**Please note:** This is a conceptual guide. You'll need to create the actual repository and push your existing project files to it.

**Instructions to Run the Project (Local Development)**

Here are the step-by-step instructions to get your Student Management System running locally:

**1. Clone the Repository:**

If you haven't already, clone your GitHub repository to your local machine:

git clone https://github.com/ZxNinja/Bapor-student-management-system.git

cd your-repository-name

**2. Backend Setup (Django)**

Navigate into your Django backend directory:

cd sms\_backend

**a. Create a Virtual Environment:** It's highly recommended to use a virtual environment to manage project dependencies.

python -m venv venv

**b. Activate the Virtual Environment:**

* **On macOS / Linux:**

source venv/bin/activate

* **On Windows (Command Prompt):**

venv\Scripts\activate.bat

* **On Windows (PowerShell):**

PowerShell

venv\Scripts\Activate.ps1

**c. Install Dependencies:** Install all the required Python packages listed in requirements.txt.

pip install -r requirements.txt

**d. Environment Variables:** Create a .env file in the sms\_backend directory based on .env.example. This file should contain sensitive information like your SECRET\_KEY and DATABASE\_URL.

* **Create .env:**

cp .env.example .env

* **Edit .env:** Open the newly created .env file and fill in your details. For local development, you can use SQLite and a simple debug setting:
* DJANGO\_SECRET\_KEY='your-insecure-local-secret-key' # CHANGE THIS FOR PRODUCTION!
* DJANGO\_DEBUG='True'
* DATABASE\_URL='sqlite:///db.sqlite3'

(You will need to install python-decouple and dj\_database\_url if you are using .env files for configuration. Make sure they are in your requirements.txt.)

**e. Apply Migrations:** This will set up your database tables based on your Django models.

python manage.py makemigrations

python manage.py migrate

* Remember to explicitly specify the app name for makemigrations if you only want to make migrations for a specific app, e.g., python manage.py makemigrations students. If you just run makemigrations it will detect changes across all apps. Then python manage.py migrate applies them.

**f. Create a Superuser (for Admin Panel):** This allows you to access the Django admin interface to manage data.

python manage.py createsuperuser

Follow the prompts to create a username, email, and password.

**g. Run the Django Development Server:**

python manage.py runserver

Your backend API will now be running at <http://127.0.0.1:8000/>.

You can access the Django admin panel at <http://127.0.0.1:8000/admin/>

and your API status at http://127.0.0.1:8000/ (as per your urls.py).

**3. Frontend Setup (HTML/JavaScript)**

Your frontend is a simple HTML file. You can open frontend/index.html directly in your web browser.

* **Open frontend/index.html in your browser.**

The JavaScript in index.html is configured to interact with your running Django backend at http://127.0.0.1:8000/api. Ensure that your Django server is running before opening the frontend.

**Important Considerations for Deployment (e.g., to Render.com):**

* **settings.py:** Ensure your ALLOWED\_HOSTS, CORS\_ALLOWED\_ORIGINS, and CSRF\_TRUSTED\_ORIGINS are correctly configured for your production domain (e.g., https://your-backend-service-name.onrender.com and https://zxninja.github.io).
* **Environment Variables:** Do *not* hardcode sensitive keys in settings.py. Use environment variables (e.g., os.environ.get('DJANGO\_SECRET\_KEY')). Render.com allows you to set these in their dashboard.
* **Static Files:** Your settings.py already includes configuration for WhiteNoise to serve static files in production, which is crucial for deployment platforms like Render.
* **Database:** For production, you'll typically use a PostgreSQL database, not SQLite. Render provides easy integration with PostgreSQL. Update your DATABASE\_URL in your Render environment variables accordingly.