Suggest me a school exam and progress report school management project based on recommendations given by you. Provide the directory structure and code as well for the entire project

Certainly! Here’s a **School Exam and Progress Report Management System** built using **Next.js for frontend** and **Node.js with Express for backend**, optimized for deployment on **Vercel**.

**Project Overview**

This system allows:

* **Admins** to manage students, exams, results, and progress reports.
* **Teachers** to input student marks.
* **Students** to view their grades and progress reports.
* **Authentication** via NextAuth.js..
* **Database** with PostgreSQL using Prisma.

**Directory Structure**

school-management-system/

├── frontend/ # Next.js app

│ ├── pages/

│ │ ├── index.js # Dashboard

│ │ ├── login.js # Auth page

│ │ ├── students/ # Student management

│ ├── components/ # Reusable UI elements

│ ├── styles/ # Styling (CSS/Tailwind)

│ ├── utils/ # Helper functions

│ ├── services/ # API calls

│ ├── next.config.js

│ ├── package.json

│ └── .env

│

├── backend/ # Node.js API

│ ├── src/

│ │ ├── routes/

│ │ │ ├── auth.js # Login / register

│ │ │ ├── students.js # Student operations

│ │ │ ├── exams.js # Exams & results

│ │ │ ├── reports.js # Progress reports

│ │ ├── controllers/ # Business logic

│ │ ├── models/ # Prisma schema

│ │ ├── middleware/ # Authentication & validation

│ ├── server.js # Express server

│ ├── prisma/

│ │ ├── schema.prisma # Database schema

│ ├── package.json

│ ├── .env

│ └── README.md

**Backend (Node.js + Express)**

**Server Setup (server.js)**

javascript

import express from 'express';

import cors from 'cors';

import dotenv from 'dotenv';

import authRoutes from './routes/auth.js';

import studentRoutes from './routes/students.js';

import examRoutes from './routes/exams.js';

import reportRoutes from './routes/reports.js';

dotenv.config();

const app = express();

app.use(cors());

app.use(express.json());

app.use('/api/auth', authRoutes);

app.use('/api/students', studentRoutes);

app.use('/api/exams', examRoutes);

app.use('/api/reports', reportRoutes);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**Database Schema (prisma/schema.prisma)**

prisma

model Student {

id String @id @default(uuid())

name String

email String @unique

exams Exam[]

}

model Exam {

id String @id @default(uuid())

subject String

score Int

student Student @relation(fields: [studentId], references: [id])

studentId String

}

**Frontend (Next.js)**

**Login Page (pages/login.js)**

javascript

import { signIn } from "next-auth/react";

export default function Login() {

return (

<div>

<h1>Login to School Management System</h1>

<button onClick={() => signIn('google')}>Login with Google</button>

</div>

);

}

**Dashboard (pages/index.js)**

javascript

import { useSession } from "next-auth/react";

export default function Dashboard() {

const { data: session } = useSession();

if (!session) return <p>Please login</p>;

return <h1>Welcome {session.user.name}</h1>;

}

**Deployment**

1. **Backend on Vercel**:
   * Use Vercel serverless functions (backend/src/api/).
   * Alternatively, deploy on **Railway** for persistent connections.
2. **Frontend on Vercel**:
   * npm run build
   * vercel deploy

**Starting a fresh project once again**

Let's build your **School Exam and Progress Report Management System** step by step from scratch. We'll go through **backend setup**, **frontend implementation**, and **deployment** to Vercel.

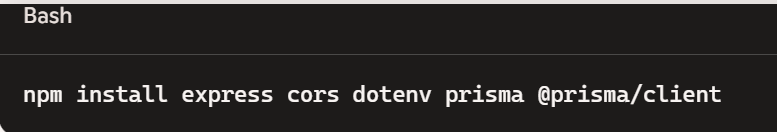
**🔹 Step 1: Initialize Your Project**

**1️. Setup the Backend**

Navigate to your workspace and create the backend project:



Install essential dependencies:

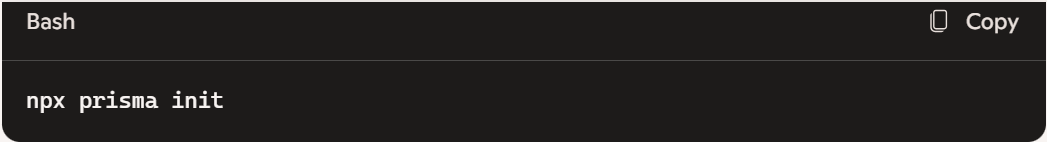


Install nodemon for development:



**🔹 Step 2: Configure Prisma for PostgreSQL**

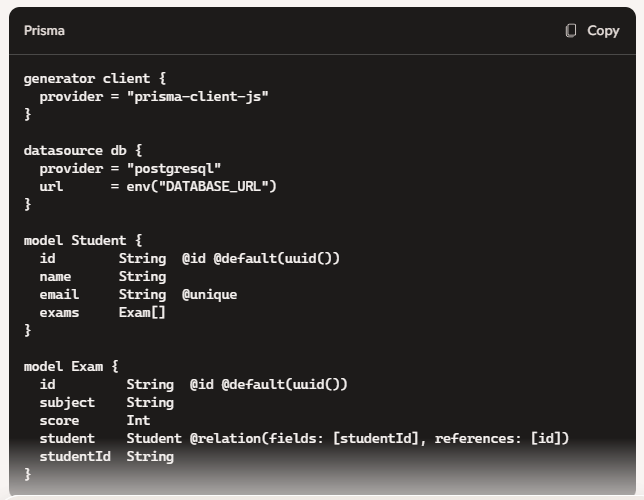
**1️ Initialize Prisma**

****

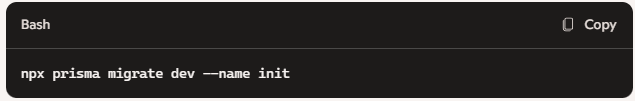
This creates a prisma/schema.prisma file and an .env file.

**2️. Define Your Database Schema (prisma/schema.prisma)**

Replace its contents with:



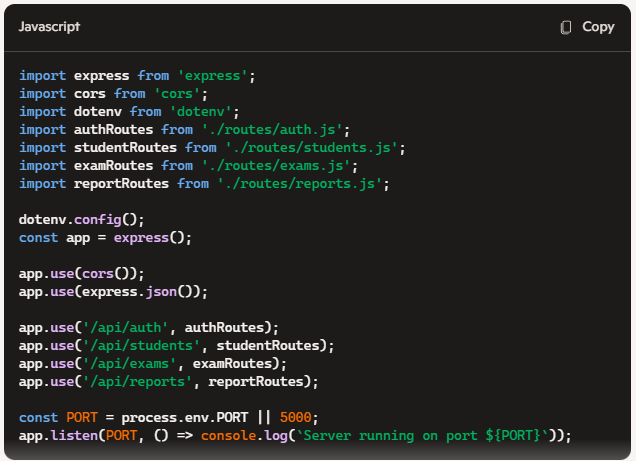
3️. **Migrate the Database**



**🔹 Step 3: Backend Server Setup**

**1️. Create server.js**

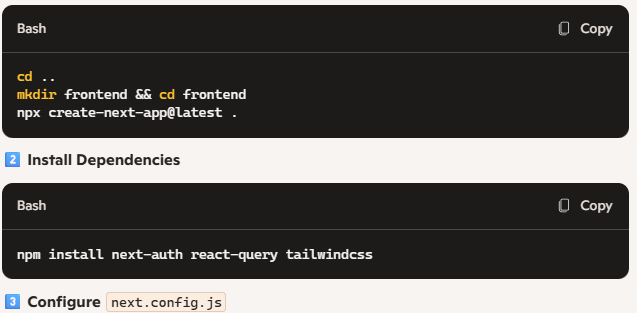
Inside backend/, create server.js and add:

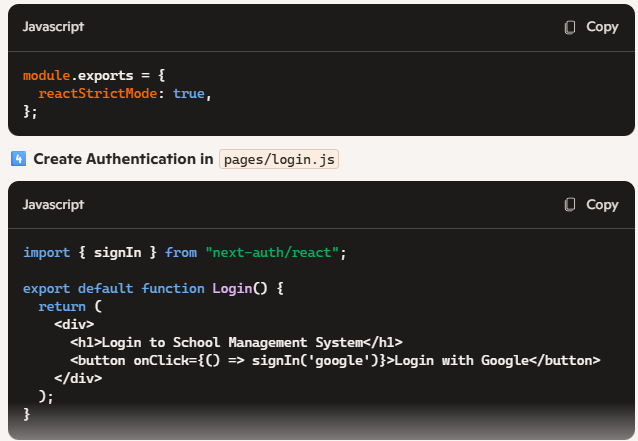


**🔹 Step 4: Set Up Frontend (Next.js)**

**1️. Initialize Next.js**

Navigate back to the root project directory:

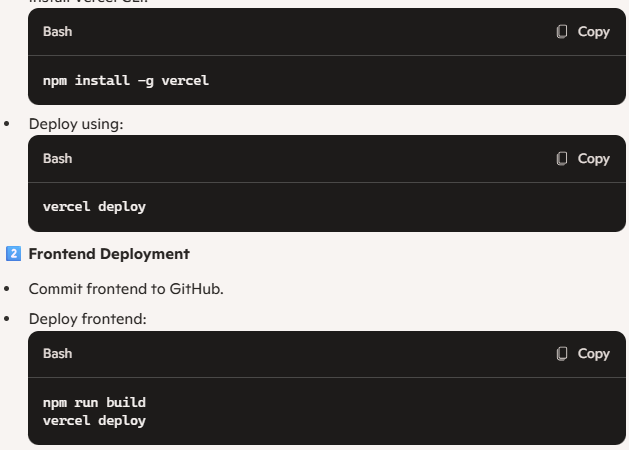




**🔹 Step 5: Deployment**

**1️. Backend Deployment (Vercel)**

* Commit your backend code to GitHub.
* Install Vercel CLI:



This gives you a **fresh start** for your project! 🚀 Let me know if you run into any hurdles along the way.

**Result at Terminal console while following the steps mentioned**

15 packages are looking for funding

(base) PS C:\Users\jagdevsinghdosanjh.RHINO\school-mamagement-system\backend> npx prisma init

>>

Fetching latest updates for this subcommand...

✔ Your Prisma schema was created at prisma/schema.prisma

You can now open it in your favorite editor.

Next steps:

1. Set the DATABASE\_URL in the .env file to point to your existing database. If your database has no tables yet, read https://pris.ly/d/getting-started

2. Set the provider of the datasource block in schema.prisma to match your database: postgresql, mysql, sqlite, sqlserver, mongodb or cockroachdb.

3. Run prisma db pull to turn your database schema into a Prisma schema.

4. Run prisma generate to generate the Prisma Client. You can then start querying your database.

5. Tip: Explore how you can extend the ORM with scalable connection pooling, global caching, and real-time database events. Read: https://pris.ly/cli/beyond-orm

More information in our documentation:

https://pris.ly/d/getting-started

(base) PS C:\Users\jagdevsinghdosanjh.RHINO\school-mamagement-system\backend>

**School Management System commands**

Command under prisma

***npx prisma studio***

Command under backend

***node server.js***

Command under frontend

***npm start***

Give me a code to register the users and enter the students in students table enter the marks in exams table so that I can fetch these records

Sure! Here’s a **complete backend setup** for **registering users**, **adding students**, and **recording exam marks** using **Node.js, Express, Prisma, and PostgreSQL**.

**📂 Backend Routes**

Save these files in your backend/routes/ folder:

**🔹 auth.js – User Registration & Login**

javascript

import express from 'express';

import bcrypt from 'bcrypt';

import jwt from 'jsonwebtoken';

import { PrismaClient } from '@prisma/client';

const router = express.Router();

const prisma = new PrismaClient();

// User Registration

router.post('/register', async (req, res) => {

const { name, email, password } = req.body;

try {

const hashedPassword = await bcrypt.hash(password, 10);

const user = await prisma.user.create({

data: { name, email, password: hashedPassword },

});

res.status(201).json({ message: 'User registered successfully', user });

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Error creating user' });

}

});

// User Login

router.post('/login', async (req, res) => {

const { email, password } = req.body;

try {

const user = await prisma.user.findUnique({ where: { email } });

if (!user || !(await bcrypt.compare(password, user.password))) {

return res.status(401).json({ error: 'Invalid credentials' });

}

const token = jwt.sign({ userId: user.id }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.json({ token });

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Login failed' });

}

});

export default router;

**🔹 students.js – Add & Fetch Students**

javascript

import express from 'express';

import { PrismaClient } from '@prisma/client';

const router = express.Router();

const prisma = new PrismaClient();

// Get All Students

router.get('/', async (req, res) => {

try {

const students = await prisma.student.findMany();

res.json(students);

} catch (error) {

console.error('Error retrieving students:', error);

res.status(500).json({ error: 'Error retrieving students' });

}

});

// Add Student

router.post('/', async (req, res) => {

const { name, email } = req.body;

try {

const student = await prisma.student.create({ data: { name, email } });

res.status(201).json(student);

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Error creating student' });

}

});

export default router;

**🔹 exams.js – Enter & Fetch Marks**

javascript

import express from 'express';

import { PrismaClient } from '@prisma/client';

const router = express.Router();

const prisma = new PrismaClient();

// Get All Exams

router.get('/', async (req, res) => {

try {

const exams = await prisma.exam.findMany({ include: { student: true } });

res.json(exams);

} catch (error) {

console.error('Error retrieving exams:', error);

res.status(500).json({ error: 'Error retrieving exams' });

}

});

// Enter Marks for a Student

router.post('/', async (req, res) => {

const { subject, score, studentId } = req.body;

try {

const exam = await prisma.exam.create({

data: { subject, score, studentId },

});

res.status(201).json(exam);

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Error entering marks' });

}

});

export default router;

**🚀 Setup & Testing**

✅ **Ensure PostgreSQL is running** ✅ **Update** .env **file with database details** ✅ **Run Prisma migrations**

bash

npx prisma migrate dev --name init

npx prisma generate

✅ **Start the backend**

bash

nodemon server.js

✅ **Test the API using Postman or browser**

Try this setup and let me know if you need tweaks! 🚀🔥