cd backend
npm init -yes
npm install prisma @prisma/client
npx prisma init

.env
DATABASE_URL="file:./dev.db"

prisma/schema.prisma
```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}
model User {
  id       Int     @id @default(autoincrement())
  name     String
  email    String  @unique
  age      Int?
  createdAt DateTime @default(now())
}
```

npx prisma migrate dev --name init //first time only
npx prisma generate

npm install --save-dev nodemon

package.json
```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

server.js
```
import http from "http";
import { PrismaClient } from "@prisma/client";
import { parse } from "url";
import { randomUUID } from "crypto";

const prisma = new PrismaClient();
const PORT = 5001;

const sendJSON = (res, status, data) => {
  res.writeHead(status, { "Content-Type": "application/json" });
  res.end(JSON.stringify(data));
};

const parseBody = async (req) =>
  new Promise((resolve, reject) => {
    try {
      let body = "";
      req.on("data", (chunk) => (body += chunk.toString()));
      req.on("end", () => resolve(JSON.parse(body || "{}")));
```

```
    } catch (err) {
      reject(err);
    }
  });

const server = http.createServer(async (req, res) => {
  const { method, url } = req;
  const { pathname } = parse(url, true);

  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE,
OPTIONS");
  res.setHeader("Access-Control-Allow-Headers", "*");
  if (method === "OPTIONS") return res.end();

  try {
    // GET /users
    if (pathname === "/users" && method === "GET") {
      const users = await prisma.user.findMany();
      return sendJSON(res, 200, users);
    }
    // GET /users (with optional search)
       /**
       * Example: /users?search=amir
       */
    if (pathname === "/users" && method === "GET") {
       const { search } = parse(url, true).query;
       const users = await prisma.user.findMany({
         where: search
          ? {
             OR: [
              { name: { contains: search, mode: "insensitive" } },
              { email: { contains: search, mode: "insensitive" } },
             ],
          }
         : {},
       });
      return sendJSON(res, 200, users);
    }
    // POST /users
    if (pathname === "/users" && method === "POST") {
      const body = await parseBody(req);
      const newUser = await prisma.user.create({ data: body });
      return sendJSON(res, 201, newUser);
    }

    // PUT /users/:id
    if (pathname.startsWith("/users/") && method === "PUT") {
      const id = parseInt(pathname.split("/")[2]);
      const body = await parseBody(req);
      const updated = await prisma.user.update({ where: { id }, data: body });
      return sendJSON(res, 200, updated);
    }

    // DELETE /users/:id
    if (pathname.startsWith("/users/") && method === "DELETE") {
      const id = parseInt(pathname.split("/")[2]);
      await prisma.user.delete({ where: { id } });
      return sendJSON(res, 200, { message: "User deleted" });
    }

    // POST /users/login
    if (pathname === "/users/login" && method === "POST") {
      const body = await parseBody(req);
```

```
      const { email, password } = body;

      // ❗ Fixed credentials for now
      const FIXED_EMAIL = "test@example.com";
      const FIXED_PASSWORD = "123456";

      if (email !== FIXED_EMAIL || password !== FIXED_PASSWORD) {
        return sendJSON(res, 401, { message: "Invalid credentials" });
      }

      // Optional: fetch actual user from DB if needed
      const user = await prisma.user.findFirst({
        where: { email: FIXED_EMAIL },
      });

      // If user not found in DB, respond with minimal user
      const responseUser = user || { id: 1, email: FIXED_EMAIL };

      // Generate token
      const token = randomUUID();

      return sendJSON(res, 200, {
        success: true,
        message: "Login successful",
        token,
        user: responseUser,
      });
    }


    sendJSON(res, 404, { message: "Route not found" });
  } catch (err) {
    sendJSON(res, 500, { error: err.message });
  }
});

server.listen(PORT, () => console.log(`🚀 Server running on http://localhost:$
{PORT}`));
```

npx nodemon server.js

IF USING EXPRESSJS

/routes/users.js

```
import { Router } from 'express';
import { prisma } from '../prisma.js';

const router = Router();

router.get('/', async (req, res) => {
  const users = await prisma.user.findMany();
  res.json(users);
```

```
});

/**
 * GET /users
 * Optional search: /users?search=amir
 */
router.get('/', async (req, res) => {
  const { search } = req.query;

  try {
    const users = await prisma.user.findMany({
      where: search
        ? {
            OR: [
              { name: { contains: search, mode: "insensitive" } },
              { email: { contains: search, mode: "insensitive" } },
            ],
          }
        : {}, // No search => return all
    });

    res.json(users);
  } catch (error) {
    res.status(500).json({ message: "Error retrieving users", error: error.message });
  }
});

router.post('/', async (req, res) => {
  const newUser = await prisma.user.create({ data: req.body });
  res.status(201).json(newUser);
});

router.put('/:id', async (req, res) => {
  const updatedUser = await prisma.user.update({
    where: { id: Number(req.params.id) },
    data: req.body,
  });
  res.json(updatedUser);
});

router.delete('/:id', async (req, res) => {
  await prisma.user.delete({ where: { id: Number(req.params.id) } });
  res.json({ message: 'User deleted' });
});

export default router;



// src/server.js
import express from 'express';
import usersRouter from './routes/users.js';

const app = express();
```

```
app.use(express.json());

app.use('/users', usersRouter);

app.listen(5001, () => console.log(`🚀 Server running on http://localhost:5001`));
```