**COMP308 Sample Test**

This exercise involves creating a full-stack application using Express, React Vite, and MongoDB that allows users to manage tasks. We'll be utilizing GraphQL with Apollo Server to interact with our MongoDB database, enhancing our application's flexibility and efficiency.

To complete this exercise, you should use Visual Studio Code (VS Code) as your integrated development environment (IDE).

Follow the steps outlined below:

**Step 1: Develop the GraphQL API**

This API will provide the functionality to add, list, and delete tasks.

* **Create a Contact Model**: Define a mongoose model for tasks, specifying the schema that includes fields such as contactId, name, email, phone, address.
* **Create the GraphQL Schema**: Define your GraphQL schema, which includes type definitions for your Contact model and Query & Mutation types for fetching and modifying contacts.
* **Create Resolvers**: Implement resolvers for your schema's queries and mutations, enabling the actual database interactions for adding, listing, and deleting contacts.

**Step 2: Develop the React Vite Front-end**

Use functional components in React to interact with the GraphQL API.

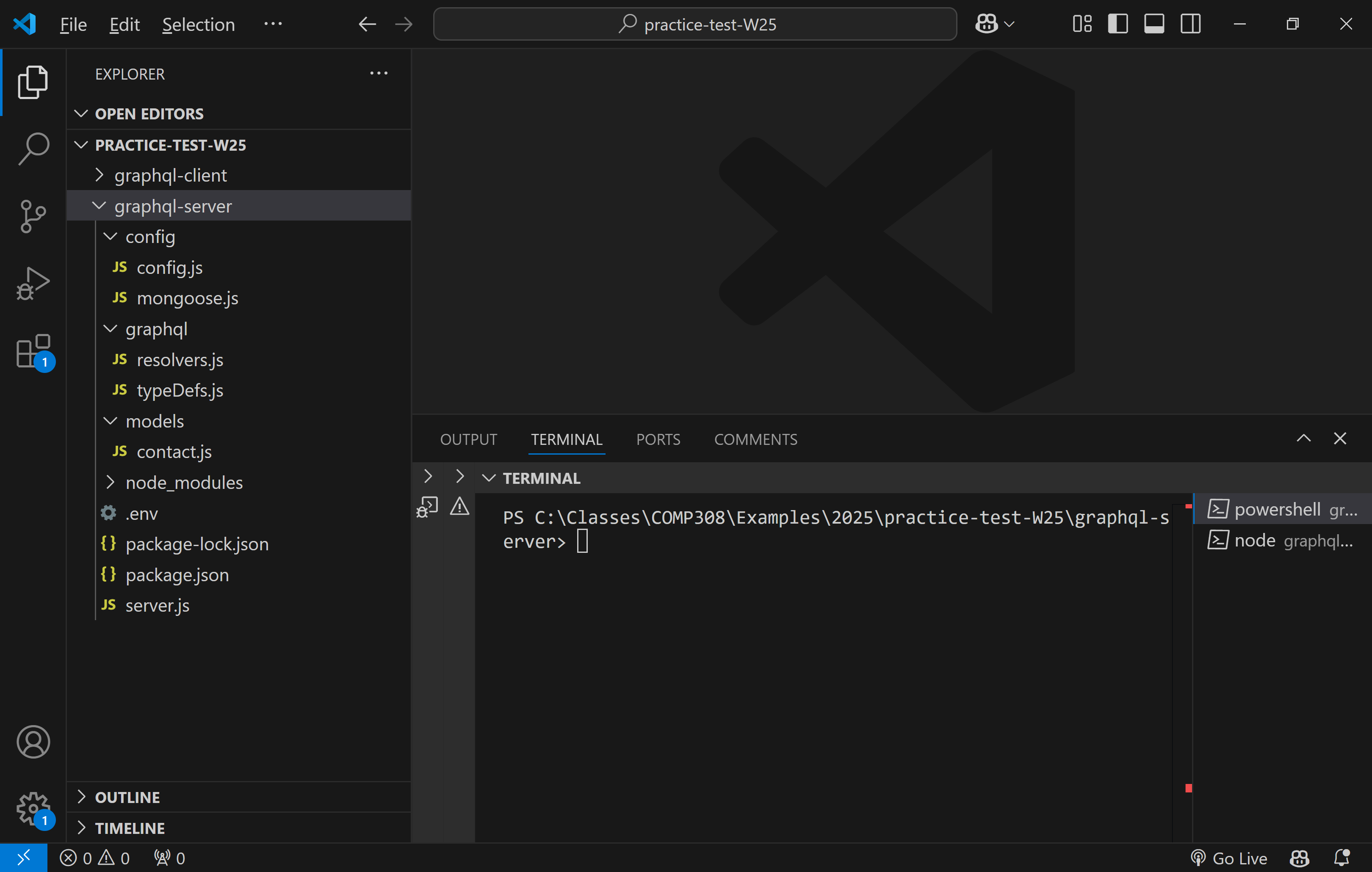
* **CreateContact Component**: Create a functional component to add a new contact. This component should have a form to input contact details.
* **ContactList Component**: Develop a component to list all contacts. Include options to delete or update contacts directly from this list. This component should interact with your GraphQL API to fetch, delete, and update contacts.

**Building the Express GraphQL API:**

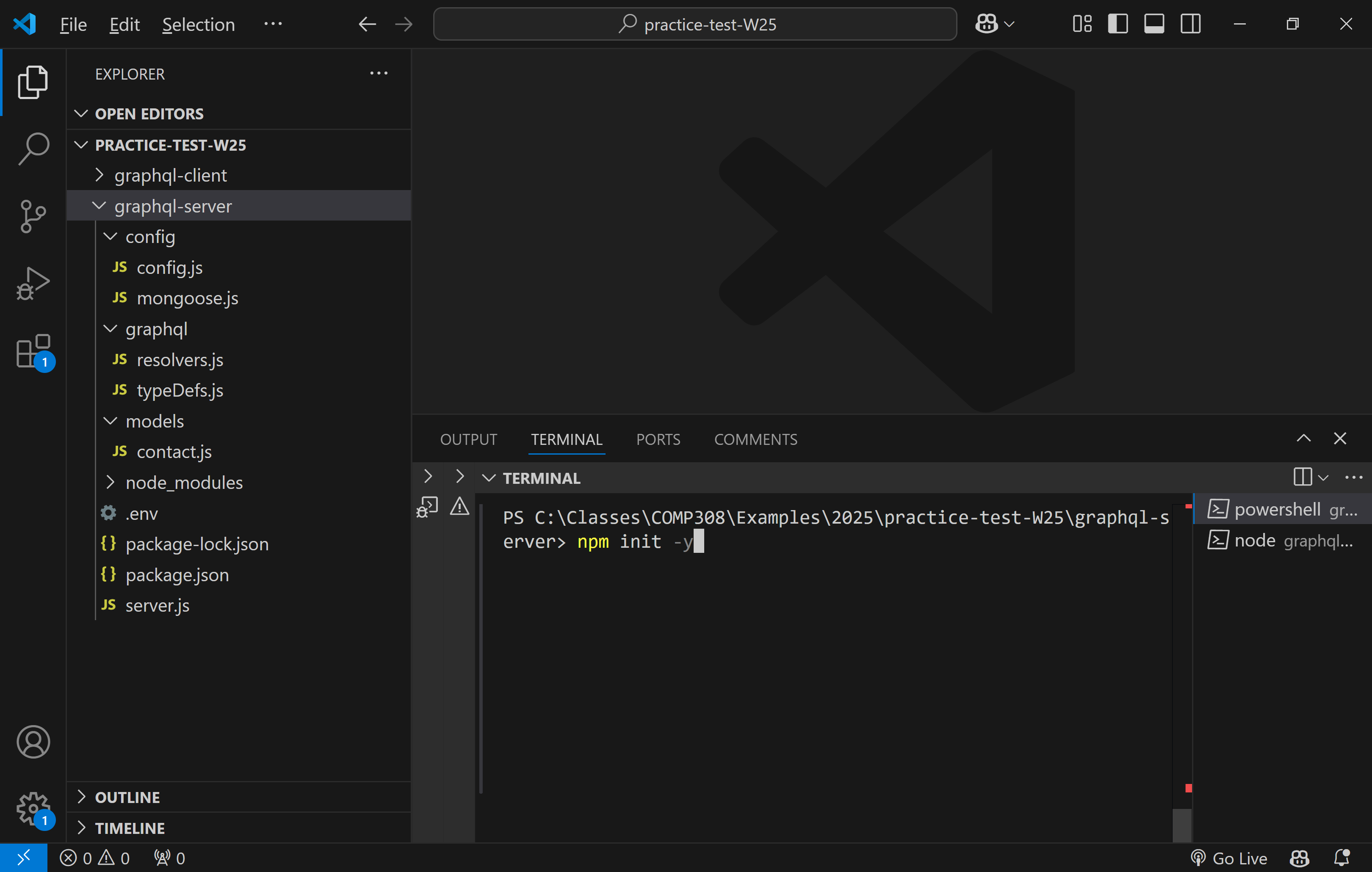
Open VS Code and select New Folder to create a new project folder. Name it SampleTest-F24.

Continue with setting up your project by initializing a new Node.js project, installing necessary dependencies (such as express, graphql, express-graphql, mongoose, etc.), and follow the steps to create your Contact model, GraphQL schema, and resolvers.

Create the following application structure for the graphql server:



In graphql-server folder, create the *package.json* as shown below:



Copy the following code to the newly created package.json file:

{

  "name": "graphql-server",

  "version": "1.0.0",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "node server.js",

    "dev": "nodemon server.js"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "description": "",

  "dependencies": {

    "@apollo/server": "^4.11.2",

    "body-parser": "^1.20.3",

    "cors": "^2.8.5",

    "dotenv": "^16.4.7",

    "express": "^4.21.2",

    "graphql": "^16.9.0",

    "mongoose": "^8.8.4"

  },

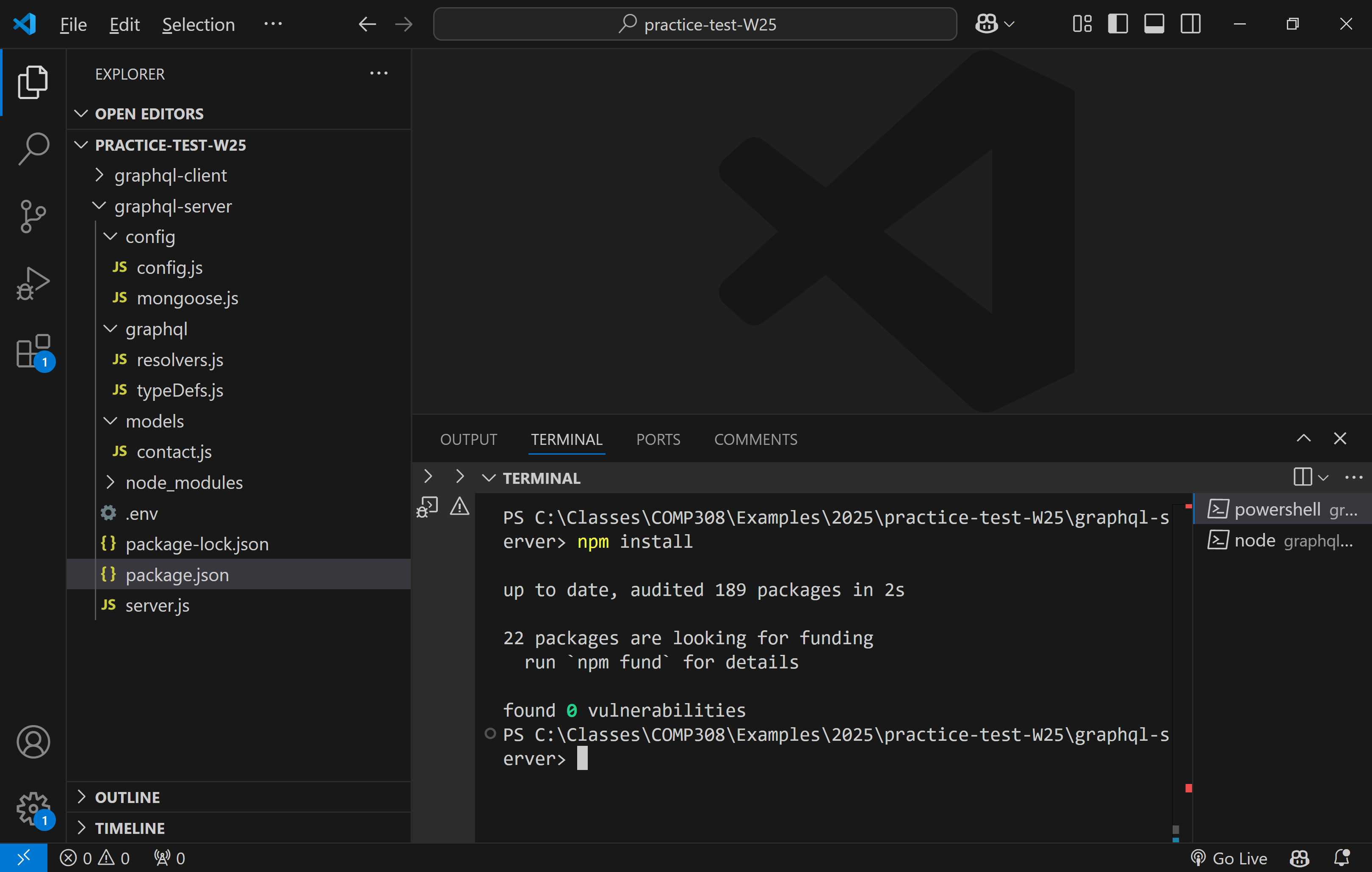
  "devDependencies": {

    "nodemon": "^3.1.7"

  }

}

Install node modules by running **npm install**:



**Creating the Contact model**

In models folder, create the file ***contact.js***. In this file, paste the following lines of code:

// contact.js

// Load the module dependencies

const mongoose = require('mongoose'),

    Schema = mongoose.Schema;

// Define a new 'ContactSchema'

const ContactSchema = new Schema({

    contactId: { type: String, unique: true, required: true },

    name: String,

    email: String,

    phone: String,

    address: String

});

// Create the 'Contact' model out of the 'ContactSchema'

const Contact = mongoose.model('Contact', ContactSchema);

// Export the 'Contact' model

module.exports = Contact;

In **config** folder, create a file named **config.js** and paste the following code:

// config.js is a configuration file that exports an object

// with the configuration options for the application.

// The configuration object contains the database URI and

// the session secret. The database URI is loaded from

// the MONGO\_URI environment variable.

// The configuration object is exported based on the NODE\_ENV

// environment variable, which determines the current environment

// (development, production, or test). The configuration object

// for the current environment is exported from the module.

//

// Import necessary modules

const dotenv = require('dotenv');

// Load environment variables from .env file

dotenv.config();

// Dynamically load the configuration file based on NODE\_ENV

const env = process.env.NODE\_ENV || 'development';

const config = {

  development: {

    db: process.env.MONGO\_URI || 'mongodb://localhost:27017/hands-on-test-db-2025',

  },

};

// Export the configuration for the current environment

module.exports = config[env];

In graphql-server folder, create the file **.env** and paste the following lines:

NODE\_ENV=development

MONGO\_URI=mongodb://localhost:27017/hands-on-test-db-2025

**Creating GraphQL type definitions and resolvers**

Create a file named ***typeDef.js*** inside the ***graphql*** folder. In this newly created file, paste the following lines of code:

|  |
| --- |
| // typeDefs.js is a file that contains the GraphQL  // schema definition language (SDL) that defines the types,  // queries, and mutations that the GraphQL server supports.  // The schema is defined using the GraphQL schema definition  // language (SDL).  const typeDefs = `#graphql    type Contact {      id: ID!      contactId: String!      name: String!      email: String!      phone: String!      address: String!    }    type Query {      contacts: [Contact]      contact(id: ID!): Contact    }    type Mutation {        createContact(        contactId: String!        name: String!        email: String!        phone: String!        address: String!      ): Contact        updateContact(        id: ID!        contactId: String!        name: String!        email: String!        phone: String!        address: String!      ): Contact        deleteContact(id: ID!): Contact        deleteContactByEmail(email: String!): Contact    }  `;  module.exports = typeDefs; |

Create the resolvers in resolvers.js file:

// resolvers.js code for the resolvers of the GraphQL server

const Contact = require('../models/contact');

//

const resolvers = {

  Query: {

    contacts: async () => {

      try {

        const contacts = await Contact.find();

        return contacts.map((contact) => ({

          id: contact.\_id.toString(), // Convert MongoDB `\_id` to GraphQL `id`

          ...contact.toObject(),

        }));

      } catch (error) {

        console.error('Error fetching contacts:', error);

        throw new Error('Failed to fetch contacts');

      }

    },

    contact: async (\_, { id }) => {

      try {

        const contact = await Contact.findById(id);

        if (!contact) {

          throw new Error(`Contact with ID ${id} not found`);

        }

        return {

          id: contact.\_id.toString(), // Convert MongoDB `\_id` to GraphQL `id`

          ...contact.toObject(),

        };

      } catch (error) {

        console.error('Error fetching contact by ID:', error);

        throw new Error('Failed to fetch contact');

      }

    },

  },

  Mutation: {

    createContact: async (\_, args) => {

      try {

        const contact = new Contact(args);

        const newContact = await contact.save();

        return {

          id: newContact.\_id.toString(), // Convert MongoDB `\_id` to GraphQL `id`

          ...newContact.toObject(),

        };

      } catch (error) {

        console.error('Error adding contact:', error);

        throw new Error('Failed to add contact');

      }

    },

    updateContact: async (\_, { id, ...update }) => {

      try {

        const updatedContact = await Contact.findByIdAndUpdate(id, update, { new: true });

        if (!updatedContact) {

          throw new Error(`Contact with ID ${id} not found`);

        }

        return {

          id: updatedContact.\_id.toString(), // Convert MongoDB `\_id` to GraphQL `id`

          ...updatedContact.toObject(),

        };

      } catch (error) {

        console.error('Error updating contact:', error);

        throw new Error('Failed to update contacts');

      }

    },

    deleteContact: async (\_, { id }) => {

      try {

        const deletedContact = await Contact.findByIdAndDelete(id);

        if (!deletedContact) {

          throw new Error(`Contact with ID ${id} not found`);

        }

        return {

          id: deletedContact.\_id.toString(), // Convert MongoDB `\_id` to GraphQL `id`

          ...deletedContact.toObject(),

        };

      } catch (error) {

        console.error('Error deleting contact:', error);

        throw new Error('Failed to delete contact');

      }

    },

    deleteContactByEmail: async (\_, { email }) => {

      try {

        const deletedContact = await Contact.findOneAndDelete({ email });

        if (!deletedContact) {

          throw new Error(`Contact with email ${email} not found`);

        }

        return {

          id: deletedContact.\_id.toString(),

          ...deletedContact.toObject(),

        };

      } catch (error) {

        console.error('Error deleting contact:', error);

        throw new Error('Failed to delete contact');

      }

    },

  },

};

//

module.exports = resolvers;

**Creating server.js file**

Create ***server.js*** file in root folder of Express app and paste the following code:

// server.js is the entry point for the GraphQL server.

// It connects to MongoDB, creates an Apollo Server, and starts

// the server on port 4000.

require('dotenv').config(); // Load environment variables

const { ApolloServer } = require('@apollo/server');

const { startStandaloneServer } = require('@apollo/server/standalone');

const configureMongoose = require('./config/mongoose');

const typeDefs = require('./graphql/typeDefs');

const resolvers = require('./graphql/resolvers');

// Initialize the application

const startServer = async () => {

  // Step 1: Connect to MongoDB

  await configureMongoose();

  // Step 2: Create Apollo Server

  const server = new ApolloServer({

    typeDefs,

    resolvers,

  });

  // Step 3: Start Apollo Server

  const { url } = await startStandaloneServer(server, {

    listen: { port: 4000 },

  });

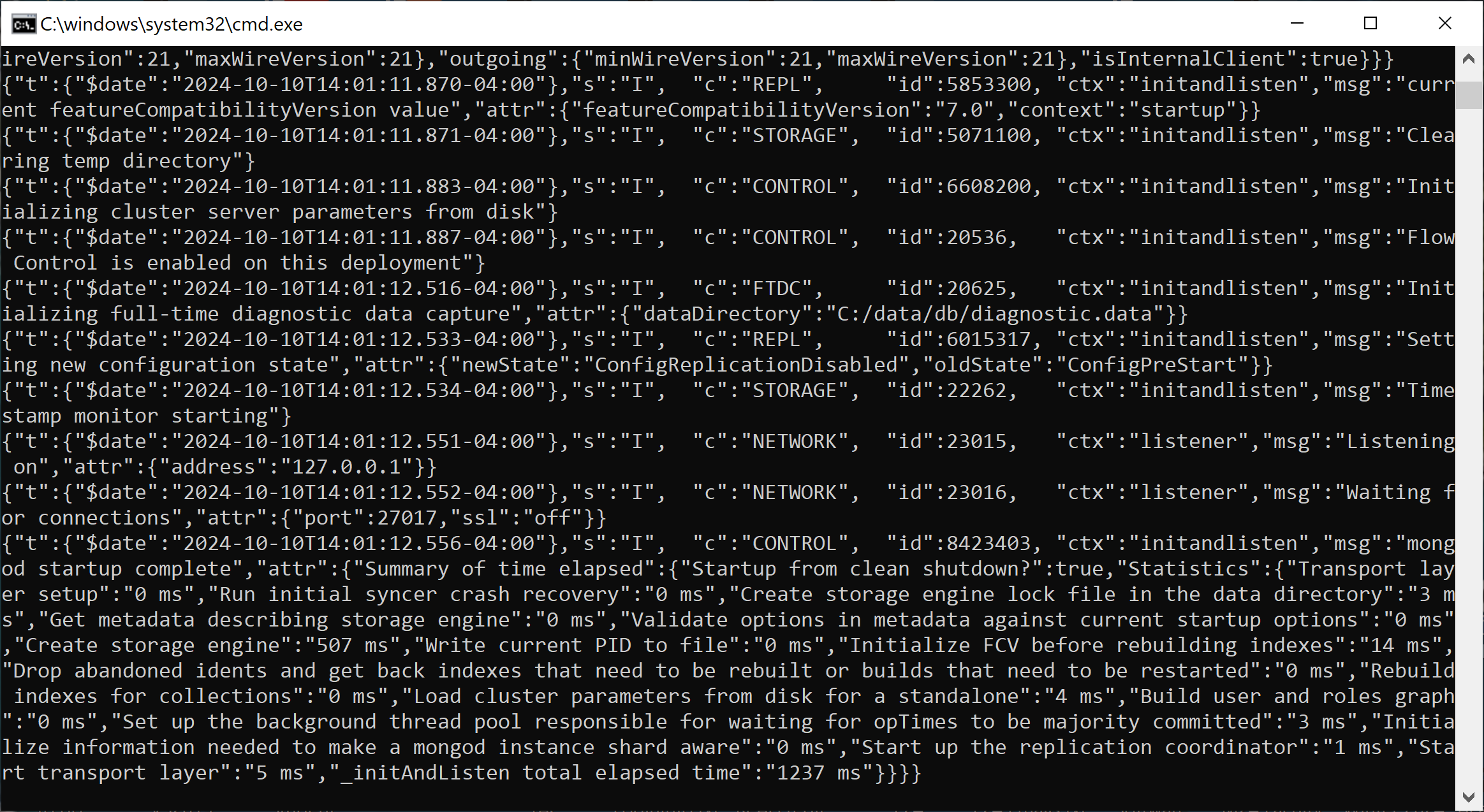
  console.log(`🚀 GraphQL server ready at ${url}`);

};

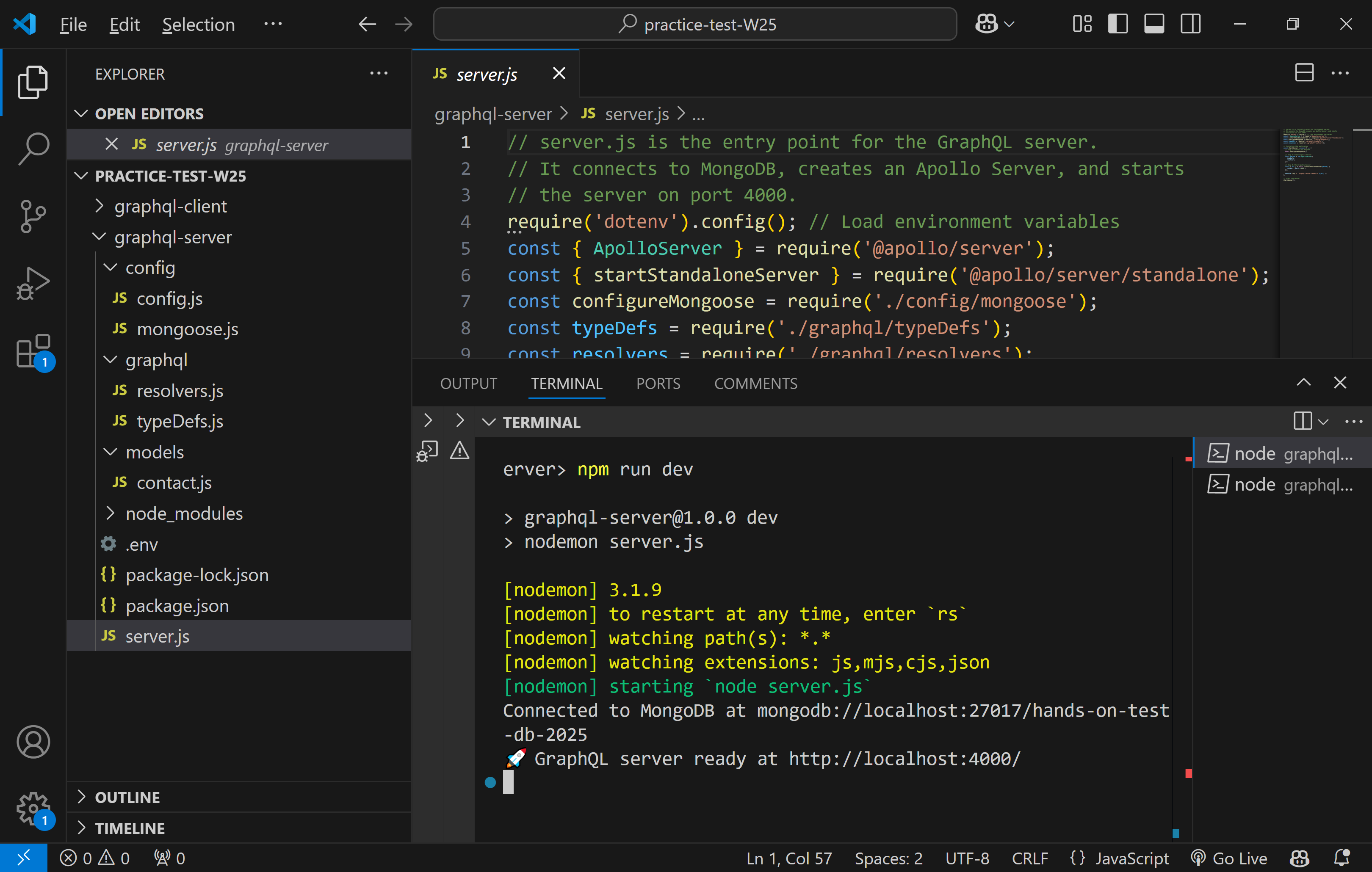
// Start the server

startServer();

Run ***mongod.exe*** to start the MongoDb server:



You can now run the application, **npm run dev**, to see if the connection is accepted:

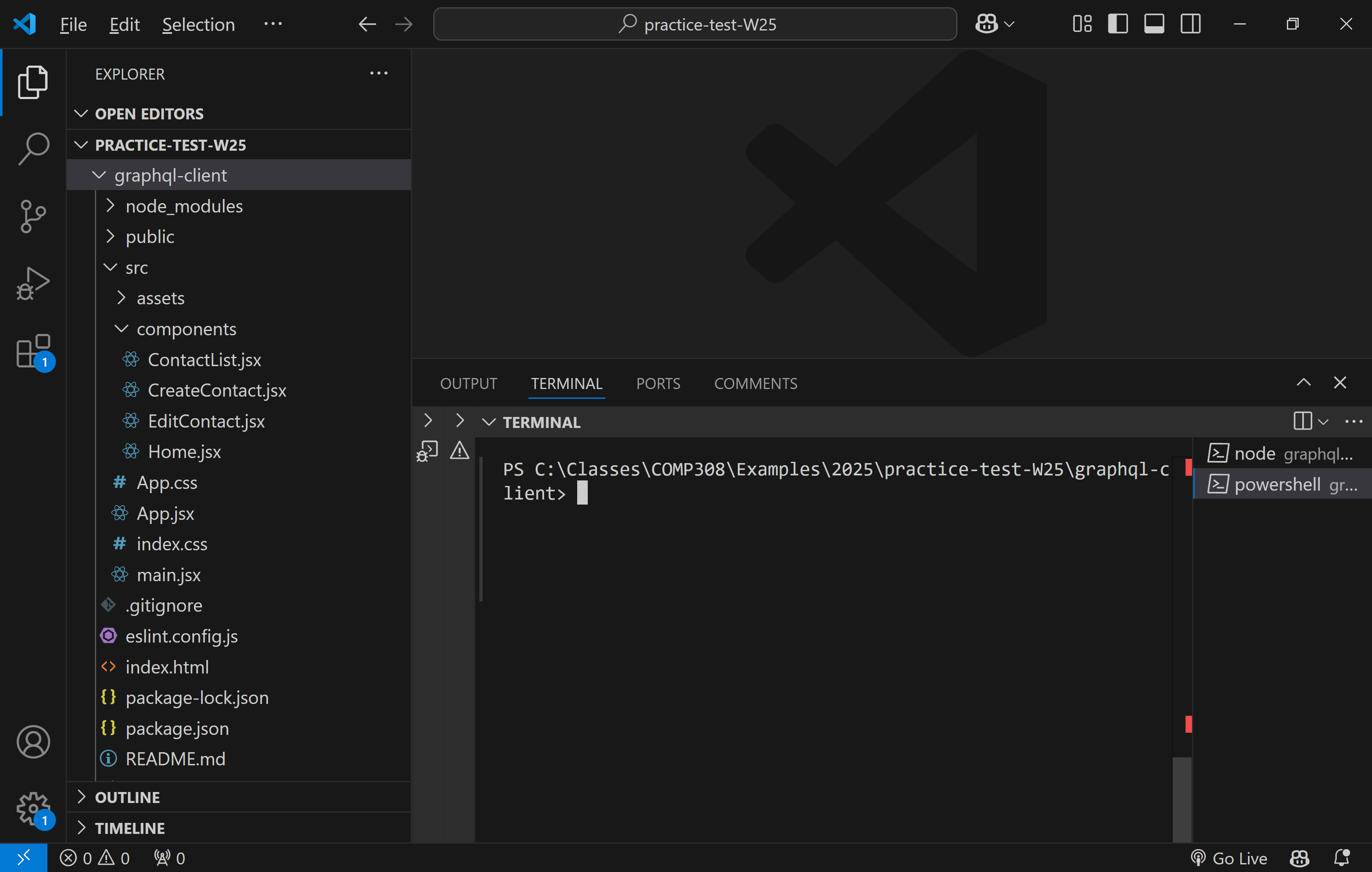


**Creating the React Front-end**

Open a new terminal window in the root folder of Express app, run the following command to create the React Vite client app:

**npm create vite@latest graphql-client**

In terminal prompt, run: **cd graphql-client**, then **npm install** to install Vite React modules.



Run: **npm install react-bootstrap bootstrap**. Now, install react-router-dom: **npm install react-router-dom**.

Here is a view of the **package.json** file after installation is complete:

{

  "name": "graphql-client",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint .",

    "preview": "vite preview"

  },

  "dependencies": {

    "@apollo/client": "^3.11.8",

    "bootstrap": "^5.3.3",

    "react": "^18.3.1",

    "react-bootstrap": "^2.10.5",

    "react-dom": "^18.3.1",

    "react-router-dom": "^6.26.2"

  },

  "devDependencies": {

    "@eslint/js": "^9.11.1",

    "@types/react": "^18.3.10",

    "@types/react-dom": "^18.3.0",

    "@vitejs/plugin-react": "^4.3.2",

    "eslint": "^9.11.1",

    "eslint-plugin-react": "^7.37.0",

    "eslint-plugin-react-hooks": "^5.1.0-rc.0",

    "eslint-plugin-react-refresh": "^0.4.12",

    "globals": "^15.9.0",

    "vite": "^5.4.8"

  }

}

**Creating the UI for the React Contact Management App**

We'll be developing the user interface for our task management app using React functional components. The components we aim to create include:

**CreateContact**: This component is responsible for creating new contacts. It provides a form where users can enter contact details and submit them.

**ContactList**: This component lists all the contacts and offers functionality to edit a contact directly from the list.

**EditContact**: This component is responsible for editing an existing contact.

**Home**: This component serves as the landing or home page of our application.

**App**: This is the main component that orchestrates routing and includes navigation bars to move between different parts of the application.

Inside the **src** folder of your React project, create a new folder named **components**. This will organize your component files neatly.

Navigate to the **components** folder you just created.

Create a new file named **CreateContact.jsx**. This file will contain the code for our **CreateContact** component.

Open **CreateContact**.jsx and add the following code:

import React, { useState } from 'react';

import { gql, useMutation } from "@apollo/client";

import { useNavigate } from 'react-router-dom';

import { Container, Form, Button, Row, Col } from 'react-bootstrap';

// Mutation to add a contact

const ADD\_CONTACT = gql`

  mutation AddContact($contactId: String!, $name: String!, $email: String!, $phone: String, $address: String) {

    createContact(contactId: $contactId, name: $name, email: $email, phone: $phone, address: $address) {

      id

      contactId

      name

    }

  }

`;

function CreateContact() {

    let navigate = useNavigate();

    const [addContact] = useMutation(ADD\_CONTACT);

    const [contactId, setContactId] = useState('');

    const [name, setName] = useState('');

    const [email, setEmail] = useState('');

    const [phone, setPhone] = useState('');

    const [address, setAddress] = useState('');

    const handleSubmit = (e) => {

        e.preventDefault();

        addContact({ variables: { contactId, name, email, phone, address } });

        setContactId('');

        setName('');

        setEmail('');

        setPhone('');

        setAddress('');

        navigate('/contactlist');

    };

    return (

        <Container>

            <h2>Add Contact</h2>

            <Form onSubmit={handleSubmit}>

                <Form.Group as={Row} className="mb-3">

                    <Form.Label column sm={2}>Contact ID</Form.Label>

                    <Col sm={10}>

                        <Form.Control

                            type="text"

                            placeholder="Contact ID"

                            value={contactId}

                            onChange={(e) => setContactId(e.target.value)}

                        />

                    </Col>

                </Form.Group>

                <Form.Group as={Row} className="mb-3">

                    <Form.Label column sm={2}>Name</Form.Label>

                    <Col sm={10}>

                        <Form.Control

                            type="text"

                            placeholder="Name"

                            value={name}

                            onChange={(e) => setName(e.target.value)}

                        />

                    </Col>

                </Form.Group>

                <Form.Group as={Row} className="mb-3">

                    <Form.Label column sm={2}>Email</Form.Label>

                    <Col sm={10}>

                        <Form.Control

                            type="email"

                            placeholder="Email"

                            value={email}

                            onChange={(e) => setEmail(e.target.value)}

                        />

                    </Col>

                </Form.Group>

                <Form.Group as={Row} className="mb-3">

                    <Form.Label column sm={2}>Phone</Form.Label>

                    <Col sm={10}>

                        <Form.Control

                            type="text"

                            placeholder="Phone"

                            value={phone}

                            onChange={(e) => setPhone(e.target.value)}

                        />

                    </Col>

                </Form.Group>

                <Form.Group as={Row} className="mb-3">

                    <Form.Label column sm={2}>Address</Form.Label>

                    <Col sm={10}>

                        <Form.Control

                            type="text"

                            placeholder="Address"

                            value={address}

                            onChange={(e) => setAddress(e.target.value)}

                        />

                    </Col>

                </Form.Group>

                <Button variant="primary" type="submit">Add Contact</Button>

            </Form>

        </Container>

    );

}

export default CreateContact;

Create the ***login.css*** file in **components** folder and paste the following code:

@media all and (min-width: 480px) {

    .login {

      padding: 10px 0;

    }

    .login form {

      margin: 0 auto;

      max-width: 600px;

    }

  }

Create an ***images*** folder in ***src*** folder. Put the logo image there (sample\_test.png).

Create **Contact*List.jsx*** file in folder ***components***. Paste the following code:

import React from 'react';

import Table from 'react-bootstrap/Table';

import Spinner from 'react-bootstrap/Spinner';

import { Link } from 'react-router-dom';

import { gql, useQuery } from '@apollo/client';

// import Button from 'react-bootstrap/Button';

import Button from 'react-bootstrap/Button';

// Query to fetch contacts

const GET\_CONTACTS = gql`

  query GetContacts {

    contacts {

      id

      contactId

      name

      email

      phone

      address

    }

  }

`;

function ContactList() {

    const { loading, error, data, refetch } = useQuery(GET\_CONTACTS);

    if (loading) return <Spinner animation="border" />;

    if (error) return <p>Error :(</p>;

    return (

        <div>

            <h2>Contact List</h2>

            <Table striped bordered hover>

                <thead>

                    <tr>

                        <th>Contact ID</th>

                        <th>Name</th>

                        <th>Email</th>

                        <th>Phone</th>

                        <th>Address</th>

                        <th>Action</th>

                    </tr>

                </thead>

                <tbody>

                    {data.contacts.map((contact) => (

                        <tr key={contact.id}>

                            <td>{contact.contactId}</td>

                            <td>{contact.name}</td>

                            <td>{contact.email}</td>

                            <td>{contact.phone}</td>

                            <td>{contact.address}</td>

                            <td>

                                <Link to={`/editcontact/${contact.id}`}>Edit</Link>

                            </td>

                        </tr>

                    ))}

                </tbody>

            </Table>

            <Button variant="secondary" onClick={() => refetch()}>Refetch</Button>

        </div>

    );

}

export default ContactList;

Create ***Home.jsx*** file in ***components*** folder and paste the following code:

import React from 'react';

import  pic from "./images/sample\_test.png";

function Home(props)

{

    return (

        <div>

            <img src={pic} />

            <h2> Express - React with CRUD Operations</h2>

            <p>React front-end calls Express REST API to add,

            list, update, or delete a task.</p>

        </div>

    );

}

//

export default Home;

Create **EditContact.jsx** file in components folder:

|  |
| --- |
| import React, { useState, useEffect } from 'react';  import { useParams, useNavigate } from 'react-router-dom';  import { gql, useQuery, useMutation } from '@apollo/client';  import Form from 'react-bootstrap/Form';  import Button from 'react-bootstrap/Button';  const GET\_CONTACT = gql`    query GetContact($id: ID!) {      contact(id: $id) {        contactId        name        email        phone        address      }    }  `;  const UPDATE\_CONTACT = gql`    mutation UpdateContact($id: ID!, $contactId: String!, $name: String!, $email: String!, $phone: String, $address: String) {      updateContact(id: $id, contactId: $contactId, name: $name, email: $email, phone: $phone, address: $address) {        id        contactId        name        email        phone        address      }    }  `;  function EditContact() {      const { id } = useParams();      let navigate = useNavigate();      const { loading, error, data } = useQuery(GET\_CONTACT, { variables: { id } });      const [updateContact] = useMutation(UPDATE\_CONTACT);      const [contact, setContact] = useState({ contactId: '', name: '', email: '', phone: '', address: '' });      useEffect(() => {          if (data) setContact(data.contact);      }, [data]);      const handleInputChange = (e) => {          const { name, value } = e.target;          setContact({ ...contact, [name]: value });      };      const handleSubmit = (e) => {          e.preventDefault();          updateContact({ variables: { id: String(id), ...contact } }); // Ensure `id` is a string          navigate('/contactlist');      };      if (loading) return <p>Loading...</p>;      if (error) return <p>Error :(</p>;      return (          <Form onSubmit={handleSubmit}>              <Form.Group controlId="formContactId">                  <Form.Label>Contact ID</Form.Label>                  <Form.Control                      type="text"                      name="contactId"                      value={contact.contactId}                      onChange={handleInputChange}                      placeholder="Enter contact ID"                  />              </Form.Group>              <Form.Group controlId="formName">                  <Form.Label>Name</Form.Label>                  <Form.Control                      type="text"                      name="name"                      value={contact.name}                      onChange={handleInputChange}                      placeholder="Enter name"                  />              </Form.Group>              <Form.Group controlId="formEmail">                  <Form.Label>Email</Form.Label>                  <Form.Control                      type="email"                      name="email"                      value={contact.email}                      onChange={handleInputChange}                      placeholder="Enter email"                  />              </Form.Group>              <Form.Group controlId="formPhone">                  <Form.Label>Phone</Form.Label>                  <Form.Control                      type="text"                      name="phone"                      value={contact.phone}                      onChange={handleInputChange}                      placeholder="Enter phone"                  />              </Form.Group>              <Form.Group controlId="formAddress">                  <Form.Label>Address</Form.Label>                  <Form.Control                      type="text"                      name="address"                      value={contact.address}                      onChange={handleInputChange}                      placeholder="Enter address"                  />              </Form.Group>              <Button variant="primary" type="submit">                  Update Contact              </Button>          </Form>      );  }  export default EditContact; |

Copy the following code to App.jsx file:

import './App.css';

import React from 'react';

import { BrowserRouter as Router, Route, Link, Routes } from "react-router-dom";

import 'bootstrap/dist/css/bootstrap.min.css';

import Navbar from 'react-bootstrap/Navbar';

import Nav from 'react-bootstrap/Nav';

import Container from 'react-bootstrap/Container';

import ContactList from './components/ContactList';

import CreateContact from './components/CreateContact';

import EditContact from './components/EditContact';

import Home from './components/Home';

function App() {

  return (

    <Router>

      <Navbar bg="primary" variant="dark" expand="lg">

        <Container>

          <Navbar.Brand href="home">React Client For GraphQL API</Navbar.Brand>

          <Navbar.Toggle aria-controls="basic-navbar-nav" />

          <Navbar.Collapse id="basic-navbar-nav">

            <Nav className="mr-auto">

              <Nav.Link as={Link} to="/home">Home</Nav.Link>

              <Nav.Link as={Link} to="/createcontact">Create Contact</Nav.Link>

              <Nav.Link as={Link} to="/contactlist">Contact List</Nav.Link>

            </Nav>

          </Navbar.Collapse>

        </Container>

      </Navbar>

      <div>

        <Routes>

          <Route index element={<Home />} />

          <Route path="home" element={<Home />} />

          <Route path="contactlist" element={<ContactList />} />

          <Route path="createcontact" element={<CreateContact />} />

          <Route path="editcontact/:id" element={<EditContact />} />

        </Routes>

      </div>

    </Router>

  );

}

export default App;

Update also the **main.jsx** file:

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App.jsx';

import './index.css';

import { ApolloClient, InMemoryCache } from '@apollo/client';

import { ApolloProvider } from '@apollo/client';

const client = new ApolloClient({

  uri: 'http://localhost:4000/graphql',

  cache: new InMemoryCache()

});

ReactDOM.createRoot(document.getElementById('root')).render(

  <React.StrictMode>

    <ApolloProvider client={client}>

      <App />

    </ApolloProvider>

  </React.StrictMode>,

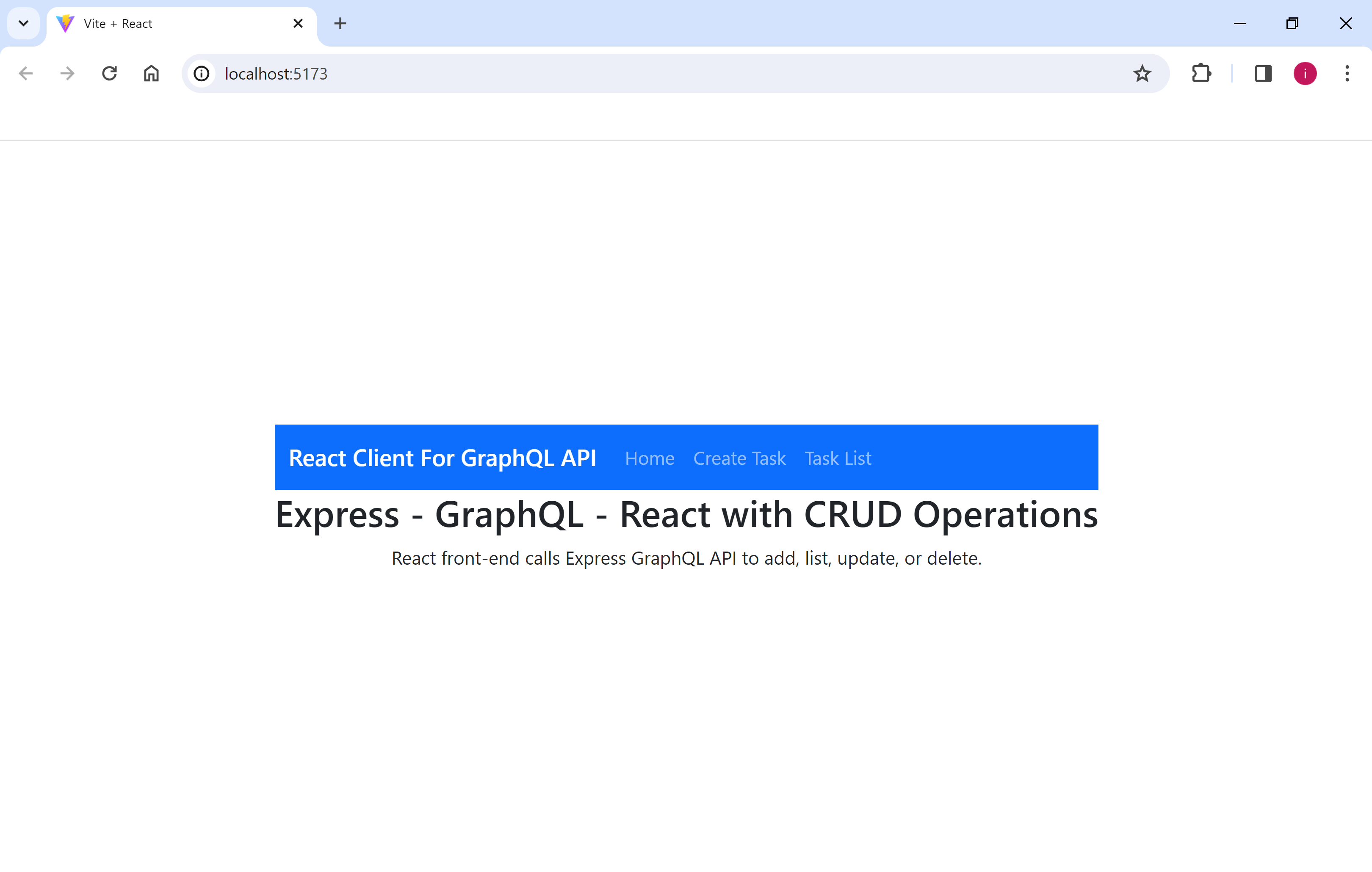
);

**Testing the Full-stack app**

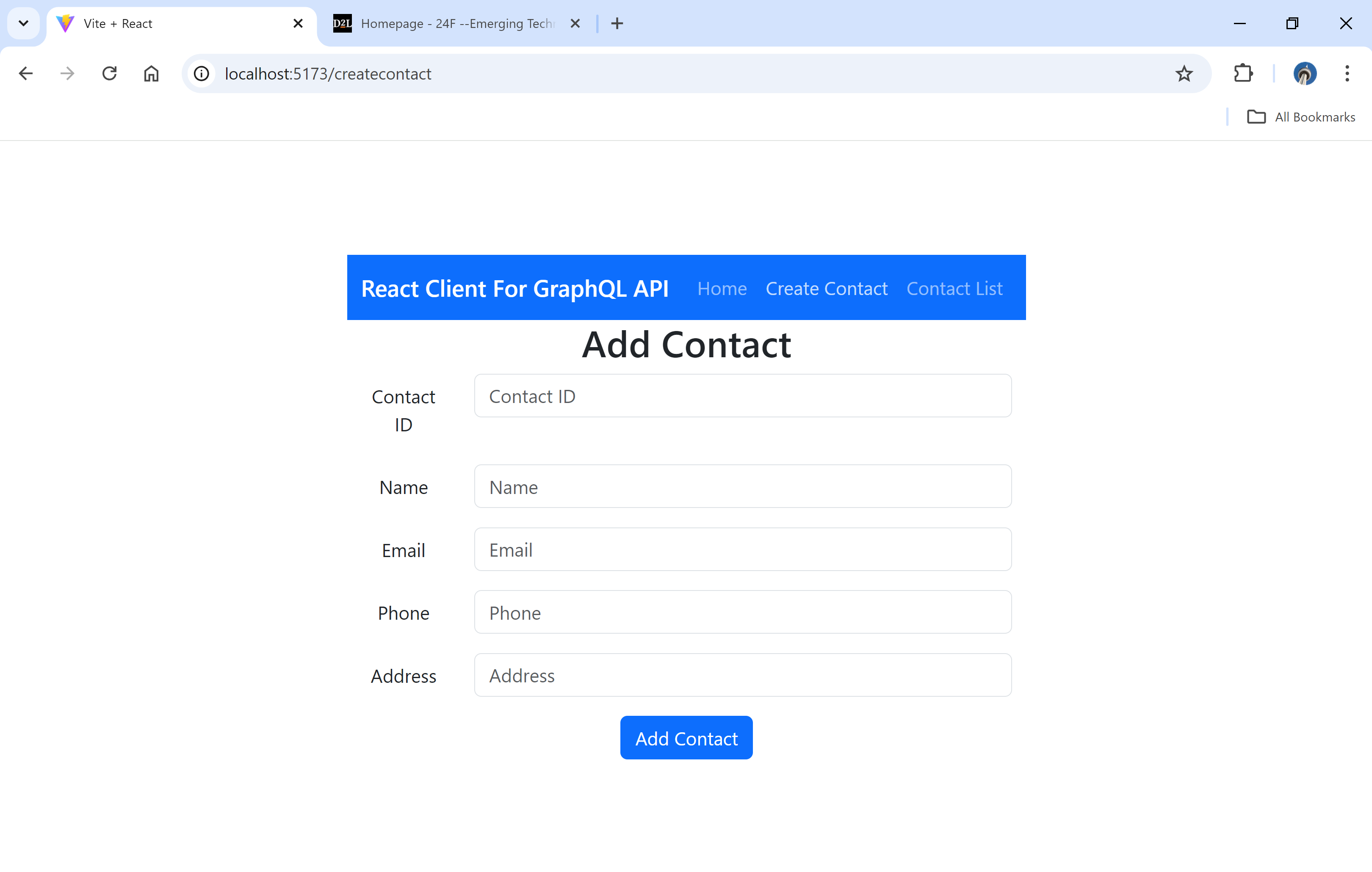
The following commands have to be executed if you ended the servers.

1. Run **mongod.exe** to run the MongoDB server
2. Run Express server in a new terminal window, pointing to root folder: **npm run dev**
3. Run React app in a new terminal window pointing to **graphql-client** folder: **npm run dev**

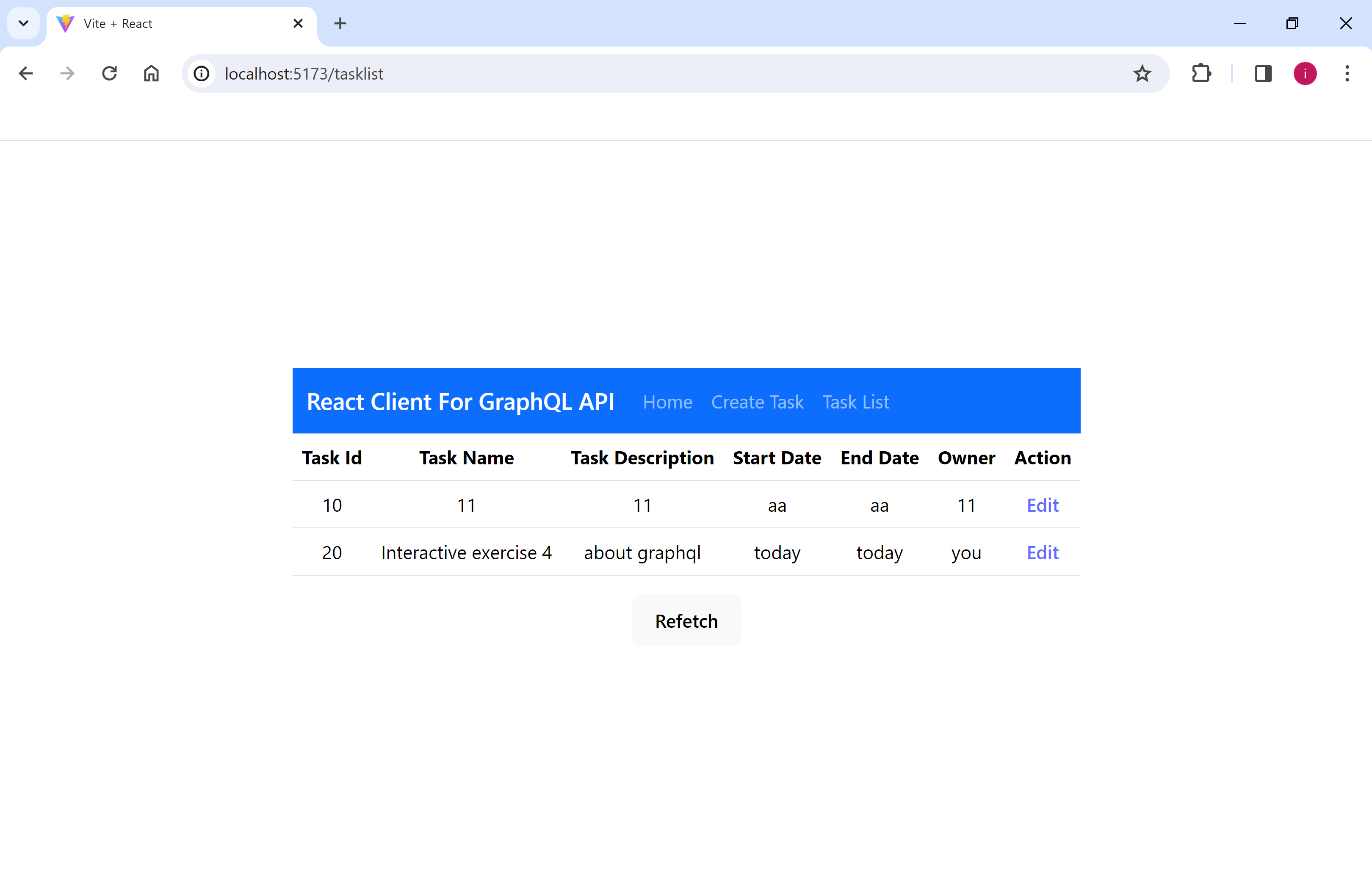
Control-click on the link. You will see the following view:



Select Create Contact to add a contact:



Click Save. Add another task. Here is a view of List component:



Nice, eh!

Now, test Edit operation, and add delete link to implement the Delete operation.