

## **Labfinder Assessment**

**Author: Varun Murthy Mokarala**

**Documentation:**

### **LabFinder**

A web application built with Next.js, GraphQL, and React to allow users to find lab test providers. This document provides setup instructions, technical choices, a feature list, and API documentation.

#### **Table of Contents:**

1. Setup Instructions
2. Technical Choices
3. Implemented Features
4. Known Limitations
5. Running the Application Locally
6. Future Enhancements
7. API Documentation

### **1. Setup Instructions:**

To set up the project locally, follow these steps:

#### **1.1 Clone the repository:**

```
git clone https://github.com/varunmurthymokarala/labfinder.git
```

#### **1.2 Navigate to the project directory:**

```
cd labfinder
```

#### **1.3 Install dependencies:**

```
npm install
```

#### **1.4 Run the application in development mode:**

```
npm run dev
```

#### **1.5 Build for production:**

```
npm run build
```

#### **1.6 Start the production server:**

```
npm start
```

The application will be running on <http://localhost:3000>.

## 2. Technical choices:

### 2.1 API Style: GraphQL with Apollo Server

GraphQL was chosen to enable flexible querying of data. The GraphQL schema defines the structure of the API and allows clients to request only the data they need. Apollo Server was used for its robust GraphQL implementation and ease of integration with Next.js.

### 2.2 Database Solution: JSON-based file storage (for demonstration purposes)

For simplicity, a JSON file (providers.json) is used as the data source instead of a traditional database. If the project scales, a database like MongoDB or PostgreSQL would be a better fit to handle relational data and complex queries.

### 2.3 Framework: Next.js

Next.js was chosen for its server-side rendering capabilities, static generation, and ability to integrate seamlessly with React and GraphQL.

## 3. Implemented Features:

**Search Lab Providers:** Users can search for lab test providers and view all the providers list.

**View Lab Details:** Each provider has detailed information such as doctor's name, available times, and speciality tests.

**Book Appointments:** Users/anonymous users can book lab test appointments.

**GraphQL API:** Provides an interface to query providers and their details.

## 4. Known Limitations

**JSON as a Database:** Using a JSON file (providers.json) limits the scalability and performance of the application. It should be replaced with a proper database.

**Minimal Error Handling:** Error handling and validation are basic. Adding more robust validation is recommended.

**No Authentication:** The application currently does not have any user authentication or authorization implemented.

## 5. Running the Application Locally:

**Start the development server:**

```
npm run dev
```

**Access the application:** Open <http://localhost:3000> in your browser.

## 6. Future Enhancements:

If I had more time, I would add the following features:

**User Authentication:** Implement JWT-based authentication and authorization.

**Database Integration:** Replace the JSON file with a proper database like MongoDB or PostgreSQL.

**User Profile Management:** Allow users to manage their profiles and view their appointment history.

**Testing:** Add comprehensive unit and integration tests using Jest and Cypress.

**Improved UI/UX:** Make the UI more interactive and responsive.

**Search Filters:** Allow users to filter providers based on price, rating, and availability.

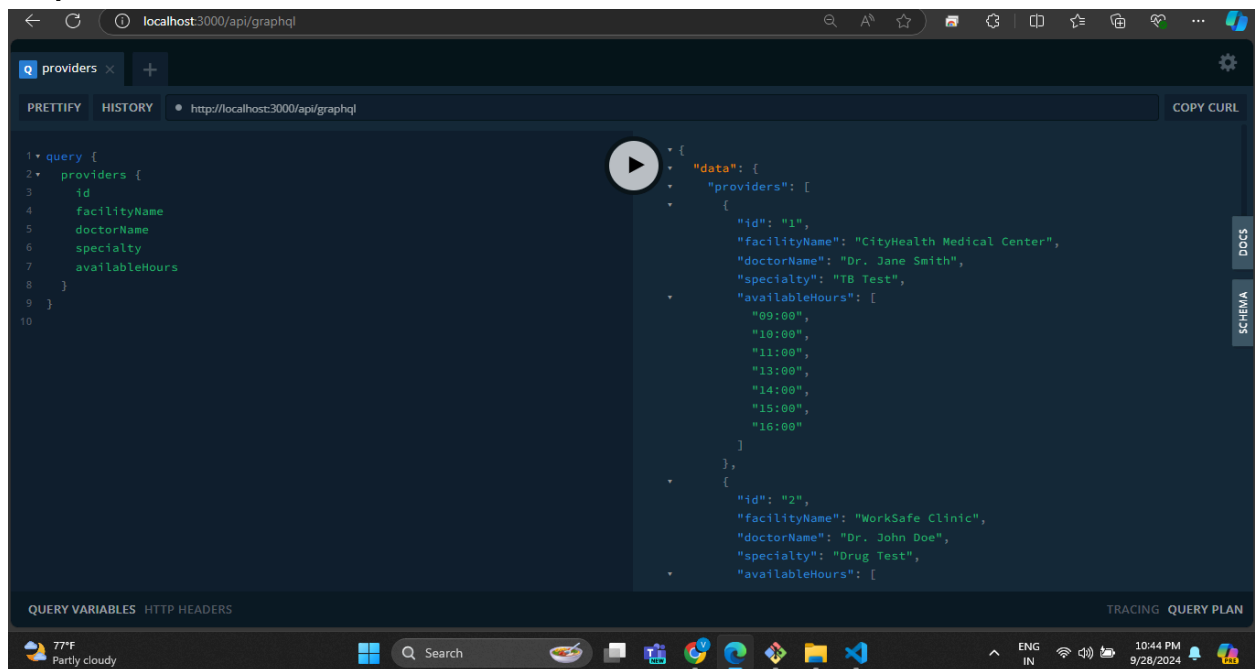
## 7.API Documentation

The following are the GraphQL queries and mutations for the API:

### 7.1 Fetch All Providers:

```
query {  
  providers {  
    id  
    facilityName  
    doctorName  
    specialty  
    availableHours  
  }  
}
```

**Output:**



## 7.2 Fetch a Single Provider by ID:

```
query {  
  provider(id: "3") {  
    id  
    facilityName  
    doctorName  
    specialty  
    availableHours  
  }  
}
```

Output:

The screenshot shows a web browser window with a GraphQL Playground interface. The URL bar shows `localhost3000/api/graphql`. The interface has a tab labeled "provider" and a "COPY CURL" button. The query editor on the left contains the following query:

```
1 query {  
2   provider(id: "3") {  
3     id  
4     facilityName  
5     doctorName  
6     specialty  
7     availableHours  
8   }  
9 }  
10
```

The response editor on the right shows the JSON output:

```
{  
  "data": {  
    "provider": {  
      "id": "3",  
      "facilityName": "HeartWell Institute",  
      "doctorName": "Dr. Emily Brown",  
      "specialty": "Heart Screening",  
      "availableHours": [  
        "09:00",  
        "10:00",  
        "11:00",  
        "13:00",  
        "14:00",  
        "15:00"  
      ]  
    }  
  }  
}
```

At the bottom of the interface, there are tabs for "QUERY VARIABLES", "HTTP HEADERS", "TRACING", and "QUERY PLAN". The Windows taskbar at the very bottom shows the date and time as 10:46 PM on 9/28/2024.

### 7.3 Fetch All Appointments:

```
query {  
  appointments {  
    id  
    providerId  
    dateTime  
    userEmail  
    reservationCode  
  }  
}
```

Output:

The screenshot shows a web browser window with a GraphQL Playground interface. The URL bar shows `localhost:3000/api/graphql`. The query editor on the left contains the following query:

```
1 query {  
2   appointments {  
3     id  
4     providerId  
5     dateTime  
6     userEmail  
7     reservationCode  
8   }  
9 }  
10
```

The response editor on the right shows the following JSON output:

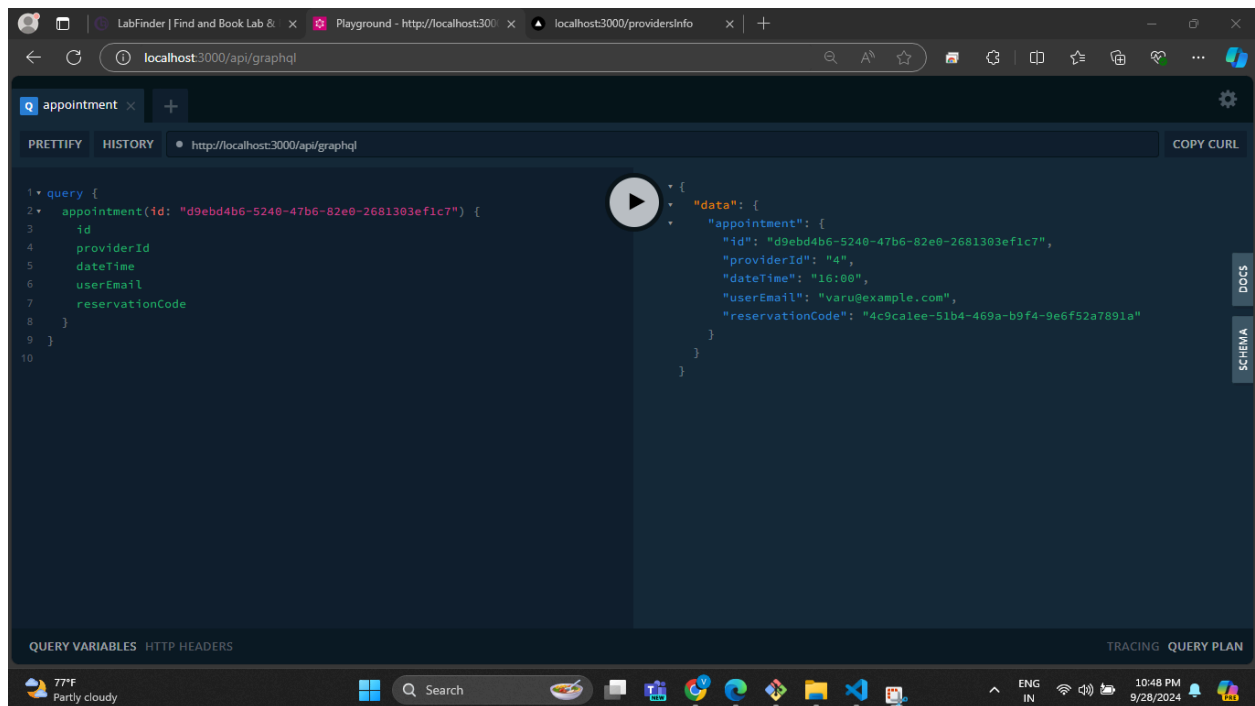
```
{  
  "data": {  
    "appointments": [  
      {  
        "id": "d9ebd4b6-5240-47b6-82e0-2681303ef1c7",  
        "providerId": "4",  
        "dateTime": "16:00",  
        "userEmail": "Varu@example.com",  
        "reservationCode": "4c9ca1ee-51b4-469a-b9f4-9e6f52a7891a"  
      },  
      {  
        "id": "debb864f-e287-4ef8-a3e5-771c186adae4",  
        "providerId": "4",  
        "dateTime": "12:00",  
        "userEmail": "example@test.com",  
        "reservationCode": "25cf9161-d63c-4d97-8684-bf44015a4629"  
      },  
      {  
        "id": "b203425c-d6f9-4954-bf8d-ea7411a11b17",  
        "providerId": "4",  
        "dateTime": "16:00",  
        "userEmail": "var@gmail.com",  
        "reservationCode": "6a77bceb-4ad3-4547-8d1d-0d431ba0d64c"  
      }  
    ]  
  }  
}
```

The interface includes tabs for 'PRETTIFY', 'HISTORY', 'QUERY VARIABLES', 'HTTP HEADERS', 'TRACING', and 'QUERY PLAN'. The status bar at the bottom shows the system clock as 10:46 PM on 9/28/2024.

## 7.4 Fetch an Appointment by ID:

```
query {  
  appointment(id: "d9ebd4b6-5240-47b6-82e0-2681303ef1c7") {  
    id  
    providerId  
    dateTime  
    userEmail  
    reservationCode  
  }  
}
```

### Output:



The screenshot shows a web browser window with a GraphQL Playground interface. The URL bar shows `localhost3000/api/graphql`. The interface has a dark theme. On the left, there's a query editor with the following query:

```
1 query {  
2   appointment(id: "d9ebd4b6-5240-47b6-82e0-2681303ef1c7") {  
3     id  
4     providerId  
5     dateTime  
6     userEmail  
7     reservationCode  
8   }  
9 }  
10
```

On the right, there's a JSON response viewer showing the result of the query:

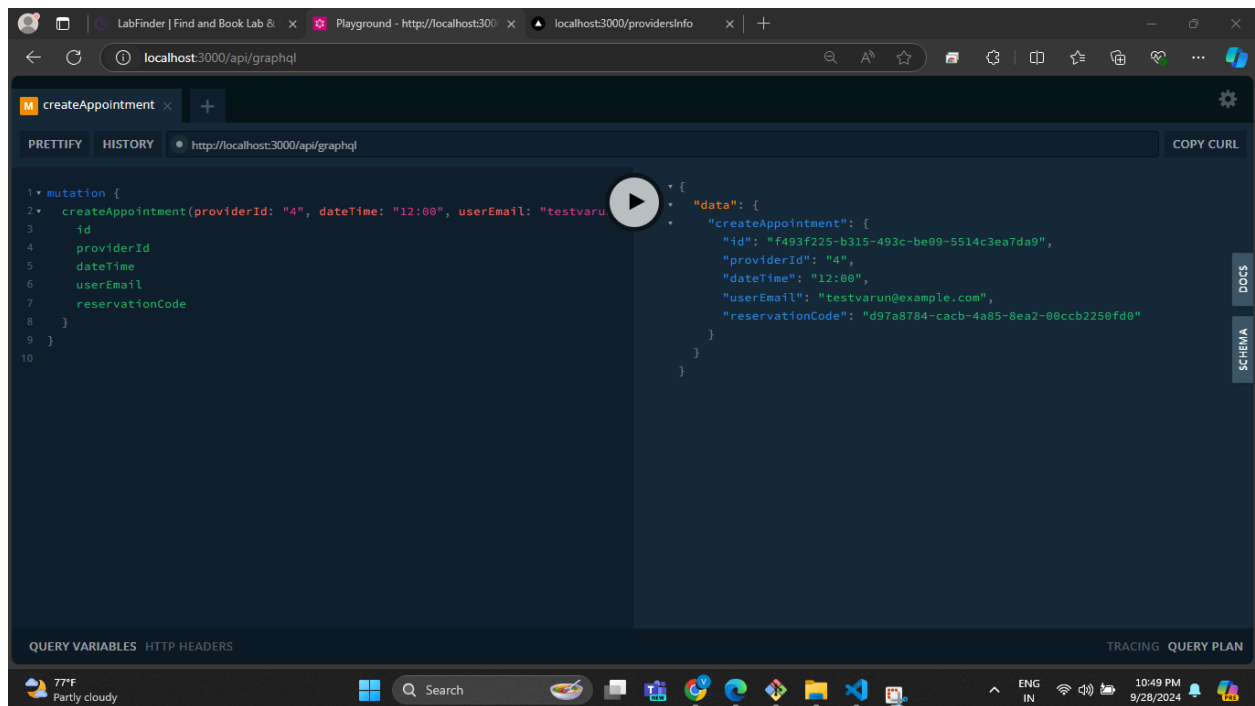
```
{  
  "data": {  
    "appointment": {  
      "id": "d9ebd4b6-5240-47b6-82e0-2681303ef1c7",  
      "providerId": "4",  
      "dateTime": "16:00",  
      "userEmail": "varu@example.com",  
      "reservationCode": "4c9calee-51b4-469a-b9f4-9e6f52a7891a"  
    }  
  }  
}
```

At the bottom of the interface, there are tabs for "QUERY VARIABLES", "HTTP HEADERS", "TRACING", and "QUERY PLAN". The Windows taskbar is visible at the very bottom, showing the time as 10:48 PM on 9/28/2024.

## 7.5 Create a New Appointment:

```
mutation {  
  createAppointment(providerId: "4", dateTime: "12:00", userEmail: "testvarun@example.com") {  
    id  
    providerId  
    dateTime  
    userEmail  
    reservationCode  
  }  
}
```

**Output:**



## 7.6 Reschedule Appointment:

```
mutation {  
  rescheduleAppointment(  
    appointmentId: "d9ebd4b6-5240-47b6-82e0-2681303ef1c7",  
    newDateTime: "16:00"  
  ) {  
    id  
    providerId  
    dateTime  
    userEmail  
    reservationCode  
  }  
}
```

### Output:

The screenshot displays a web browser window with the GraphQL Playground interface. The URL bar shows `localhost:3000/api/graphql`. The interface includes tabs for `rescheduleAppointment`, `PRETTIFY`, `HISTORY`, and `COPY CURL`. The query editor on the left contains the following mutation:

```
1 mutation {  
2   rescheduleAppointment(  
3     appointmentId: "d9ebd4b6-5240-47b6-82e0-2681303ef1c7",  
4     newDateTime: "16:00"  
5   ) {  
6     id  
7     providerId  
8     dateTime  
9     userEmail  
10    reservationCode  
11  }  
12 }  
13
```

The response viewer on the right shows the JSON output:

```
{  
  "data": {  
    "rescheduleAppointment": {  
      "id": "d9ebd4b6-5240-47b6-82e0-2681303ef1c7",  
      "providerId": "4",  
      "dateTime": "16:00",  
      "userEmail": "varu@example.com",  
      "reservationCode": "4c9calee-51b4-469a-b9f4-9e6f52a7891a"  
    }  
  }  
}
```

At the bottom of the interface, there are tabs for `QUERY VARIABLES`, `HTTP HEADERS`, `TRACING`, and `QUERY PLAN`. The Windows taskbar at the very bottom shows the system clock as 10:50 PM on 9/28/2024.

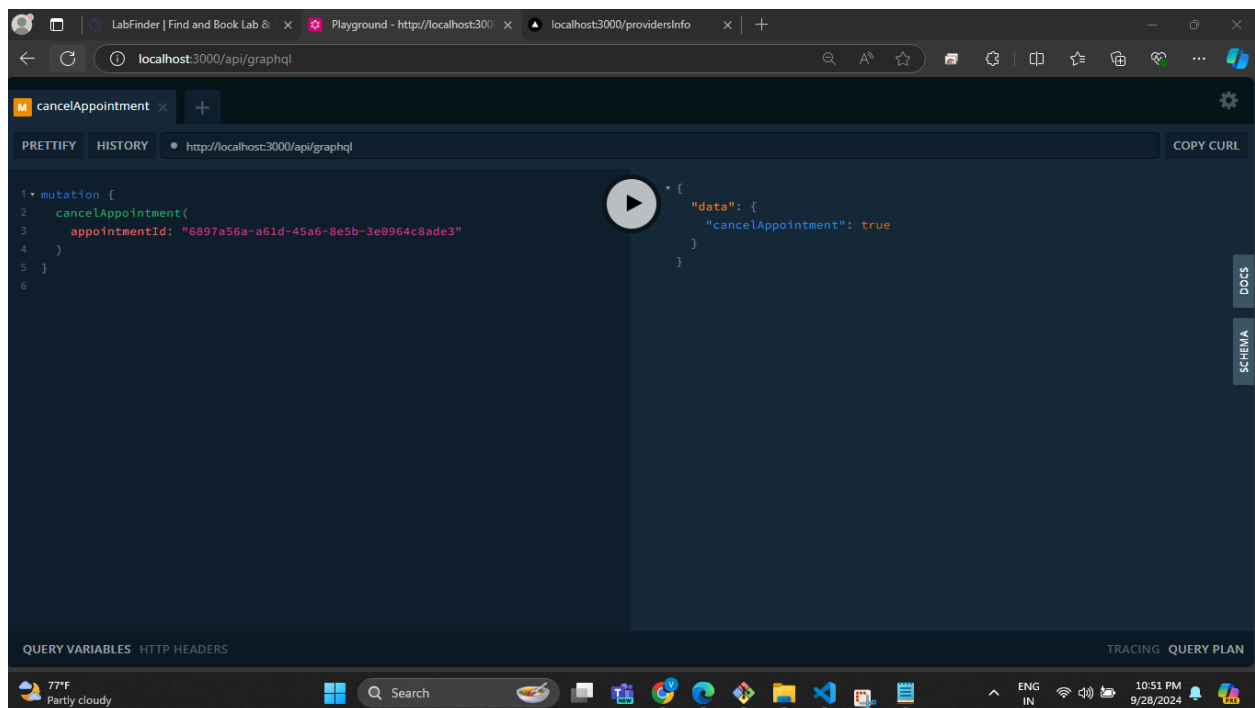


## 7.7 cancel appointment:

```
mutation {  
  cancelAppointment(  
    appointmentId: "583c0342-fe07-48db-856f-4d677f3a95c1"  
  )  
}
```

**Note:** The appointment with this ID has already been canceled. Please use a new `appointmentId` from `appointments.json` to validate this test case.

## Output:



The UI can be significantly improved. Due to time constraints, I couldn't allocate sufficient time to refine the front-end. For transparency, I utilized some online resources to generate a portion of the HTML content. This was my first experience implementing GraphQL in a real-world scenario. Setting up the server, writing the APIs, and testing them locally within 24 hours was challenging given the tight timeline. Nonetheless, I remained fully committed to completing the assessment to the best of my abilities. I sincerely appreciate the opportunity and your time. Thank you.