Documentação API Graphql

Renan Sales Ribeiro Pool

Professor Rommel Vieira

Março/2024

Subindo a aplicação

No diretório raíz, devemos executar o seguinte comando:

```
docker build -t api_node_graphql .
```

Para subir o container na porta 80, devemos executar o comando:

```
docker run -p 80:4000 api_node_graphql
```

Exemplos de consulta

Lista de todas as tarefas

```
query {
  tasks {
    id
    title
    completed
  }
}
```

Detalhes de uma tarefa específica com base no seu ID

```
query {
  task(id: "1") {
    id
    title
    completed
  }
}
```

Lista de tarefas concluídas

```
query {
  completedTasks {
    id
    title
  }
}
```

Lista de tarefas pendentes

```
query {
  pendingTasks {
    id
    title
  }
}
```

Lista de usuários

```
query {
  users {
    id
    name
    tasks {
      id
      title
      completed
    }
  }
}
```

Exemplos de Mutação

Criar uma nova tarefa

```
mutation {
  createTask(title: "Comprar mantimentos", userId: "1") {
    id
    title
    completed
  }
}
```

Marcar uma tarefa como concluída

```
mutation {
  markTaskAsCompleted(id: "1") {
```

```
    id
    title
    completed
  }
}
```

Atualizar informações de uma tarefa existente

```
mutation {
  updateTask(id: "2", title: "Estudar GraphQL") {
    id
    title
    completed
  }
}
```

Excluir uma tarefa

```
mutation {
  deleteTask(id: "3")
}
```

Documentação do código

```
const { ApolloServer, gql } = require('apollo-server');

Criando Dados fictícios para simular um banco de dados
let tasks = [
  { id: '1', title: 'Fazer compras', completed: false, userId: '1' },
  { id: '2', title: 'Estudar PHP', completed: true, userId: '2' },
];

let users = [
  { id: '1', name: 'Renan' },
  { id: '2', name: 'Rommel' },
];

Definição do esquema GraphQL
const typeDefs = gql`
```

```
  type Query {
    tasks: [Task]
    task(id: ID!): Task
    completedTasks: [Task]
    pendingTasks: [Task]
    users: [User]
  }

  type Mutation {
    createTask(title: String!, userId: ID!): Task
    markTaskAsCompleted(id: ID!): Task
    updateTask(id: ID!, title: String!): Task
    deleteTask(id: ID!): ID
  }

  type Task {
    id: ID!
    title: String!
    completed: Boolean!
    user: User!
  }

  type User {
    id: ID!
    name: String!
    tasks: [Task]
  }
`;

Resolvers para cada campo do esquema
const resolvers = {
  Query: {
    tasks: () => tasks,
    task: (parent, { id }) => tasks.find(task => task.id === id),
    completedTasks: () => tasks.filter(task => task.completed),
    pendingTasks: () => tasks.filter(task => !task.completed),
    users: () => users,
  },
  Mutation: {
    createTask: (parent, { title, userId }) => {
      const newTask = { id: String(tasks.length + 1), title, completed:
false, userId };
      tasks.push(newTask);
```

```
      return newTask;
    },
    markTaskAsCompleted: (parent, { id }) => {
      const taskIndex = tasks.findIndex(task => task.id === id);
      if (taskIndex === -1) throw new Error('Task not found');
      tasks[taskIndex].completed = true;
      return tasks[taskIndex];
    },
    updateTask: (parent, { id, title }) => {
      const taskIndex = tasks.findIndex(task => task.id === id);
      if (taskIndex === -1) throw new Error('Task not found');
      tasks[taskIndex].title = title;
      return tasks[taskIndex];
    },
    deleteTask: (parent, { id }) => {
      const taskIndex = tasks.findIndex(task => task.id === id);
      if (taskIndex === -1) throw new Error('Task not found');
      tasks.splice(taskIndex, 1);
      return id;
    },
  },
  Task: {
    user: (parent) => users.find(user => user.id === parent.userId),
  },
  User: {
    tasks: (parent) => tasks.filter(task => task.userId === parent.id),
  },
};

Configuração do servidor Apollo GraphQL
const server = new ApolloServer({ typeDefs, resolvers });

Inicialização do servidor
server.listen().then(({ url }) => {
  console.log(`Server ready at ${url}`);
});
```