First: Deployment environment & steps

❖ Backend Stack

- **Platform:** Node.js (v22)
- **Framework:** Express.js
- **Database:** MongoDB (tested locally and ready for MongoDB Atlas)
- **Hosting Options:** Render (for backend), Firebase Hosting (for frontend), MongoDB Atlas (for DB in production)

❖ Environment Requirements

Component	Version
Node.js	18+ (v22 tested)
MongoDB	6.0+ (local or Atlas)
npm	9+
Git	Latest

❖ Deployment Instructions (Backend on Render)

1. **Push your project to GitHub** (public or private repo)
2. **Create a new Render Web Service**
    - Connect to your GitHub repo
    - Select branch (usually `main`)
    - Set `Build Command`: `npm install`
    - Set `Start Command`: `node server.js` or `npm start`
3. **Set Environment Variables:**

    - `MONGO_URI` → Your MongoDB connection string (local or Atlas)
    - `JWT_SECRET` → A secure random string
    - `PORT` → `5000` (optional; defaults to 5000)
4. **Enable Auto Deploys**
5. **Click Deploy** → Wait for logs to show "Server is running on port 5000"

✅ Your backend is now live on a Render URL like:
`https://skillsync-backend.onrender.com`

Second: Installation Guide (Local Setup)

This guide describes how to install and set up the SkillSync backend application on a local system.

1. Overview

SkillSync is a Node.js-based backend system built to manage user registration, resume analysis, skill matching, and role-based access for students, mentors, and admins. This guide walks you through installation, configuration, and initial startup.

2. System Requirements

Before installing SkillSync, ensure your system meets the following requirements:

Component	Version
Operating System	Windows 10/11, macOS, or any Linux distro
Node.js	18+ (v22 tested)
MongoDB	6.0+ (local or Atlas)
npm	Version 9 or higher (bundled with Node.js)
MongoDB	Version 6.0+ (local or cloud - MongoDB Atlas)
Git	Latest

3. Installation Steps

- **Step 1: Download or Clone the Project**

Open your terminal and run:

```
{ git clone https://github.com/your-username/skillsync-backend.git}
```

Then navigate into the project folder:

```
{cd skillsync-backend }
```

- **Step 2: Install Dependencies**

To install all required packages:

```
{npm install}
```

This command downloads all backend libraries (like Express, Mongoose, Multer, etc.).

- **Step 3: Create the Environment Configuration File**

Create a new file in the root directory named `.env`.

This file contains sensitive project settings such as database credentials and secrets. Use any text editor (e.g., VS Code, Notepad) to create and edit it.

4. Verifying the Installation

You can confirm the backend is working by visiting:

```
{http://localhost:5000/}
```

Expected Output:

```
{SkillSync Backend API}
```

You can also test features like registration, login, and resume upload using Postman or your connected frontend.

Third: Configuration Settings (.env file)

Add the following lines to your `.env` file:

```
{MONGO_URI=mongodb://localhost:27017/skillsync
JWT_SECRET=your_jwt_secret_here
PORT=5000 }
```

Setting	Description
`MONGO_URI`	MongoDB connection string. For local databases use `localhost`, for production use MongoDB Atlas URI.
`JWT_SECRET`	A random secret string used to securely sign JWT tokens. Keep this private
`PORT`	The port on which the backend will run. Use `5000` unless you need to change it.

Note: Do not commit your `.env` file to any public repository. It contains private keys.

- **Step 4: Start the MongoDB Database**

If you're using MongoDB locally, open a new terminal and start the MongoDB server:

```
{mongod}
```

If you're using MongoDB Atlas, ensure your URI in `.env` is correct and the cluster is running.

- **Step 5: Run the Backend Server**

To start the server in development mode (auto restarts on code changes):

```
{npm run dev}
```

For production:

```
{node server.js}
```

If everything is correct, you'll see output like:

```
{Server is running on port 5000
Connected to MongoDB}
```

Verifying the Installation

You can confirm the backend is working by visiting:

```
{http://localhost:5000/}
```

Expected Output:

```
{SkillSync Backend API}
```

You can also test features like registration, login, and resume upload using Postman or your connected frontend.

Forth: User guide for key functionalities

❖ 1. User Registration & Login

- **Endpoints:**
  - `POST /api/auth/register` — register as a student, mentor, or admin
  - `POST /api/auth/login` — login and receive JWT token
- **Fields:**
  - `name`, `email`, `password`, `role` (`student`, `mentor`, `admin`), `subscription` (`free` or `premium`)
- **Response:**
  - On success: JWT token and user object
  - On error: appropriate status codes (`400`, `401`, etc.)

❖ 2. Resume Upload & Analysis (Students)

- **Endpoint:** `POST /api/resume/upload`
- **Authentication:** Required (JWT)
- **File Type:** PDF or DOCX only
- **Functionality:**
  - Extracts text from resume
  - Uses either:
    - `KeywordResumeAnalyzer` (for free users)
    - `AIResumeAnalyzer` (for premium users)
  - Returns:

        - Resume score
        - Matched skills and keywords
        - Feedback string

❖ 3. Skill Assessment

- **Trigger:** Integrated with resume upload or standalone endpoint *(to be added or extended)*
- **Planned Workflow:**
    - Extracts and matches skills from resume or user profile
    - Compares against required job/industry skills
    - Returns assessment gaps (missing skills)
- **Future endpoint example:**
    - `GET /api/skills/gap-analysis` — *(coming phase)*

❖ 4. Progress Tracking (Per User)

- **Goal:** Track improvements across multiple resume uploads or skills
- **Current Support:**
    - Resume uploads stored per user
    - Could display analysis trends over time
- **Planned endpoint:**
    - `GET /api/users/:id/progress` — *(future feature for frontend integration)*

❖ 5. Mentor Communication (Real-Time or Asynchronous)

- **Role:** Mentor
- **Features:**
    - Chat with students (planned via WebSockets or Firestore chat)
    - View student resume summaries or progress

    - Provide advice/feedback

- **Planned endpoints:**
    - `GET /api/mentors/students`
    - `POST /api/chat/send` *(if chat system is backend-controlled)*

❖ 6. Admin Functionalities

Action	Endpoint	Method	Description
View All Users	`/api/admin/users`	GET	Returns list of all registered users
Delete User	`/api/admin/users/:id`	DELETE	Removes user by ID
Get Stats (planned)	`/api/admin/stats`	GET	Dashboard analytics (planned feature)

❖ 7. Notifications System (Observer Pattern)

- Sends feedback notifications to users after resume upload
- **Extensible** to email, SMS, or in-app
- Works internally without endpoint

❖ Role-Based Access Summary

Role	Allowed Actions
Student	Register, Login, Upload Resume, Receive Feedback
Mentor	Register, Login, View/Guide Students *(planned)*
Admin	Manage all users, monitor activity, delete accounts

❖ Notes on Authentication

- All sensitive routes are protected via JWT-based authentication middleware
- Role-based middleware ensures correct access level
- Token is passed via `Authorization: Bearer <token>` header in each request