Bandwidth:**
  - Adequate bandwidth to handle concurrent users, estimated at 10 Mbps per 50 active users.
- **Protocols:**
  - HTTPS for secure communication.

Security Configurations:

- Data encryption (SSL/TLS).
- Role-based access control (RBAC).
- Authentication protocols: OAuth 2.0, JWT for session management.

3.User Interface (UI) Requirements for Clinic Management System

The user interface is a crucial aspect of the system, ensuring seamless interaction for all user roles: patients, doctors, administrators, and staff. The design must be intuitive, accessible, and responsive.

---

1. General Design Principles

- **Consistency:**
  - Use a uniform color scheme, typography, and layout across all screens.
- **Responsiveness:**
  - Ensure the UI adapts to different screen sizes (desktop, tablet, mobile).
- **Accessibility:**
  - Adhere to Web Content Accessibility Guidelines (WCAG) for users with disabilities.
  - Provide features such as screen readers, large fonts, and contrast adjustment.
- **Minimalism:**
  - Focus on essential elements, avoiding unnecessary complexity or clutter.

---

2. Key Components of the Interface

2.1 Patient Portal:

- **Dashboard:**
    - Display upcoming appointments, billing summaries, and recent medical records.
- **Navigation Menu:**
    - Simple menu with options: *Appointments*, *Medical Records*, *Billing*, and *Support*.
- **Appointment Booking Page:**
    - Calendar view to select dates and available time slots.
- **Notifications:**
    - Alerts for upcoming appointments, bills due, and health tips.
- **Forms:**
    - User-friendly forms for updating personal details and registering complaints.

2.2 Doctor Dashboard:

- **Patient Overview:**
    - List of scheduled appointments with patient details and medical history.
- **Medical Notes Section:**
    - Area for writing prescriptions and adding diagnostic notes.
- **Calendar:**
    - View and update personal availability.
- **Search Functionality:**
    - Quickly find patient details by name or ID.

2.3 Admin Panel:

- **Overview Screen:**
    - Display clinic statistics: total patients, revenue, inventory status, and active appointments.
- **User Management:**
    - Interface for registering, updating, and managing users (patients, doctors, staff).
- **Billing and Finance Section:**
    - Generate reports, view transaction history, and monitor outstanding payments.
- **Inventory Management:**
    - Table view with stock levels, expiry dates, and reorder alerts.

2.4 Staff Interface:

- **Task Management Page:**
    - List of tasks assigned for the day (e.g., patient registration, billing).
- **Appointment Scheduler:**
    - Simplified interface for booking, rescheduling, or canceling appointments on behalf of patients.
- **Inventory Section:**
    - Basic access to check and update inventory stock levels.

---

3. Usability Features

- **Navigation:**
    - Breadcrumbs and clear labels for easy navigation.
- **Feedback Mechanism:**
    - Allow users to report UI issues or suggest improvements.
- **Language Support:**
    - Multi-language support for clinics in diverse regions.
- **Error Handling:**
    - Display clear error messages with guidance for resolution.

---

4. Visual Elements

- **Icons:**
    - Use recognizable icons for common actions (e.g., edit, delete, add).
- **Typography:**
    - Fonts: Sans-serif fonts for readability.
    - Size: 14-16px for body text, larger for headings.
- **Colors:**
    - Use calming colors like blues and greens, with accent colors for alerts.
- **Graphs and Charts:**

    - Represent reports visually (e.g., bar charts for revenue trends, pie charts for appointment distribution).

4.Security Requirements for Clinic Management System

The security requirements ensure the confidentiality, integrity, and availability of sensitive data in the Clinic Management System. These measures are crucial to protect patient information, clinic operations, and compliance with regulatory standards like HIPAA and GDPR.

---

1. Authentication and Access Control

- **User Authentication:**
    - Implement multi-factor authentication (MFA) for all user roles.
    - Use secure login mechanisms (OAuth 2.0, JWT).
    - Enforce strong password policies (minimum length, special characters).
- **Role-Based Access Control (RBAC):**
    - Grant users access to features based on their roles (e.g., patient, doctor, admin).
    - Restrict admin-level features from non-administrative users.
- **Session Management:**
    - Automatically log out inactive users after a configurable timeout.
    - Use secure session tokens with encryption.

---

2. Data Security

- **Encryption:**
    - Encrypt all data in transit using SSL/TLS protocols.
    - Use AES-256 encryption for data stored in the database.
- **Data Masking:**
    - Mask sensitive data like patient IDs and billing details during display.
- **Database Security:**

    - Secure the database with firewalls and restricted IP access.
    - Implement regular backups with encryption for disaster recovery.

---

3. Network Security

- **Secure Communication:**
    - Enforce HTTPS for all communications between clients and the server.
    - Use VPNs for secure remote access to the system.
- **Firewall Protection:**
    - Deploy firewalls to monitor and control incoming and outgoing network traffic.
- **Intrusion Detection Systems (IDS):**
    - Use IDS to identify and mitigate unauthorized access attempts.

---

4. IoT Device Security (if applicable)

- **Device Authentication:**
    - Use secure pairing protocols for IoT devices like biometric scanners and smart meters.
    - Update device firmware regularly to patch vulnerabilities.
- **Secure Communication:**
    - Use MQTT or CoAP over TLS for IoT data transmission.

---

5. Application Security

- **Input Validation:**
    - Validate all user inputs to prevent SQL injection, XSS, and CSRF attacks.
- **Error Handling:**
    - Display generic error messages to users while logging detailed errors for debugging.
- **API Security:**
    - Secure all APIs with authentication tokens and rate-limiting.
    - Use CORS policies to restrict unauthorized API access.

---

6. Compliance Requirements

- **Regulatory Compliance:**
  - Adhere to HIPAA guidelines for healthcare data in the US.
  - Follow GDPR for data protection if operating in the EU.
- **Audit Trails:**
  - Maintain logs of all user activities, including logins, data access, and updates.

---

7. Backup and Recovery

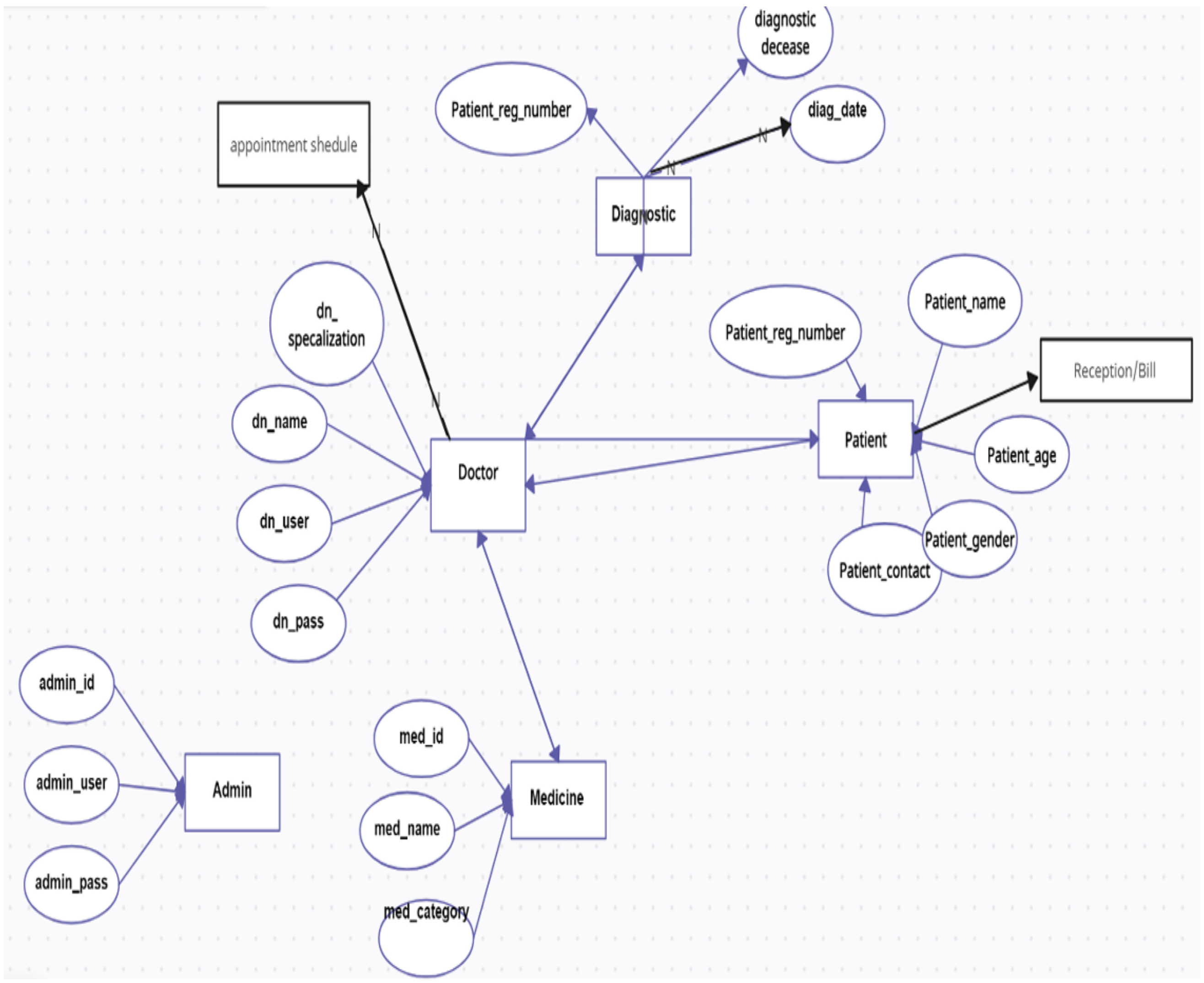
- **Regular Backups:**
  - Schedule daily backups of critical data.
  - Store backups securely, both on-site and off-site.
- **Disaster Recovery Plan:**
  - Define procedures for system recovery in case of data loss or cyberattacks.

---

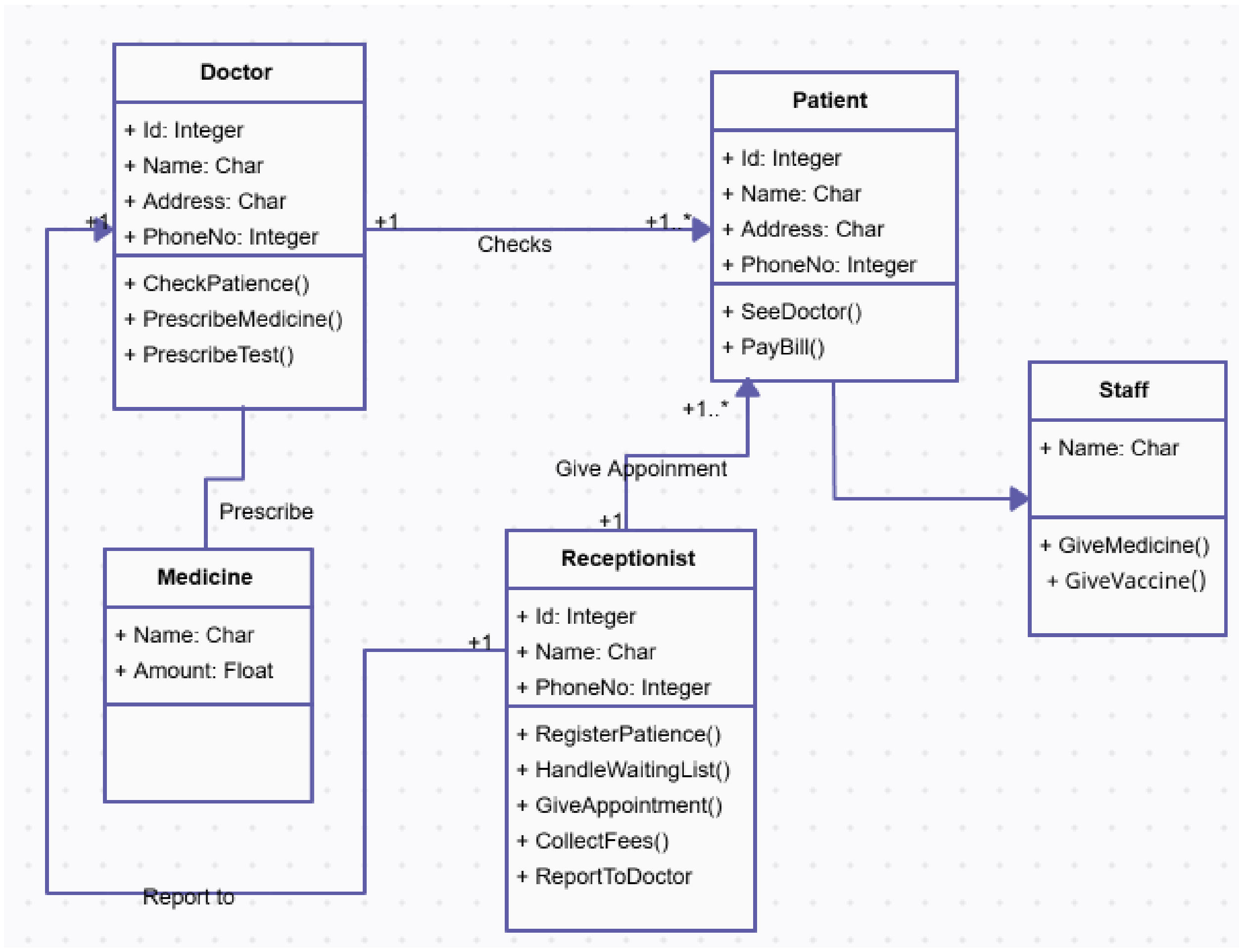
8. Monitoring and Alerts

- **Real-Time Monitoring:**
  - Monitor system activities for suspicious behavior or policy violations.
- **Alerts:**
  - Configure alerts for unusual login attempts, failed transactions, or unauthorized data access.

Er diagram


diagnostic decease
Patient_reg_number
diag_date
appointment shedule
N
N
N
Diagnostic
N
dn_ specalization
Patient_reg_number
Patient_name
Reception/Bill
dn_name
Doctor
Patient
Patient_age
dn_user
Patient_gender
Patient_contact
dn_pass
admin_id
med_id
admin_user
Admin
Medicine
med_name
admin_pass
med_category

CLASS DIAGRAM



Here is the class diagram for the Clinic Management System App, illustrating the classes, their attributes, methods, and relationships

Code :

Frontend:

Appointment.html

```
{% extends 'base.html' %}

{% load static %}
{% block title %} appointment {% endblock %}

{% block content %}


<main>

  <hr>

  <div class="d-flex justify-content-between">

    <h4>Appointment Details</h4>


    <div>

      <button type="button" class="btn text-center  btn-sm custoemBtn" data-bs-toggle="modal" data-
bs-target="#addAppoinment"><i class="bi bi-plus"></i> Add Appoinment</button>

      <a href="{% url 'doctor_wise' %}" class="btn text-center  btn-sm custoemBtn ms-2"><i class="bi
bi-file-earmark"></i> Doctor Wise</a>

      {% include 'modals/add_appoinment.html' %}

      {% include 'modals/add_patient.html' %}

    </div>

  </div>

  <hr>

  <table id="myTable" class="shadow-lg table table-stripedy mt-2">

    <thead>

    <tr>

      <th>No.</th>

      <th>patient</th>

      <th>doctor</th>

      <th>appointed date</th>

      <th>added date</th>

      <th>doctor shift</th>

      <th>priority</th>

      <th>payment</th>
```

```
        <th>amount</th>

        <th>status</th>

        <th>token</th>

        <th>Action</th>

      </tr>

      </thead>


      <tbody>

      {% for a in appoinments %}

        <tr>

          <td>{{forloop.counter}}</td>

          <td>{{a.patient}} </td>

          <td>{{a.doctor}}</td>

          <td>{{a.appointed_date}}</td>

          <td>{{a.added_date}}</td>

          <td>{{a.shift}}</td>

          <td>{{a.get_priority_display}}</td>

          <td>{{a.get_payment_display}}</td>

          <td class="text-center">{{a.amount}}</td>

          <td>{{a.get_status_display}}</td>

          <td>{{a.token_number}}</td>

          <td class="d-flex ">


            <button type="button" class="btn btn-primary text-center m-1 fs-9 btn-sm" data-bs-
toggle="modal" data-bs-target="#editAppoinment{{a.id}}" title="Edit">

              <i class="bi bi-pencil-square"></i>

            </button>
            <button type="button" class="btn btn-danger text-center m-1 fs-9 btn-sm" data-bs-
  toggle="modal" data-bs-target="#deleteAppoinment{{a.id}}" title="View & Edit">
              <i class="bi bi-file-earmark-fill"></i>
            </button>

            {% if a.status == "P" %}
              <form method="post" action="{% url 'approve_appoinment' a.id %}" class="m-1 fs-9">
                {% csrf_token %}
                <button type="submit" class="btn btn-info text-center btn-sm" title="Approve">
                  <i class="bi bi-check-lg"></i>
                </button>
              </form>
            {% endif %}

            {% include 'print/appoinment.html' %}
            <button onclick="printAppoinment('printPage{{a.id}}')" type="button" class="btn btn-success
  m-1 fs-9 btn-sm" title="Print Appoinment">
              <i class="bi bi-printer-fill"></i>
            </button>

          </td>
          {% include 'modals/edit_appoinment.html' %}
          {% include 'modals/delete_appoinment.html' %}
        </tr>
      {% endfor %}
      </tbody>
    </table>
  </main>
```

```
<script type="text/javascript">
   function printAppoinment(page) {
      let printContents = document.getElementById(page).innerHTML;
      let originalContents = document.body.innerHTML;
      document.body.innerHTML = printContents;
      window.print();
      document.body.innerHTML = originalContents;
      location.reload();
   }
</script>

{% endblock %}
```

Base.html

```
{% load static %}

<html lang="en">
<head class="d-print-none">
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <link href="https://fonts.googleapis.com/css2?family=Jost:wght@500&display=swap"
rel="stylesheet">
   <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/css/selectize.bootstrap3.min.css">
   <link href="https://fonts.googleapis.com/css2?family=Jost:wght@500&display=swap"
rel="stylesheet">
```

```
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/css/selectize.bootstrap3.min.css">


    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-
icons.min.css">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-
icons.min.css">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/css/selectize.bootstrap3.min.css"
integrity="sha256-ze/OEYGcFbPRmvCnrSeKbRTtjG4vGLHXgOqsyLFTRjg=" crossorigin="anonymous" />

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/boxicons@latest/css/boxicons.min.css">


    <link rel="stylesheet" href="https://code.jquery.com/jquery-3.7.1.js">

    <link rel="stylesheet" href="https://cdn.datatables.net/2.0.5/css/dataTables.css">

    <link rel="stylesheet" href="https://cdn.datatables.net/buttons/3.0.2/css/buttons.dataTables.css">

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/css/selectize.bootstrap3.min.css">


    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/css/bootstrap.min.css">


    <link rel="icon" href="../static/assets/images/favicon.svg" type="image/x-icon" />

    <!-- [Google Font : Public Sans] icon -->

    <link
href="https://fonts.googleapis.com/css2?family=Public+Sans:wght@400;500;600;700&display=swap"
rel="stylesheet">
```

```html
   <!-- [Tabler Icons] https://tablericons.com -->
   <link rel="stylesheet" href="../static/assets/fonts/tabler-icons.min.css" >
   <!-- [Feather Icons] https://feathericons.com -->
   <link rel="stylesheet" href="../static/assets/fonts/feather.css" >
   <!-- [Font Awesome Icons] https://fontawesome.com/icons -->
   <link rel="stylesheet" href="../static/assets/fonts/fontawesome/" >
   <!-- [Material Icons] https://fonts.google.com/icons -->
   <link rel="stylesheet" href="../static/assets/fonts/material.css" >
   <!-- [Template CSS Files] -->
   <link rel="stylesheet" href="../static/assets/css/style.css" id="main-style-link" >
   <link rel="stylesheet" href="../static/assets/css/style-preset.css" >


   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/js/bootstrap.bundle.min.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
   <script
src="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/js/standalone/selectize.min.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.7/pdfmake.min.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.7/vfs_fonts.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js"></script>
   <script src="https://cdn.datatables.net/2.0.5/js/dataTables.js"></script>
   <script src="https://cdn.datatables.net/buttons/3.0.2/js/dataTables.buttons.js"></script>
   <script src="https://cdn.datatables.net/buttons/3.0.2/js/buttons.dataTables.js"></script>
   <script src="https://cdn.datatables.net/buttons/3.0.2/js/buttons.html5.min.js"></script>
   <script src="https://cdn.datatables.net/buttons/3.0.2/js/buttons.print.min.js"></script>
   <link rel="stylesheet" href="{% static 'style.css' %}">
```

```
  <title class="d-print-none">CMS {% block title %} {% endblock %}</title>
</head>


<body>
{% include 'navbar.html' %}


<div class="alert alert-dismissible topaleart-inside fade show rounded-0" role="alert" style="text-align:
end; background-color: #5b5960; color: #fff; display: none;">
  Alert! You are in a Kgec IT project. this is not for commercial purpus.
  <a class="close" style="cursor: pointer;"><span aria-hidden="true">x</span></a>
</div>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
<script>
  $(document).ready(function() {
    // Slide down the alert on page load
    $(".alert").slideDown(1000);


    // Add click event to close the alert
    $(".alert .close").click(function() {
      $(".alert").slideUp(500);
    });
  });
</script>


{% block content %}
{% endblock %}


{% include 'footer.html' %}
```

```
</body>


<link rel="stylesheet" href="https://code.jquery.com/jquery-3.7.1.js">
<link rel="stylesheet" href="https://cdn.datatables.net/2.0.5/js/dataTables.js">
<link rel="stylesheet" href="https://cdn.datatables.net/buttons/3.0.2/js/dataTables.buttons.js">
<link rel="stylesheet" href="https://cdn.datatables.net/buttons/3.0.2/js/buttons.dataTables.js">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.7/pdfmake.min.js">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.7/vfs_fonts.js">
<link rel="stylesheet" href="https://cdn.datatables.net/buttons/3.0.2/js/buttons.html5.min.js">
<link rel="stylesheet" href="https://cdn.datatables.net/buttons/3.0.2/js/buttons.print.min.js">


<link rel="stylesheet" href="{% static 'style.css' %}">


<script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>


<script>
  new DataTable('#myTable',
    {layout: {topStart:
          {buttons:
              [
                'copy',
                'csv',
                'excel',
                'pdf',
                'print'
              ]
          }
      }
```

```
        });


    $(document).ready(function () {
        $('#selectPatient').selectize({
            sortField: 'text'
        });
    });


    $(document).ready(function () {
        $('#selectDoctor').selectize({
            sortField: 'text'
        });
    });
</script>



<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/js/standalone/selectize.min.js"
integrity="sha256-+C0A5Ilqmu4QcSPxrlGpaZxJ04VjsRjKu+G82kl5UJk="
crossorigin="anonymous"></script>

<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.12.6/css/selectize.bootstrap3.min.css"
integrity="sha256-ze/OEYGcFbPRmvCnrSeKbRTtjG4vGLHXgOqsyLFTRjg=" crossorigin="anonymous" />

<script src="../static/assets/js/component.js"></script>

<script src="../static/assets/js/layout-compact.js"></script>

<script src="../static/assets/js/layout-horizontal.js"></script>

<script src="../static/assets/js/layout-tab.js"></script>

<script src="../static/assets/js/pcoded.js"></script>
```

```
</html>
```

Login.html

```
{% extends 'base.html' %}

{% load static %}

{% block title %} login {% endblock %}
{% block content %}

<div class="loat-lg-end container border border-primary" style="margin-top: 5%; margin-bottom: 5%;
border-radius: 12px; max-width: 300px">
   <h1 style="text-align: center; font-variant: small-caps">login</h1>

   {% for msg in messages %}
   <div class="alert alert-warning alert-dismissible fade show" role="alert">
       {{msg}}
       <a href="{% url 'login' %}"><button type="button" class="close" data-dismiss="alert" aria-
label="Close" style="background: none; border: none">
              <span aria-hidden="true">x</span>
          </button></a>
   </div>
   {% endfor %}

   <form class="form-floatin row g-3 p-2" style="text-align: left" method="POST" action="{% url 'login'
%}">
       {% csrf_token %}
```

```
        <div class="mb-1">
          <label for="" class="form-label" style="font-variant: all-small-caps; font-weight: 500;">Email
ID:</label>
          <input class="form-control" style="border-color: black" id="email" type="text" name="email">
        </div>

        <div class="mb-1">
          <label for="password" class="form-label" style="font-variant: all-small-caps; font-weight:
500;">Password: </label>
          <input class="form-control" style="border-color: black" id="password" type="password"
name="password">
        </div>

        <div class="col-12" style="display: flex; justify-content: space-between">
          <button style="color: black; font-variant: all-small-caps;" class="btn btn-primary"
type="submit">LOGIN
          </button>
        </div>
    </form>
</div>

{% endblock %}
```

Staff .html

```
{% extends 'base.html' %}
{% load static %}
{% block title %} staff {% endblock %}
{% block content %}
{% for msg in messages %}
  <div class="alert fade show rounded-0" role="alert" style="text-align: end; background-color: #ffcf4a;
color: #000000;">
    {{msg}}
    <a style="color:#000000;" class="close"><i class="bi bi-x"></i></a>
  </div>
  <script>$(document).ready(function() {$(".alert .close").click(function()
{$(".alert").slideUp(500)})});</script>
{% endfor %}


<main>


  <hr>
  <div class="d-flex justify-content-between align-items-center">
    <h4>Staff Details</h4>
    <button type="button" class="btn text-center fs-14 btn-sm custoemBtn" data-bs-toggle="modal"
data-bs-target="#addStaffModal"><i class="bi bi-plus"></i> Add Staff</button>
  </div>
  <hr>
  <form method="post" action="{% url 'staff' %}">
    {% csrf_token %}
      <label for="selectrole">FILTER ROLE</label>
      <select id="selectrole" name="role" required style="width: 200px; padding: 5px; border: 1px solid
#ccc; border-radius: 4px; box-sizing: border-box;margin-left: 10px;">
        <option hidden selected></option>
        {% for i in role %}<option name="role" value="{{i.id}}">{{i.name}}</option>{% endfor %}
      </select>
      <button class="btn btn-primary btn-sm">Search</button>
```

```
</form>

<hr>


<table id="myTable" class="shadow-lg table table-stripedy mt-2">

  <thead>

  <tr>

    <th>No.</th>

    <th>Name</th>

    <th>role</th>

    <th>designation</th>

    <th>department</th>

    <th>specialist</th>

    <th>gender</th>

    <th>Action</th>

  </tr>

  </thead>


  <tbody>

  {% for a in staffs %}

    <tr>

      <td>{{forloop.counter}}</td>

      <td>{{a.name}} </td>

      <td>{{a.role}}</td>

      <td>{{a.designation}}</td>

      <td>{{a.department}}</td>

      <td>{{a.specialist}}</td>

      <td>{{a.get_gender_display}}</td>

      <td>
```

```
                    <button type="button" class="btn btn-primary text-center m-1 fs-9 btn-sm" data-bs-
toggle="modal" data-bs-target="#editStaff{{a.id}}" title="Edit">
                        <i class="bi bi-pencil-square"></i>
                    </button>


                    <button type="button" class="btn btn-danger text-center m-1 fs-9 btn-sm" data-bs-
toggle="modal" data-bs-target="#deleteStaff{{a.id}}" title="View & Delete">
                        <i class="bi bi-view-list"></i>
                    </button>
                </td>
                {% include 'modals/edit_staff.html' %}
                {% include 'modals/delete_staff.html' %}
            </tr>
        {% endfor %}
        </tbody>
    </table>
</main>

{% include 'modals/add_staff.html' %}


{% endblock %}
```

Backend

Views.py

```
from django.contrib import messages
from django.http import HttpResponseRedirect, JsonResponse
from django.shortcuts import render, redirect
from django.contrib import messages
from django.utils import timezone

from .utils import login_required

from ..models import *

import datetime, json


login_attempt = {}


@login_required

def index(request):

    total_income = 0

    for apmt in Appointment.objects.all():

        total_income += apmt.amount


    doctors = list(Doctor.objects.all()[::-1])

    for doctor in doctors:

        tokens = Token.objects.filter(doctor=doctor, date=datetime.date.today())

        setattr(doctor, 'tokens', len(tokens))


    context = dict(

        date = datetime.date.today(),

        general = GeneralSetting.objects.first(),

        tokens = len(Token.objects.all()),

        appoinments = len(Appointment.objects.all()),

        staffs = Staff.objects.all()[::-1],

        doctors = doctors,

        income = total_income,

        total_visitors = len(Visit.objects.all()),

    )
    return render(request, 'index.html', context)



def login(request):
    next_url = request.GET.get('next', '/')  # Default to home page if next parameter is not provided

    if 'user' in request.session: # checking if user is logined or not
        return redirect(next_url)

    if request.method == 'POST':
        next_url = request.POST.get('next', '/')

        email = request.POST.get('email')
        password = request.POST.get('password')
        if not str(email) in login_attempt.keys():
            login_attempt[str(email)] = 0
        user = Staff.objects.filter(email=email).exists()

        # prevent brute force attack basic
        attempt = login_attempt[str(email)]
        if int(attempt) >= 3:
            messages.warning(request, "TOO MANY ATTEMPT. TRY AGAIN LATER")
            return redirect('login')

        # checking user is exist
        if not user:
            messages.warning(request, "USER DOES NOT EXIST")
            attempt = login_attempt[str(email)]
            login_attempt[str(email)] = attempt + 1
            return redirect('login')

        # checking the password is correct
        elif password != (Staff.objects.get(email=email)).password:
            messages.warning(request, 'INCORRECT PASSWORD')
            attempt = login_attempt[str(email)]
            login_attempt[str(email)] = attempt + 1
            return redirect('login')

        # after succussfully logined
        else:
            user = Staff.objects.get(email=email)
            request.session['user'] = user.email
            return redirect(next_url)

    return render(request, 'login.html', {'next': next_url})


def logout(request):
    try:
        del request.session['user']
    except: pass
    return HttpResponseRedirect('/')

@login_required
def patients(request):
    patients = Patient.objects.all()[::-1]
    if request.method == 'POST':
        name = request.POST.get("name")
        patients = Patient.objects.filter(name__startswith=name)

    context = dict(
        patients = patients,
        general = GeneralSetting.objects.first()
    )

    return render(request, 'Patients.html', context)

@login_required
def appoinment(request):
    context = dict(
        patients =  Patient.objects.all(),
        doctors = Doctor.objects.all(),
        appoinments = Appointment.objects.all()[::-1],
        shift = Shift.objects.all(),
        general = GeneralSetting.objects.first()
    )

    return render(request, 'Appointment.html', context)

@login_required
def add_appoinment(request):
    if request.method == 'POST':
        patient = Patient.objects.filter(id=int(request.POST.get('patient'))).first()
        doctor = Doctor.objects.filter(id=int(request.POST.get('doctor'))).first()
        appointed_date = request.POST.get('datetime').split('T', 1)[0]
        appointed_time = request.POST.get('datetime').split('T', 1)[1]
        shift = Shift.objects.filter(id=int(request.POST.get('shift'))).first()
        status = request.POST.get('status')
        message = request.POST.get('message')
        payment = request.POST.get('payment')
        priority = request.POST.get('priority')
        amount = request.POST.get('amount')
        token_number = request.POST.get('token')
        added_date = timezone.now()

        data = Appointment(patient=patient, doctor=doctor, appointed_date=appointed_date,
appointed_time=appointed_time, added_date=added_date, shift=shift, status=status,
message=message, payment=payment, priority=priority, amount=amount,
token_number=token_number)
        if not Appointment.objects.filter(patient=patient, doctor=doctor, appointed_date=appointed_date,
shift=shift, status=status, message=message, payment=payment, priority=priority).exists():
            data.save()
            apmnt = Appointment.objects.filter(id=int(data.id)).first()
            token = Token.objects.all()
            date = str(appointed_date)
            if not Token.objects.filter(date=date, appointment=apmnt, doctor=doctor).exists():
                token_data = Token(date=date, appointment=apmnt, doctor=doctor, token=token_number)
                token_data.save()
    return redirect('appoinment')


@login_required
def edit_appoinment(request, id):
    apmnt = Appointment.objects.filter(id=int(id)).first()
    if not apmnt:
        return redirect('appoinment')

    if request.method == 'POST':
        patient = Patient.objects.filter(id=int(request.POST.get('patient'))).first()
        doctor = Doctor.objects.filter(id=int(request.POST.get('doctor'))).first()
        appointed_date = request.POST.get('datetime').split('T', 1)[0]
        appointed_time = request.POST.get('datetime').split('T', 1)[1]
        shift = Shift.objects.filter(doctor=doctor).first()
        status = request.POST.get('status')
        message = request.POST.get('message')
        payment = request.POST.get('payment')
        priority = request.POST.get('priority')
        data = Appointment(
            id=int(id),
            patient=patient if patient else apmnt.patient,
            doctor=doctor if doctor else apmnt.doctor,
            appointed_date=appointed_date if appointed_date else apmnt.appointed_date,
            appointed_time=appointed_time if appointed_time else apmnt.appointed_time,
            added_date=apmnt.added_date,
            shift=shift if shift else apmnt.shift,
            status=status if status else apmnt.status,
            message=message if message else apmnt.message,
            payment=payment if payment else apmnt.payment,
            priority=priority if priority else apmnt.priority,
            amount=apmnt.amount,
            token_number=apmnt.token_number,
        )
        data.save()
    return redirect('appoinment')

@login_required
def delete_appoinment(request, id):
    if request.method == 'POST':
        apmnt = Appointment.objects.filter(id=int(id)).first()
        if apmnt:
            apmnt.delete()
            return redirect('appoinment')

    return redirect('appoinment')

@login_required
def approve_appoinment(request, id):
    apmnt = Appointment.objects.filter(id=int(id)).first()
    if not apmnt:
        return redirect('appoinment')

    if request.method == 'POST':
        apmnt.status = "A"
        apmnt.amount = apmnt.doctor.fees.charge
        apmnt.payment = 'C'
        apmnt.save()
    return redirect('appoinment')

@login_required
def add_patient(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        guardian = request.POST.get('guardian')
        gender = request.POST.get('gender')
        age = int(request.POST.get('age'))
        bloodgroup = request.POST.get('bloodgroup')
        phone = int(request.POST.get('phone'))

        address = request.POST.get('address')


        data = Patient(name=name, guardian=guardian, gender=gender, age=age, bloodgroup=bloodgroup,
phone=phone, address=address)

        if not Patient.objects.filter(name=name, guardian=guardian, gender=gender, age=age,
bloodgroup=bloodgroup, phone=phone, address=address).exists():

            data.save()


    return redirect('appoinment')


@login_required

def staff(request):

    context = dict(

        staffs = Staff.objects.all()[::-1],

        role = Role.objects.all(),

        designation = Designation.objects.all(),

        department = Department.objects.all(),

        specialist = Specialist.objects.all(),

        general = GeneralSetting.objects.first()

    )

    if request.method == 'POST':

        role = Role.objects.get(id=int(request.POST.get('role')))

        staffs = Staff.objects.filter(role=role)[::-1]

        context['staffs'] = staffs


    return render(request, 'staff.html', context)



@login_required
```

```
def view_staff(request):
    context = dict(
        staffs = Staff.objects.all()[::-1],
        role = Role.objects.all(),
        designation = Designation.objects.all(),
        department = Department.objects.all(),
        specialist = Specialist.objects.all(),
        general = GeneralSetting.objects.first()
    )
    return render(request, 'view_staff.html', context)


@login_required
def add_staff(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phone = request.POST.get('phone')

        if Staff.objects.filter(email=email).exists():
            messages.warning(request, f"!Email Alredy Exist")
            return HttpResponseRedirect(request.path)

        role = Role.objects.filter(id=int(request.POST.get('role'))).first()
        designation = Designation.objects.filter(id=int(request.POST.get('designation'))).first()
        department = Department.objects.filter(id=int(request.POST.get('department'))).first()
        specialist = Specialist.objects.filter(id=int(request.POST.get('specialist'))).first()
        gender = request.POST.get('gender')
        if role == Role.objects.filter(name="doctor").first():
            doctor_data = Doctor(name=name, phone=phone, designation=designation,
department=department, specialist=specialist, gender=gender)
            if not Doctor.objects.filter(name=name, phone=phone, designation=designation,
department=department, specialist=specialist, gender=gender).exists():
                doctor_data.save()


        data = Staff(name=name, email=email, password=password, role=role, phone=phone,
designation=designation, department=department, specialist=specialist, gender=gender)
        if not Staff.objects.filter(name=name, email=email, password=password, role=role, phone=phone,
designation=designation, department=department, specialist=specialist, gender=gender).exists():
            data.save()


    return redirect('staff')


@login_required
def edit_staff(request, id):
    staf = Staff.objects.filter(id=int(id)).first()
    if not staf:
        return redirect('staff')


    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phone = request.POST.get('phone')


        role = Role.objects.filter(id=int(request.POST.get('role'))).first()
        designation = Designation.objects.filter(id=int(request.POST.get('designation'))).first()
        department = Department.objects.filter(id=int(request.POST.get('department'))).first()
        specialist = Specialist.objects.filter(id=int(request.POST.get('specialist'))).first()
        gender = request.POST.get('gender')

        data = Staff(
            id=int(id),
            name=name,
            email=email,
            password=password,
            role=role,
            phone=phone,
            designation=designation,
            department=department,
            specialist=specialist,
            gender=gender,
            added_date_time=staf.added_date_time
        )
        data.save()
    return redirect('staff')

@login_required
def delete_staff(request, id):
    if request.method == 'POST':
        staf = Staff.objects.filter(id=int(id)).first()
        if staf:
            staf.delete()
            return redirect('staff')

    return redirect('staff')
```

```
@login_required

def front_office(request):

    context = dict(

        visits = Visit.objects.all()[::-1],

        general = GeneralSetting.objects.first()

    )

    return render(request, 'front_office.html', context)


@login_required

def add_visit(request):

    if request.method == 'POST':

        purpose = request.POST.get('purpose')

        name = request.POST.get('name')

        phone = request.POST.get('phone')

        id_card = request.POST.get('id_card')

        note = request.POST.get('note')


        data = Visit(purpose=purpose, name=name, phone=phone, id_card=id_card, note=note)

        if not Visit.objects.filter(purpose=purpose, name=name, phone=phone, id_card=id_card,
note=note).exists():

            data.save()


    return redirect('front_office')


@login_required

def doctor_wise_view(request):

    context = dict(

        doctors = Doctor.objects.all()

    )
    if request.method == 'POST':
        doctor = Doctor.objects.filter(id=int(request.POST.get('doctor'))).first()
        date = request.POST.get('date')

        context = dict(
            doctors = Doctor.objects.all(),
            appoinments = Appointment.objects.filter(doctor=doctor, appointed_date=date)
        )

    return render(request, 'doc_wise.html', context)
```

settings.py

```
"""
Django settings for clinic project.

Generated by 'django-admin startproject' using Django 5.0.3.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-%b!su(a7em8wr13r%-m)c(rm&1semna($n#d5wn=cjv($(%l-e'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'jazzmin',
    'django.contrib.humanize',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'main'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'clinic.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'clinic.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/
```

```
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Asia/Kolkata'

USE_I18N = True

USE_TZ = True

STATIC_URL = 'static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
MEDIA_ROOT  = BASE_DIR / "uploads"
MEDIA_URL = '/media/'
```

Urls.py

```
from django.urls import path
from .views import main_pages, settings
from . import fetches

# main page urls
urlpatterns = [
    # dashboard
    path('', main_pages.index, name='index'),
    path('index/', main_pages.index, name='index'),

    # login & logout
    path('login/', main_pages.login, name='login'),
    path('logout/', main_pages.logout, name='logout'),


    # appoinment & others
    path('appoinment/', main_pages.appoinment, name='appoinment'),
    path('staff/', main_pages.staff, name='staff'),
    path('view_staff', main_pages.view_staff, name="view_staff"),
    path('front_office/', main_pages.front_office, name='front_office'),
    path('doctor-wise/', main_pages.doctor_wise_view, name='doctor_wise'),
    path('patients/', main_pages.patients, name="patients"),
]


# main moadals urls
urlpatterns += [
    # appoinment
    path('add_appoinment/', main_pages.add_appoinment, name='add_appoinment'),
    path('edit_appoinment/<int:id>/', main_pages.edit_appoinment, name='edit_appoinment'),
    path('delete_appoinment/<int:id>/', main_pages.delete_appoinment, name='delete_appoinment'),
    path('approve_appoinment/<int:id>', main_pages.approve_appoinment,
name="approve_appoinment"),


    # patient
    path('add_patient', main_pages.add_patient, name='add_patient'),


    # staff
    path('add_staff/', main_pages.add_staff, name='add_staff'),
    path('edit_staff/<int:id>/', main_pages.edit_staff, name='edit_staff'),
    path('delete_staff/<int:id>/', main_pages.delete_staff, name='delete_staff'),


    # front office
    path('add_visit/', main_pages.add_visit, name="add_visit"),

]


# settings page urls

urlpatterns += [

    # hospital charges

    path('charge', settings.charge, name='charge'),

    path('charge_category', settings.charge_category, name='charge_category'),

    path('charge_tax', settings.charge_tax, name='charge_tax'),

    path('charge_type', settings.charge_type, name='charge_type'),

    path('charge_unit', settings.charge_unit, name='charge_unit'),


    # staff & role

    path('role/', settings.role, name="role"),

    path('designation/', settings.designation, name="designation"),

    path('department/', settings.department, name="department"),

    path('specialist/', settings.specialist, name="specialist"),


    #doctor

    path('shift/', settings.shift, name="shift"),

    path('doctor_shift/', settings.doctor_shift, name="doctor_shift"),

    path('change_shift_for_doctor/<int:doctor_id>/<int:shift_id>/', settings.change_shift_for_doctor,
name="change_shift_for_doctor"),

]



# settings modals urls

urlpatterns += [

    # add hospital charges
    path('add_charge/', settings.add_charge, name="add_charge"),
    path('add_charge_category/', settings.add_charge_category, name="add_charge_category"),
    path('add_charge_tax/', settings.add_charge_tax, name="add_charge_tax"),
    path('add_charge_type/', settings.add_charge_type, name="add_charge_type"),
    path('add_charge_unit/', settings.add_charge_unit, name="add_charge_unit"),

    # edit & delete hospital charges
    path('edit_charge/<int:id>/', settings.edit_charge, name="edit_charge"),
    path('edit_charge_category/<int:id>/', settings.edit_charge_category, name="edit_charge_category"),
    path('edit_charge_tax/<int:id>/', settings.edit_charge_tax, name="edit_charge_tax"),
    path('edit_charge_type/<int:id>/', settings.edit_charge_type, name="edit_charge_type"),
    path('edit_charge_unit/<int:id>/', settings.edit_charge_unit, name="edit_charge_unit"),
    path('delete_charges/<int:id>/<str:data_name>/<str:path>/', settings.delete_charges,
name="delete_charges"),

    #Staff settings
    path('add_role/', settings.add_role, name="add_role"),
    path('add_designation/', settings.add_designation, name="add_designation"),
    path('add_department/', settings.add_department, name="add_department"),
    path('add_specialist/', settings.add_specialist, name="add_specialist"),

    # edit & delete staff settings
    path('edit_role/<int:id>/', settings.edit_role, name='edit_role'),
    path('edit_staff_all/<int:id>/<str:class_name>/', settings.edit_staff_all, name="edit_staff_all"),
    path('delete_staff_all/<int:id>/<str:class_name>/', settings.delete_staff_all, name="delete_staff_all"),

    # doctor settings
    path('add_shift', settings.add_shift, name="add_shift"),
    # edit doctor settings
    path('edit_shift/<int:id>/', settings.edit_shift, name='edit_shift'),
]


# fetch python function from python file to javascript
urlpatterns += [
    path('get_token/<int:id>/<str:date>/', fetches.get_token, name="get_token"),
]
```

Models.py

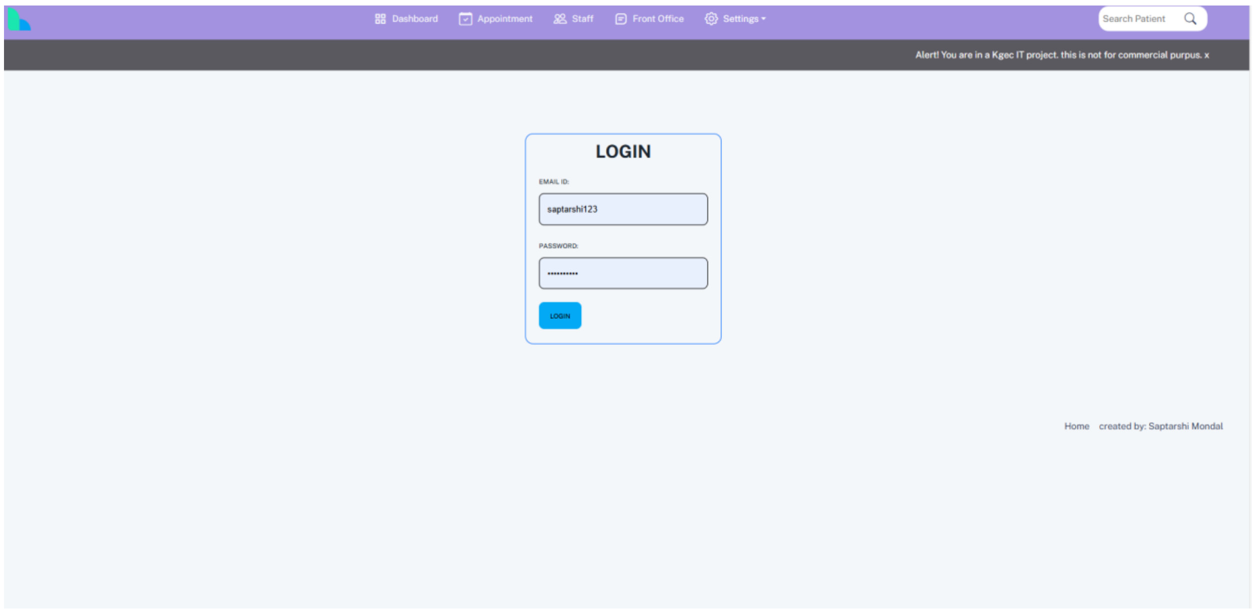
```
from django.db import models
from solo.models import SingletonModel


class Theme(models.Model):
    name = models.CharField(max_length=30)
    nav_color = models.CharField(max_length=30, null=True, blank=True)
    nav_text_color = models.CharField(max_length=30, null=True, blank=True)
    btn_color = models.CharField(max_length=30, null=True, blank=True)
    btn_text_color = models.CharField(max_length=30, null=True, blank=True)
    body_color = models.CharField(max_length=30, null=True, blank=True)
    body_text_color = models.CharField(max_length=30, null=True, blank=True)

    def __str__(self):
        return self.name

class GeneralSetting(SingletonModel):
    hospital_name = models.CharField(max_length=200, null=True, blank=True)
    hospital_code = models.CharField(max_length=200, null=True, blank=True)
    address = models.CharField(max_length=500, null=True, blank=True)
    phone_number =  models.IntegerField(null=True, blank=True)
    email = models.EmailField(null=True, blank=True)
    hospital_logo = models.ImageField(upload_to='media/')
    theme = models.ForeignKey(Theme, on_delete=models.CASCADE)

    def __str__(self):
        return self.hospital_name


# Hospital charges
class UnitType(models.Model):
    name = models.CharField(max_length=20)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class TaxCategory(models.Model):
    name = models.CharField(max_length=200)
    percentage = models.IntegerField()
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class ChargeType(models.Model):
    name = models.CharField(max_length=200)
    Appointment = models.BooleanField(default=False)
    OPD = models.BooleanField(default=False)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class ChargeCategory(models.Model):
    name = models.CharField(max_length=200)
    ChargeType = models.ForeignKey(ChargeType, on_delete=models.CASCADE)
    description = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Charge(models.Model):
    name = models.CharField(max_length=200)
    ChargeType = models.ForeignKey(ChargeType, on_delete=models.CASCADE)
    ChargeCategory = models.ForeignKey(ChargeCategory, on_delete=models.CASCADE)
    TaxCategory = models.ForeignKey(TaxCategory, on_delete=models.CASCADE)
    UnitType = models.ForeignKey(UnitType, on_delete=models.CASCADE)
    charge = models.IntegerField()
    description = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class User(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()
    password = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Role(models.Model):
    name = models.CharField(max_length=200)
    dashboard = models.BooleanField(default=True)
    appointment = models.BooleanField(default=False)
    front_office = models.BooleanField(default=False)
    staffs = models.BooleanField(default=False)
    settings = models.BooleanField(default=False)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Patient(models.Model):
    name = models.CharField(max_length=200)
    guardian = models.CharField(max_length=200, null=True, blank=True)
    gender = models.TextField(choices=[('M', 'Male'), ('F', 'Female')], null=True, blank=True)
    age = models.IntegerField(null=True, blank=True)
    bloodgroup = models.TextField(choices=[('A+', 'A+'), ('B+', 'B+'), ('AB+', 'AB+'), ('O+', 'O+')], null=True,
blank=True)
    phone = models.IntegerField(null=True, blank=True)
    address = models.CharField(max_length=400, null=True, blank=True)
    added_date_time = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.name

class Designation(models.Model):
    name = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Department(models.Model):
    name = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Specialist(models.Model):
    name = models.CharField(max_length=200)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Shift(models.Model):
    name = models.CharField(max_length=200)
    time_start = models.TimeField()
    time_end = models.TimeField()
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Staff(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField(null=True, blank=True, unique=True)
    password = models.CharField(max_length=200)
    role = models.ForeignKey(Role, on_delete=models.CASCADE, null=True, blank=True)
    phone = models.IntegerField(null=True, blank=True)
    designation = models.ForeignKey(Designation, on_delete=models.CASCADE, null=True, blank=True)
    department = models.ForeignKey(Department, on_delete=models.CASCADE, null=True, blank=True)
    specialist = models.ForeignKey(Specialist, on_delete=models.CASCADE, null=True, blank=True)
    gender = models.TextField(choices=[('M', 'Male'), ('F', 'Female')], null=True, blank=True)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Doctor(models.Model):
    name = models.CharField(max_length=200)
    fees = models.ForeignKey(Charge, on_delete=models.CASCADE, null=True, blank=True)
    designation = models.ForeignKey(Designation, on_delete=models.CASCADE)
    department = models.ForeignKey(Department, on_delete=models.CASCADE)
    specialist = models.ForeignKey(Specialist, on_delete=models.CASCADE)
    gender = models.TextField(choices=[('M', 'Male'), ('F', 'Female')], null=True, blank=True)
    phone = models.IntegerField(default=0)
    shift = models.ManyToManyField(Shift, null=True, blank=True)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Appointment(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    appointed_date = models.DateField()
    appointed_time = models.TimeField()
    added_date = models.DateTimeField(auto_now_add=True, null=True, blank=True)
    shift = models.ForeignKey(Shift, on_delete=models.CASCADE)
    status = models.TextField(choices=[('P', 'Pending'), ('A', 'Approved'), ('C', 'Cancelled')], default='P')
    message = models.TextField(blank=True, null=True)
    payment = models.TextField(choices=[('C', 'Cash'), ('U', 'UPI'), ('O', 'Other'), ('Not Paid', 'N')],
default='C')
    priority = models.TextField(choices=[('N', 'Normal'), ('U', 'Urgent')], default='N')
    amount = models.IntegerField(default=0, null=True, blank=True)
    token_number = models.IntegerField()

    def __str__(self):
        return f"{self.id}, {self.patient.name} | {self.doctor.name} | {self.status}"

class Days(models.Model):
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    monday = models.BooleanField(default=False)
    tuesday = models.BooleanField(default=False)
    wednesday = models.BooleanField(default=False)
    thursday = models.BooleanField(default=False)
    friday = models.BooleanField(default=False)
    saturday = models.BooleanField(default=False)
    sunday = models.BooleanField(default=False)
    added_date_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.doctor.name

class Visit(models.Model):
    purpose = models.CharField(max_length=500)
    name = models.CharField(max_length=200)
    related_to = models.CharField(max_length=200, null=True, blank=True)
    phone = models.IntegerField(default=0)
    id_card = models.CharField(max_length=200, null=True, blank=True)
    added_date_time = models.DateTimeField(auto_now_add=True)
    time_in = models.TimeField(null=True, blank=True)
    time_out = models.TimeField(null=True, blank=True)
    number_of_person = models.IntegerField(null=True, blank=True)
    note = models.TextField(null=True, blank=True)
    def __str__(self):
        return self.name

class Token(models.Model):
    date = models.DateField()
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    appointment = models.ForeignKey(Appointment, on_delete=models.CASCADE)
    token = models.IntegerField()

    def __str__(self):
```

```
    return f"{self.date} | {self.appointment.patient.name} | dr. {self.appointment.doctor.name}"
```

Requirements.txt

```
aiohttp==3.9.3
beautifulsoup4==4.12.3
bs4==0.0.2
Django==5.0.2
django-jazzmin==2.6.0
django-solo==2.2.0
Pillow
```

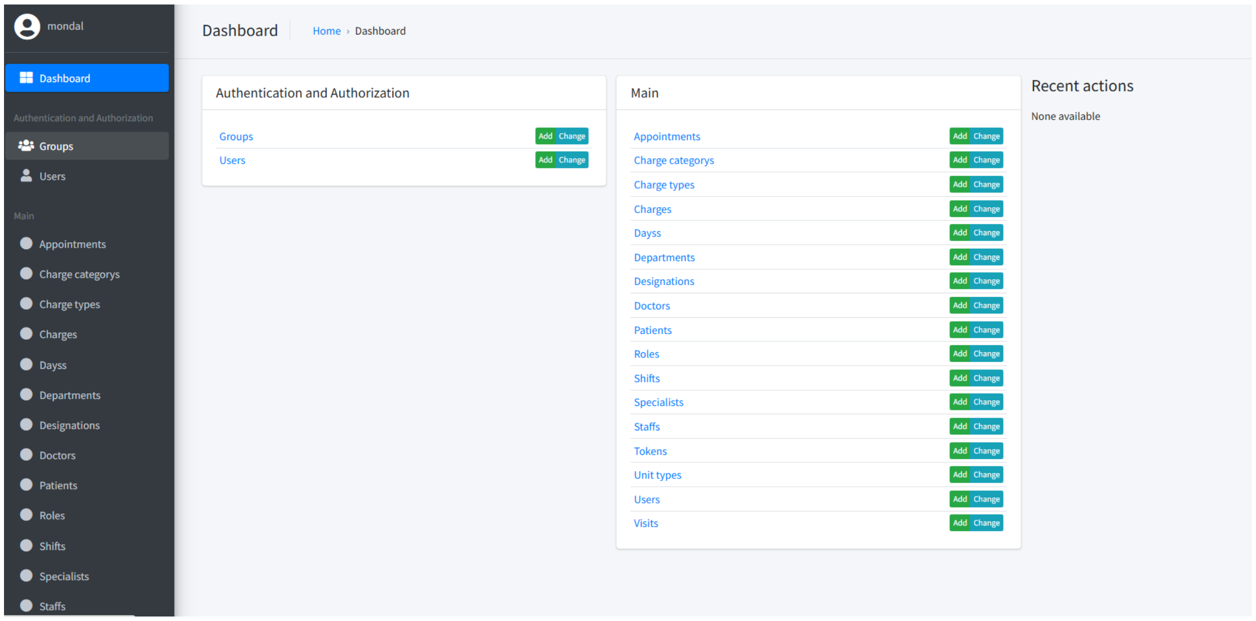
Screenshots:



Alert! You are in a Kgec IT project. this is not for commercial purpus. x

saptarshi123

created by: Saptarshi Mondal



Dashboard   Home › Dashboard

Authentication and Authorization

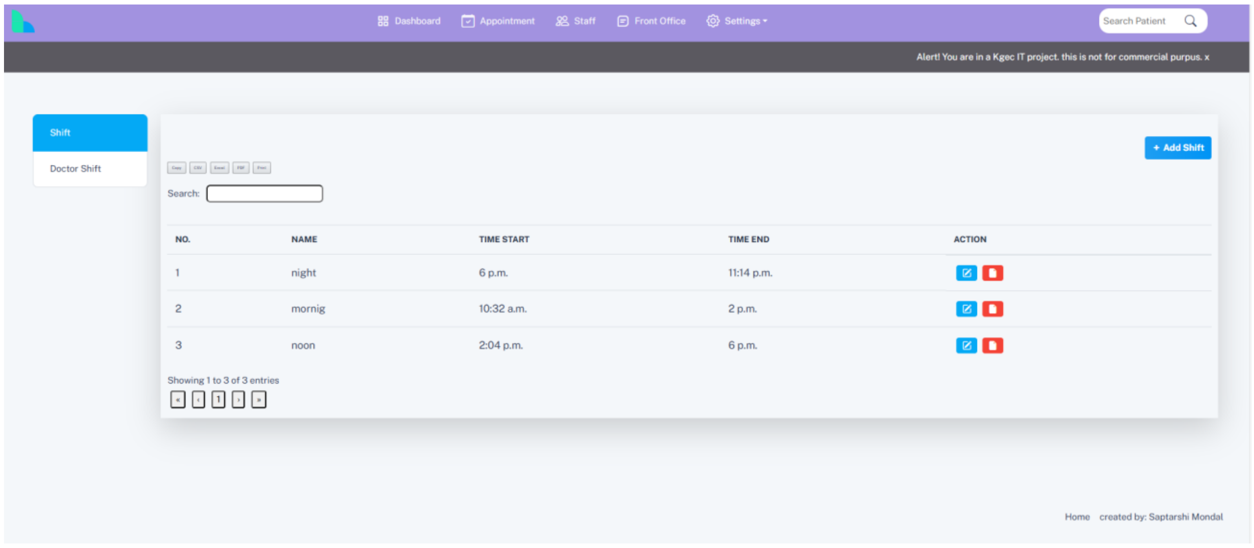
Groups	Add Change
Users	Add Change

Main

Appointments	Add Change
Charge categorys	Add Change
Charge types	Add Change
Charges	Add Change
Dayss	Add Change
Departments	Add Change
Designations	Add Change
Doctors	Add Change
Patients	Add Change
Roles	Add Change
Shifts	Add Change
Specialists	Add Change
Staffs	Add Change
Tokens	Add Change
Unit types	Add Change
Users	Add Change
Visits	Add Change

Recent actions

None available

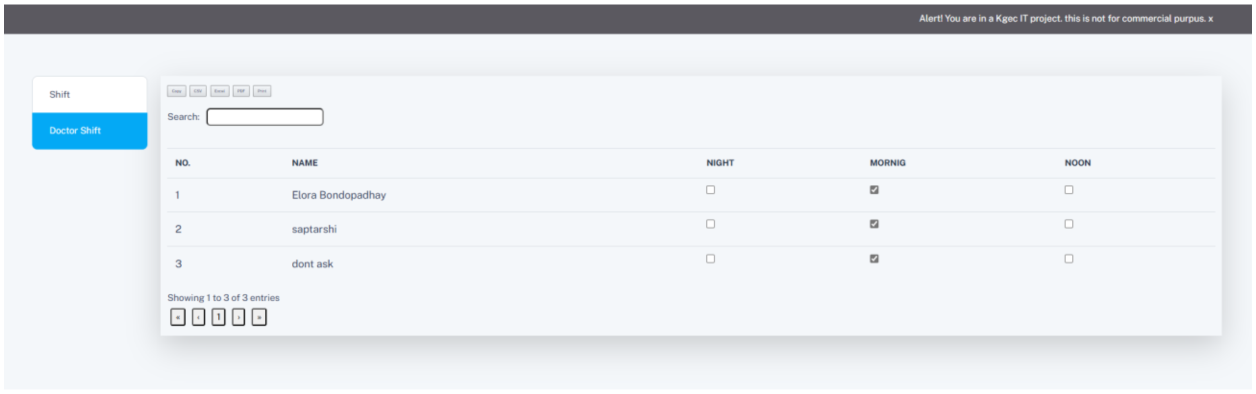


Alert! You are in a Kgec IT project. this is not for commercial purpus. x

Search:

NO.	NAME	TIME START	TIME END	ACTION
1	night	6 p.m.	11:14 p.m.	
2	mornig	10:32 a.m.	2 p.m.	
3	noon	2:04 p.m.	6 p.m.	

Showing 1 to 3 of 3 entries



Alert! You are in a Kgec IT project. this is not for commercial purpus. x

Search:

NO.	NAME	NIGHT	MORNIG	NOON
1	Elora Bondopadhay	☐	☑	☐
2	saptarshi	☐	☑	☐
3	dont ask	☐	☑	☐

Showing 1 to 3 of 3 entries

Conclusion

The Clinic Management System is designed to revolutionize the way clinics operate, offering a comprehensive solution to streamline administrative and clinical workflows. By integrating features such as patient management, appointment scheduling, billing, and inventory tracking, the system reduces manual workloads, improves accuracy, and enhances overall efficiency.

With a user-friendly interface and robust security measures, the system ensures accessibility and data protection for all stakeholders, including patients, doctors, staff, and administrators. The scalability and modular design allow the system to adapt to the evolving needs of clinics, ensuring long-term relevance and usability.

This system not only optimizes daily operations but also improves patient satisfaction by providing seamless access to healthcare services. By leveraging modern technology and adhering to regulatory standards, the Clinic Management System is positioned as a vital tool for clinics aiming to provide high-quality and efficient healthcare services.