

# HOW TO RUN

You will need to run the backend and frontend in separate terminals. The backend itself requires three concurrent processes (the web server, the Celery worker, and the Celery beat scheduler).

## Prerequisites

Before you start, ensure you have the following software installed and running on your system:

1. Python (3.8+): To run the Django backend.
2. Node.js (16+): To run the React frontend.
3. PostgreSQL: As the main database.
4. Redis: As the message broker for real-time notifications and background tasks.

---

## 1. Backend Setup

You will need three separate terminals for this part.

In your first terminal:

1. Navigate to the Backend Directory:

```
1 cd smart-queue-system/backend
```

2. Create and Activate a Python Virtual Environment:

\* On Windows:

```
1 python -m venv venv  
2 .\venv\Scripts\activate
```

\* On macOS/Linux:

```
1 python3 -m venv venv  
2 source venv/bin/activate
```

### 3. Install Dependencies:

```
1 pip install -r requirements.txt
```

### 4. Configure Environment Variables:

\* Create a new file named .env inside the smart-queue-system/backend directory.  
\* Add the following content to it, making sure to replace the placeholder values with your actual local PostgreSQL and Redis connection details.

```
1 # Generate a new secret key. You can use an online generator.  
2 DJANGO_SECRET_KEY='your-strong-secret-key-here'  
3 DEBUG=True  
4  
5 # Example for local PostgreSQL and Redis  
6 DATABASE_URL='postgres://<user>:<password>@localhost:5432/<database_name>'  
7 REDIS_URL='redis://localhost:6379/0'
```

### 5. Run Database Migrations:

```
1 python manage.py migrate
```

### 6. Start the Django Web Server:

```
1 python manage.py runserver
```

Keep this terminal running. Your backend API is now available at <http://localhost:8000>.

---

In your second terminal:

1. Navigate and Activate: Go to smart-queue-system/backend and activate the virtual environment as you did before.
2. Start the Celery Worker: This process handles background tasks like sending notifications.

1 celery -A core worker -l info

Keep this terminal running.

---

In your third terminal:

1. Navigate and Activate: Go to smart-queue-system/backend and activate the virtual environment.
2. Start the Celery Beat Scheduler: This process schedules periodic tasks.

1 celery -A core beat -l info

Keep this terminal running.

---

2. Frontend Setup

In a new, fourth terminal:

1. Navigate to the Frontend Directory:

```
1 cd smart-queue-system/frontend
```

2. Install Dependencies:

```
1 npm install
```

3. Start the React Development Server:

```
1 npm start
```

Keep this terminal running. Your frontend application is now available at <http://localhost:3000>. You can open this URL in your web

browser to use the application.

## CREATE ADMIN AND STAFFS

To get started, you first need to create an admin account. Once you have an admin account, you can create staff accounts through the web interface.

### 1. Create Your First Admin User

You can create an administrator account using a built-in Django command.

1. Open a terminal and navigate to the backend directory:

```
1 cd smart-queue-system/backend
```

2. Activate your Python virtual environment if it's not already active:

\* On Windows: .\venv\Scripts\activate

\* On macOS/Linux: source venv/bin/activate

3. Run the `createsuperuser` command:

```
1 python manage.py createsuperuser
```

4. Follow the prompts to choose a username, email, and password. This user will automatically be created with the admin role.

2. Log in as an Admin

- \* Go to the application's login page (e.g., `http://localhost:3000/login`).

- \* Enter the credentials you just created.

- \* You will be automatically redirected to the Admin Dashboard.

3. Create a Staff User

Now that you are logged in as an admin, you can create accounts for your staff:

1. On the Admin Dashboard, click on the "User Management" tab.

2. Click the "Add User" button.

3. Fill out the form for the new user, and make sure to select "Staff" from the "Role" dropdown menu.

4. Click "Save".

You now have a staff account. You can log out and log back in with the new staff user's credentials to access the Staff Dashboard.



> even after logging in using admin credentials it is redirecting me to student dashboard, put admin dashboard on another available port if

possible

◆ You've encountered a bug that I can fix. The issue isn't with the port, but with how the user's role is identified after login.

Putting the admin dashboard on a separate port would be complex and is not the standard way to solve this. The correct solution is to fix the

authentication process so that it properly recognizes you as an admin and sends you to the right page.

The bug is that the backend is not including the user's role (admin, staff, or student) inside the authentication token when you log in. The

frontend code then can't see the role and defaults to sending you to the student dashboard.

I will now fix the backend to include the role in the token. This will resolve the incorrect redirection.

---

## **DEMO**

### Introduction

"Good morning/afternoon. We are demonstrating the Smart Queue Management System, a web application designed to eliminate physical queues on a campus for services like the canteen, library, or admin office. It provides a seamless, real-time, and data-driven experience for administrators, staff, and students."

---

### Act 1: The Administrator (The Setup)

(This act shows how the system is configured from the top down.)

#### 1. Log in as Admin:

- \* Start at the login page.
- \* Say: "First, I will log in as the system Administrator."
- \* Enter your admin credentials.
- \* As it redirects, say: "The system recognizes my 'admin' role and takes me directly to the Admin Dashboard, which gives me a complete overview of the entire system."

#### 2. Create a Staff User:

- \* Navigate to the "User Management" tab.
- \* Say: "As an admin, I can manage all user accounts. I'll create an account for a new canteen staff member."
- \* Click "Add User". Fill in the details (e.g., username: canteen\_staff) and select the 'Staff' role from the dropdown. Click "Save".
- \* Point to the new user in the list.

### 3. Create and Configure a Service:

- \* Navigate to the "Service Management" tab.
- \* Say: "Now, I'll create a new service that students can join. Let's create a lunch pickup service."
- \* Click "Add Service" and create a service named Canteen - Lunch Pickup.
- \* After it's created, click "Edit" on that service.
- \* Say: "Finally, I will assign our new staff member to manage this specific service."
- \* In the "Assign Staff" box, select canteen\_staff and save.

### 4. Log Out:

- \* Say: "The setup is complete. The service is active and is managed by a specific staff member. I will now log out of the admin account."
- 

## Act 2: The Student (The Experience)

(This act shows the seamless student experience.)

### 1. Log in as a Student:

- \* Say: "Now, let's see the experience from a student's perspective."
- \* Log in with a regular student account.
- \* As it redirects, say: "The system identifies me as a student and takes me to my dashboard."

### 2. Join the Queue:

- \* Point to the "Available Services" section.
- \* Say: "Here I can see all the services available on campus, including the 'Canteen - Lunch Pickup' service the admin just created. I'll join the queue."
- \* Click the "Join Queue" button for that service.

### 3. View Real-time Status:

- \* Point to the "My Queues" widget on the right.
- \* Say: "Instantly, I've been added to the queue and assigned a token number. My status is 'waiting'. The key here is that I don't have to physically stand in line; I can now study in the library or walk around campus and monitor my status from my phone."

---

## Act 3: The Staff & The Interaction (The Core Loop)

(This act showcases the real-time communication between staff and student.)

### 1. Log in as Staff:

- \* (You can have this open in a separate browser window or log in again).
- \* Say: "While the student is waiting, let's log in as the canteen staff member we created."
- \* Log in with the canteen\_staff credentials.
- \* Say: "The system correctly identifies this user as staff and directs them to the Staff Dashboard."

### 2. Manage the Live Queue:

- \* On the Staff Dashboard, select the Canteen - Lunch Pickup service from the dropdown.
- \* Say: "The staff member can see all the students in their queue. Notice that our student appeared here in real-time as soon as they joined, without any need for a refresh."

### 3. Call the Student (The "Smart" Moment):

- \* Say: "Now, the staff is ready for the next person. I will click 'Call Next'."

- \* Click "Call Next".
- \* IMMEDIATELY switch to the student's browser window.
- \* Point to the student's screen and say: "And as you can see, the student's dashboard updated instantly. A prominent notification tells them it's their turn and where to go. This real-time communication is the core of the system."

#### 4. Complete the Service:

- \* Switch back to the staff member's view.
- \* Say: "On the staff side, the student is marked as 'in progress'. Once the student has picked up their lunch, the staff member simply clicks 'Complete'."
- \* Click the "Complete" button. The student disappears from the queue.

---

### Conclusion: The Data-Driven Advantage

#### 1. Return to Admin View:

- \* Log back in as the administrator.
- \* Navigate to the "Analytics" tab.
- \* Say: "Finally, every action in the system is logged. This provides administrators with valuable data."

#### 2. Showcase Analytics:

- \* Point to the charts.
- \* Say: "Here, we can see the analytics for today. The chart for 'Canteen - Lunch Pickup' now shows one user who was successfully served.  
Over time, this data allows administrators to identify peak hours, optimize staff allocation, and improve service efficiency across the entire campus."

### **3. Final Statement:**

\* "This concludes the demo of the Smart Queue Management System, which transforms traditional queues into an efficient, real-time, and data-driven process for everyone."