

## Module 7: Fetch Data using GraphQL

---

### Demo Document 2

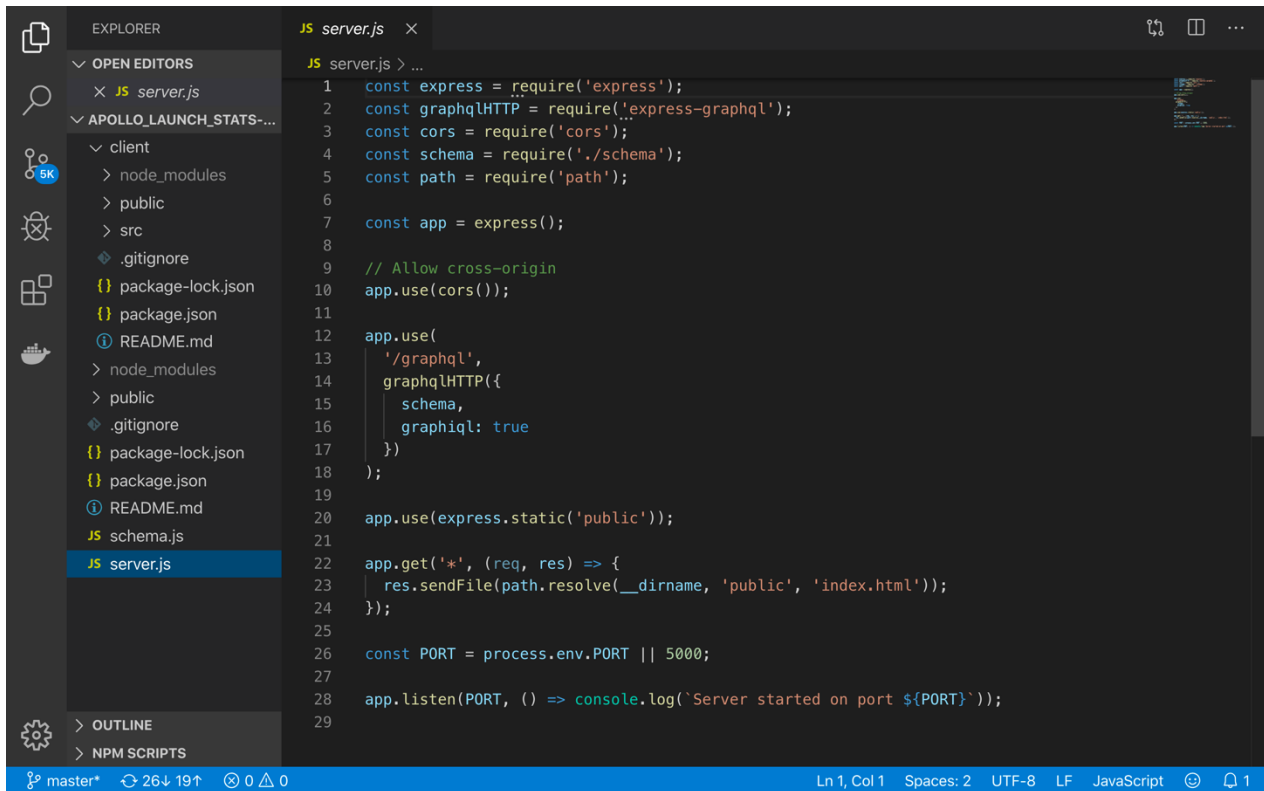
edureka!

**edureka!**

© Brain4ce Education Solutions Pvt. Ltd.

## Fetch space launch data using Apollo-GraphQL

**Step 1:** For consuming the **GraphQL API**, we need to have **GraphQL server** which will generate the **GraphQL endpoint**. Using **express-graphql** create the GraphQL server.

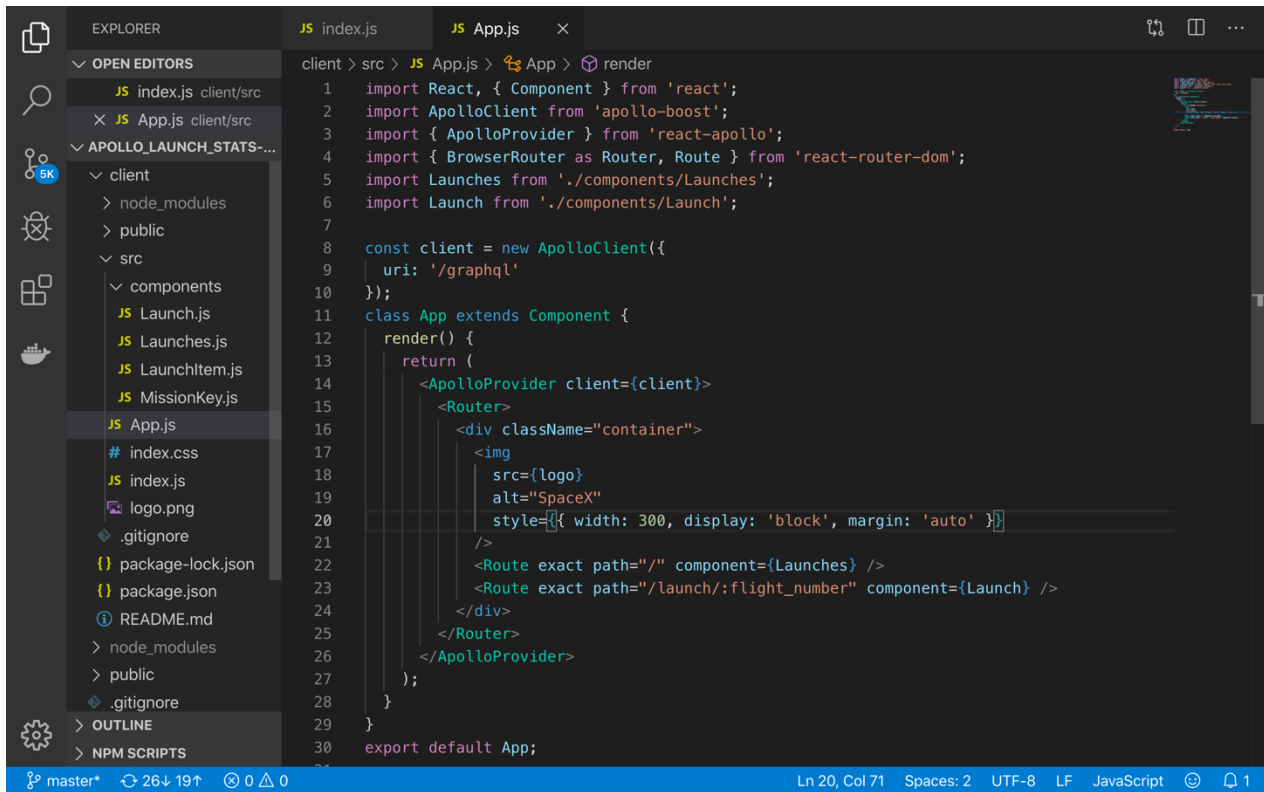


The screenshot shows a VS Code editor with a project structure on the left and a JavaScript file named `server.js` open in the editor. The project structure includes a `client` directory with `node_modules`, `public`, and `src` subdirectories, and a `server` directory with `node_modules`, `public`, and `src` subdirectories. The `server.js` file contains the following code:

```
1 const express = require('express');
2 const graphqlHTTP = require('express-graphql');
3 const cors = require('cors');
4 const schema = require('./schema');
5 const path = require('path');
6
7 const app = express();
8
9 // Allow cross-origin
10 app.use(cors());
11
12 app.use(
13   '/graphql',
14   graphqlHTTP({
15     schema,
16     graphiql: true
17   })
18 );
19
20 app.use(express.static('public'));
21
22 app.get('*', (req, res) => {
23   res.sendFile(path.resolve(__dirname, 'public', 'index.html'));
24 });
25
26 const PORT = process.env.PORT || 5000;
27
28 app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
29
```

**Step 2:** To **consume** the **GraphQL API** in React we have to use **Apollo client** from **react-apollo** package. Make a call to the GraphQL API via Apollo client.

Now inside React Provider we need to use **ApolloProvider** and pass client as GraphQL client in the App.js file.

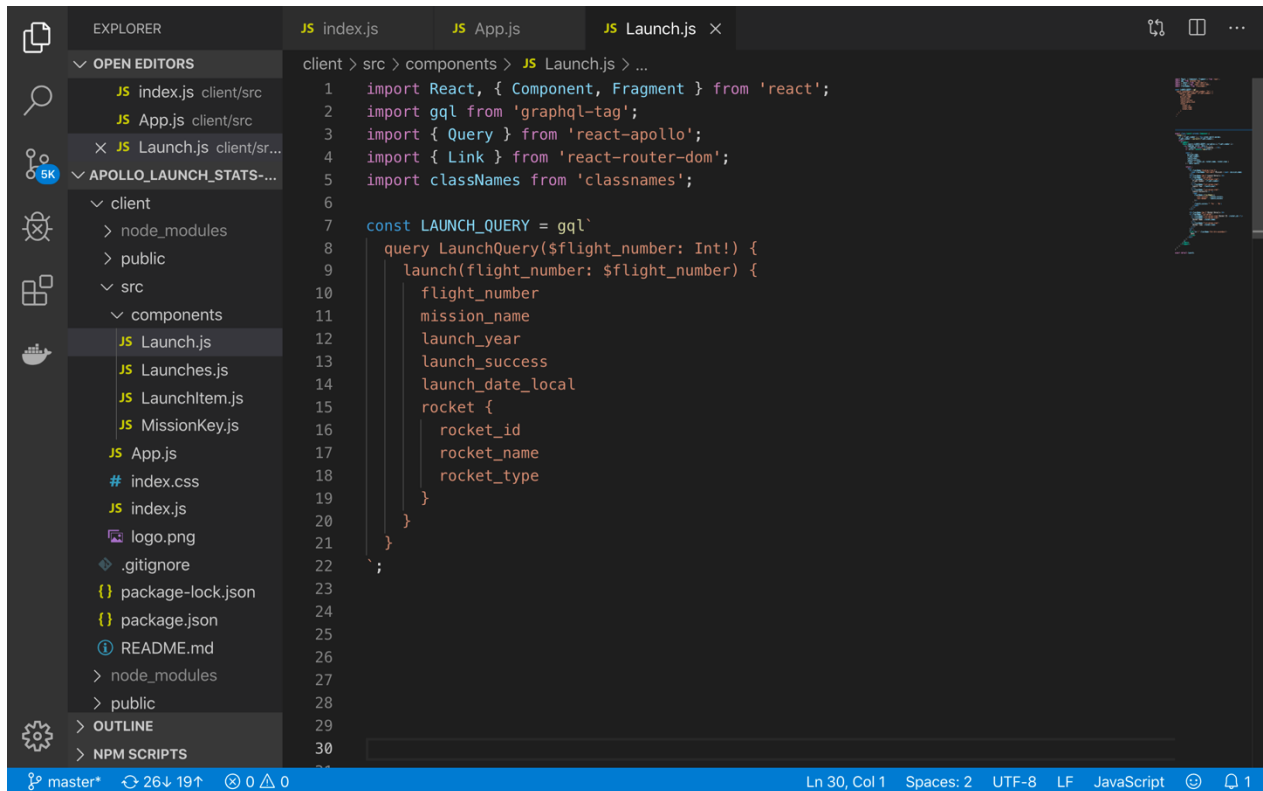


The screenshot shows a VS Code editor with the following structure:

- EXPLORER:**
  - client > src > JS App.js
  - client > src > JS index.js
  - client > src > JS Launches.js
  - client > src > JS LaunchItem.js
  - client > src > JS MissionKey.js
  - client > src > JS App.js
  - client > src > # index.css
  - client > src > JS index.js
  - client > src > logo.png
  - client > src > .gitignore
  - client > src > package-lock.json
  - client > src > package.json
  - client > src > README.md
  - client > node\_modules
  - client > public
  - client > .gitignore
  - client > OUTLINE
  - client > NPM SCRIPTS
- App.js:**

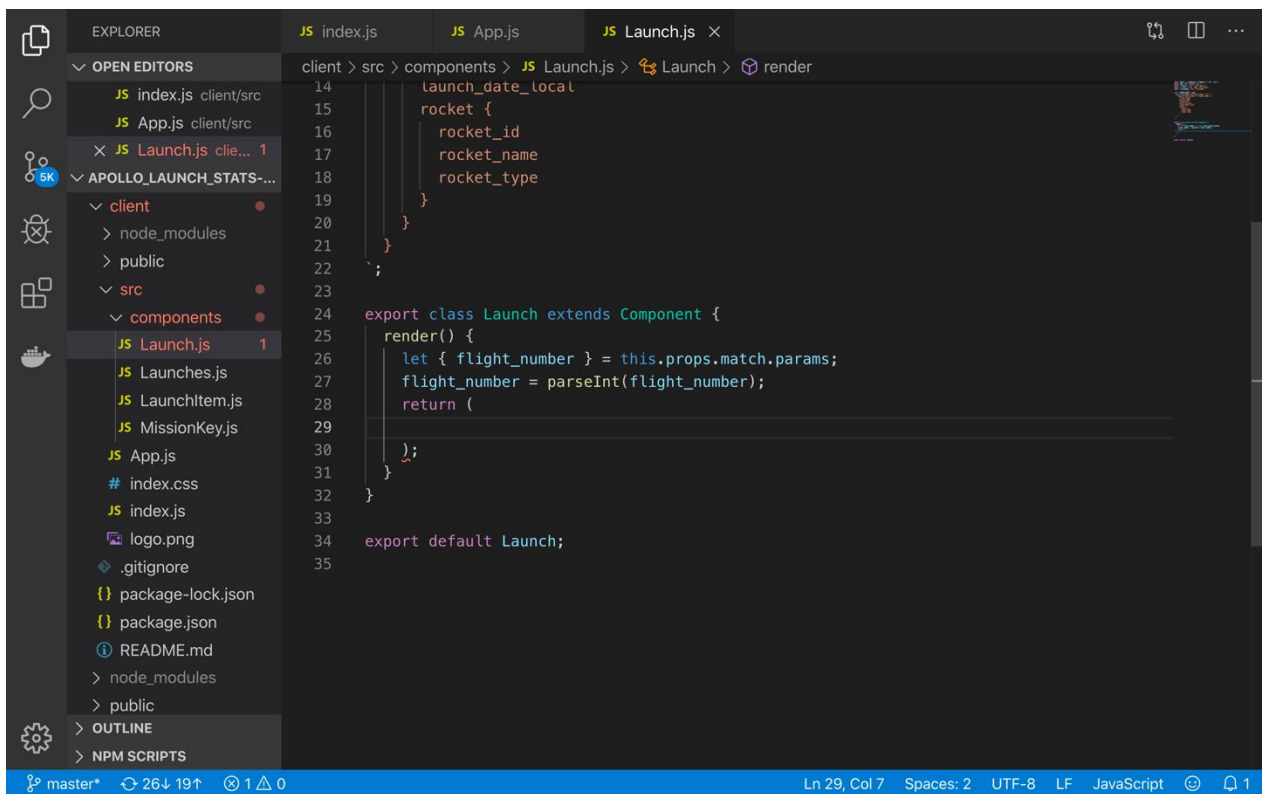
```
1 import React, { Component } from 'react';
2 import ApolloClient from 'apollo-boost';
3 import { ApolloProvider } from 'react-apollo';
4 import { BrowserRouter as Router, Route } from 'react-router-dom';
5 import Launches from './components/Launches';
6 import Launch from './components/Launch';
7
8 const client = new ApolloClient({
9   uri: '/graphql'
10 });
11
12 class App extends Component {
13   render() {
14     return (
15       <ApolloProvider client={client}>
16         <Router>
17           <div className="container">
18             <img
19               src={logo}
20               alt="SpaceX"
21               style={{ width: 300, display: 'block', margin: 'auto' }} />
22             <Route exact path="/" component={Launches} />
23             <Route exact path="/launch/:flight_number" component={Launch} />
24           </div>
25         </Router>
26       </ApolloProvider>
27     );
28   }
29 }
30 export default App;
```

**Step 3:** In GraphQL, to receive data we need to create a **query structure** on the basis of which we can make an **API call** and **GraphQL server** will return the data.



```
1 import React, { Component, Fragment } from 'react';
2 import gql from 'graphql-tag';
3 import { Query } from 'react-apollo';
4 import { Link } from 'react-router-dom';
5 import classNames from 'classnames';
6
7 const LAUNCH_QUERY = gql`
8   query LaunchQuery($flight_number: Int!) {
9     launch(flight_number: $flight_number) {
10       flight_number
11       mission_name
12       launch_year
13       launch_success
14       launch_date_local
15       rocket {
16         rocket_id
17         rocket_name
18         rocket_type
19       }
20     }
21   }
22 `;
23
24
25
26
27
28
29
30
```

**Step 4:** Below schema create a component to receive the data.



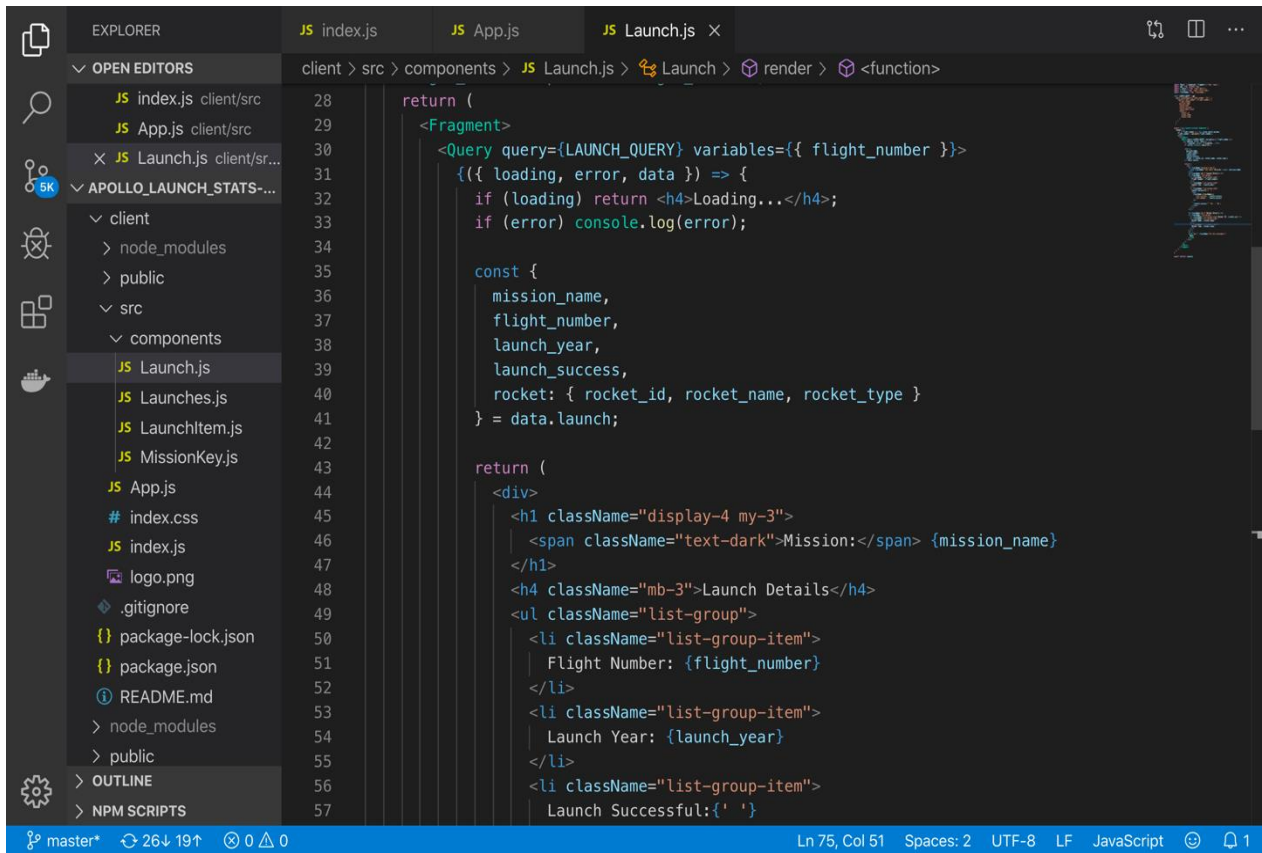
The screenshot shows a VS Code editor with the following structure:

- EXPLORER:** Shows the file tree with folders like `client`, `src`, and `components`. The file `JS Launch.js` is selected.
- Editor:** Displays the code for `Launch.js`. The code defines a `Launch` component that receives `launch_date_local` and `rocket` props. The `render` method uses `props.match.params` to get the `flight_number` and calls `parseInt` on it.

```
14 launch_date_local
15 rocket {
16   rocket_id
17   rocket_name
18   rocket_type
19 }
20 }
21
22 ;
23
24 export class Launch extends Component {
25   render() {
26     let { flight_number } = this.props.match.params;
27     flight_number = parseInt(flight_number);
28     return (
29
30   );
31 }
32
33
34 export default Launch;
35
```

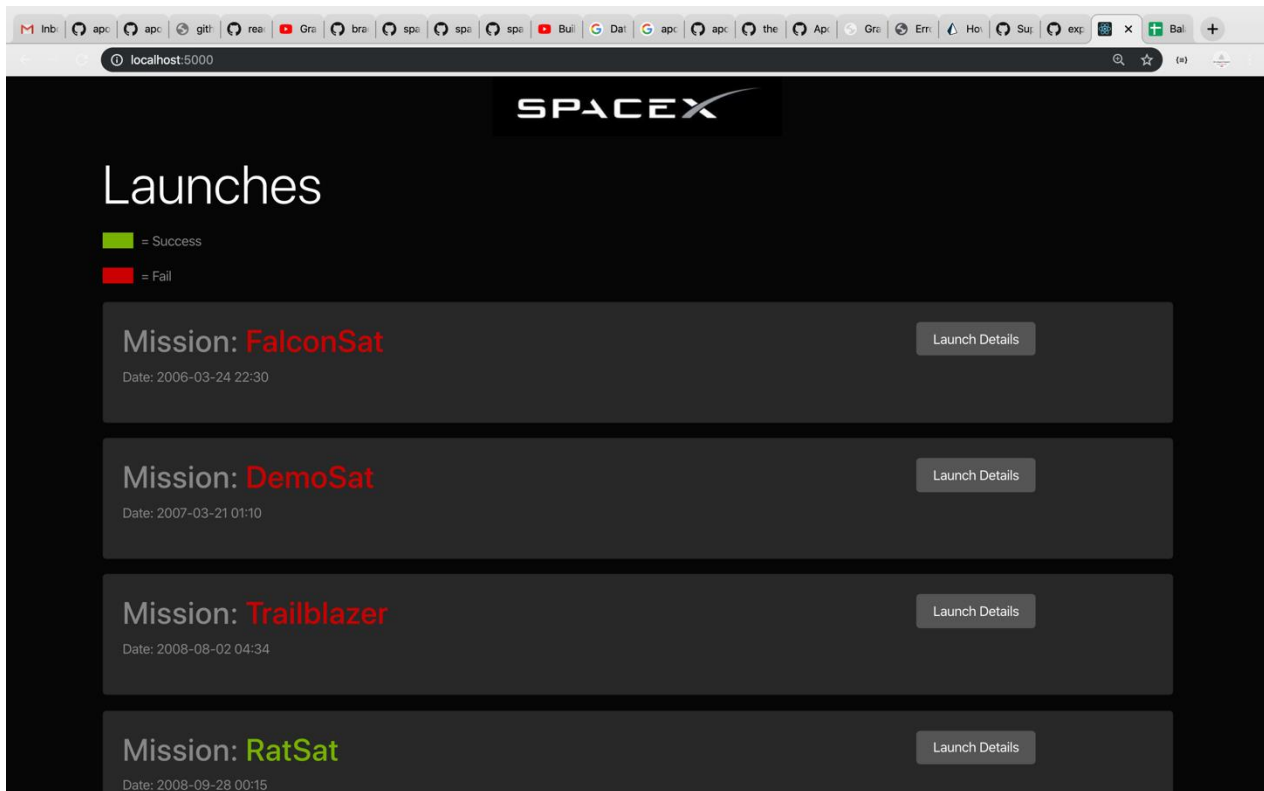
edureka!

**Step 5:** Perform the data binding as shown below. Define all the variable as constant for which we want to receive the data.

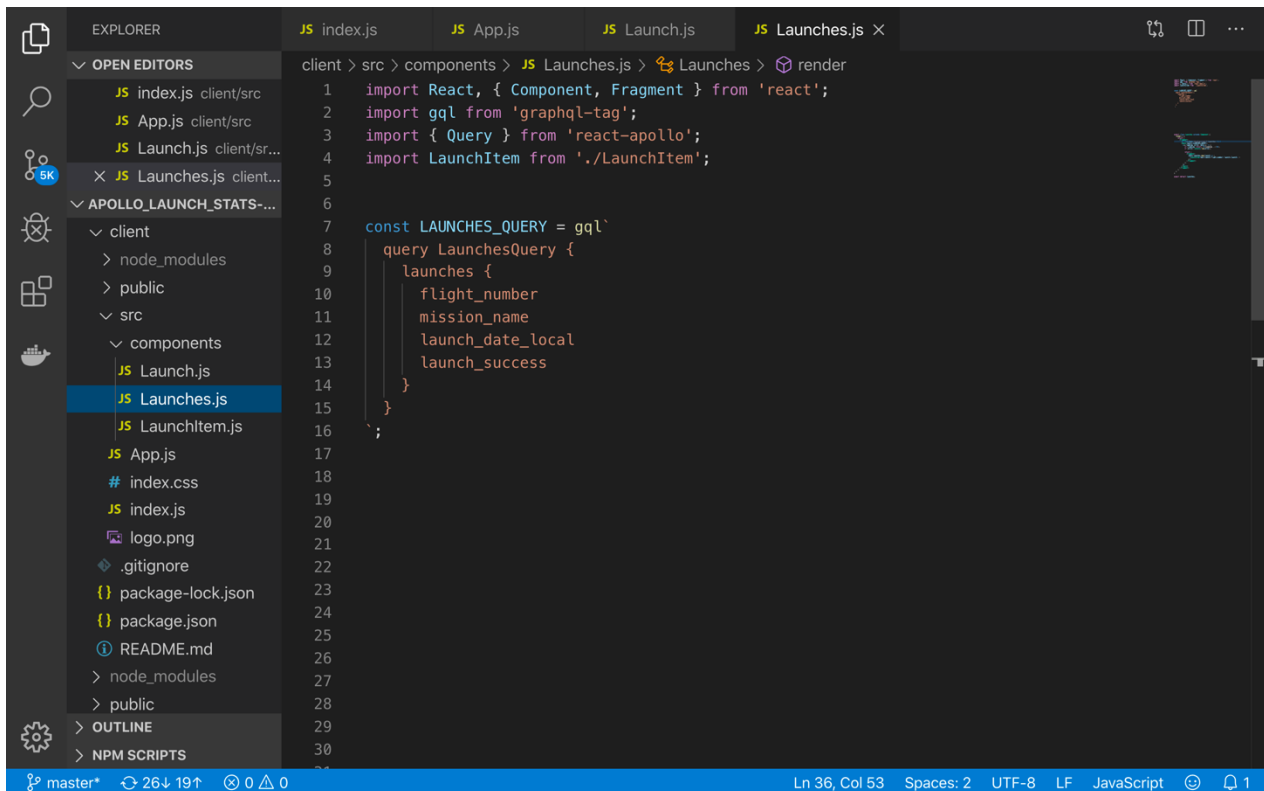


```
28 return (  
29   <Fragment>  
30     <Query query={LAUNCH_QUERY} variables={{ flight_number }}>  
31       {{ loading, error, data }} => {  
32         if (loading) return <h4>Loading...</h4>;  
33         if (error) console.log(error);  
34  
35         const {  
36           mission_name,  
37           flight_number,  
38           launch_year,  
39           launch_success,  
40           rocket: { rocket_id, rocket_name, rocket_type }  
41         } = data.launch;  
42  
43         return (  
44           <div>  
45             <h1 className="display-4 my-3">  
46               <span className="text-dark">Mission:</span> {mission_name}  
47             </h1>  
48             <h4 className="mb-3">Launch Details</h4>  
49             <ul className="list-group">  
50               <li className="list-group-item">  
51                 Flight Number: {flight_number}  
52               </li>  
53               <li className="list-group-item">  
54                 Launch Year: {launch_year}  
55               </li>  
56               <li className="list-group-item">  
57                 Launch Successful: {launch_success}  
58             </ul>  
59           </div>  
60         )  
61       }  
62     </Query>  
63   </Fragment>  
64 )
```

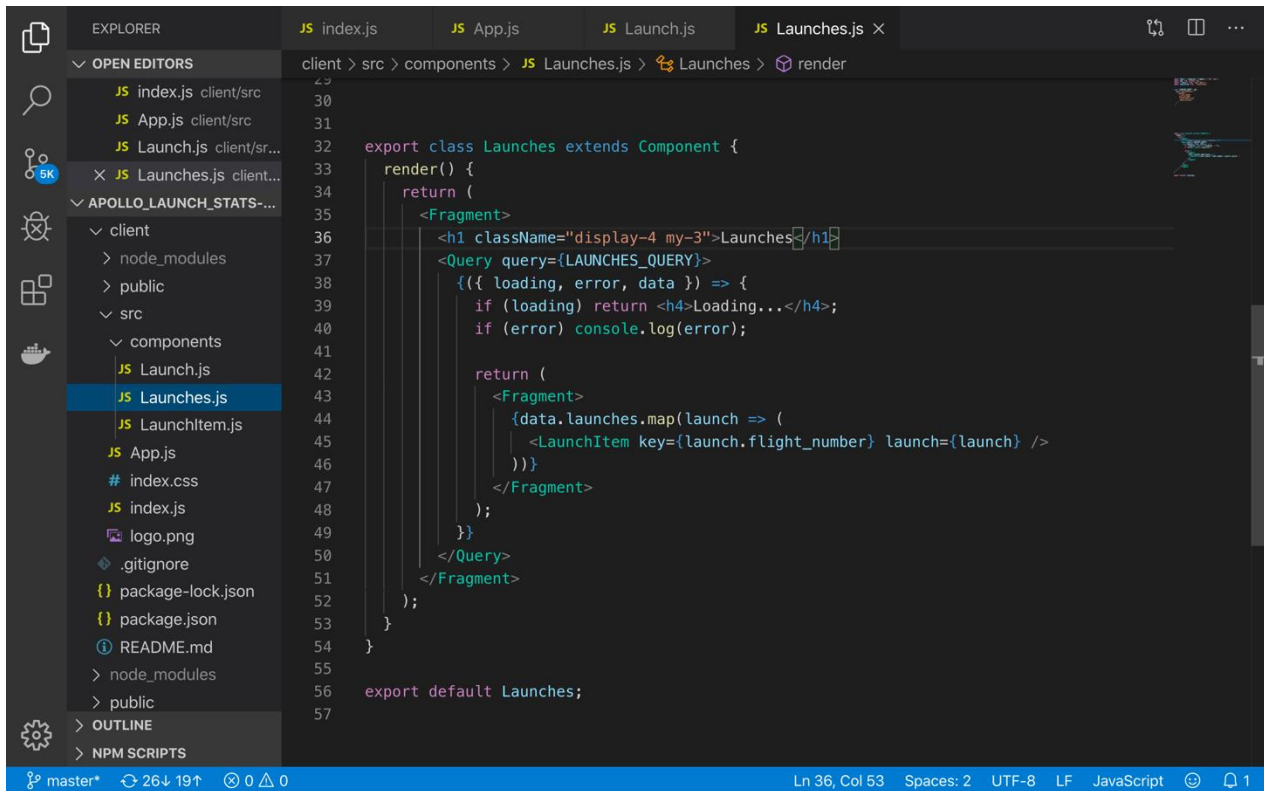
**Step 6:** Now when we check our output for the component it will show the all launches from the spaceX API call.



**Step 7:** Now for the next component we have to again create **gql schema** for rest of data collection for the next page.



**Step 8:** Now again perform the data binding to display the details of next page.



```

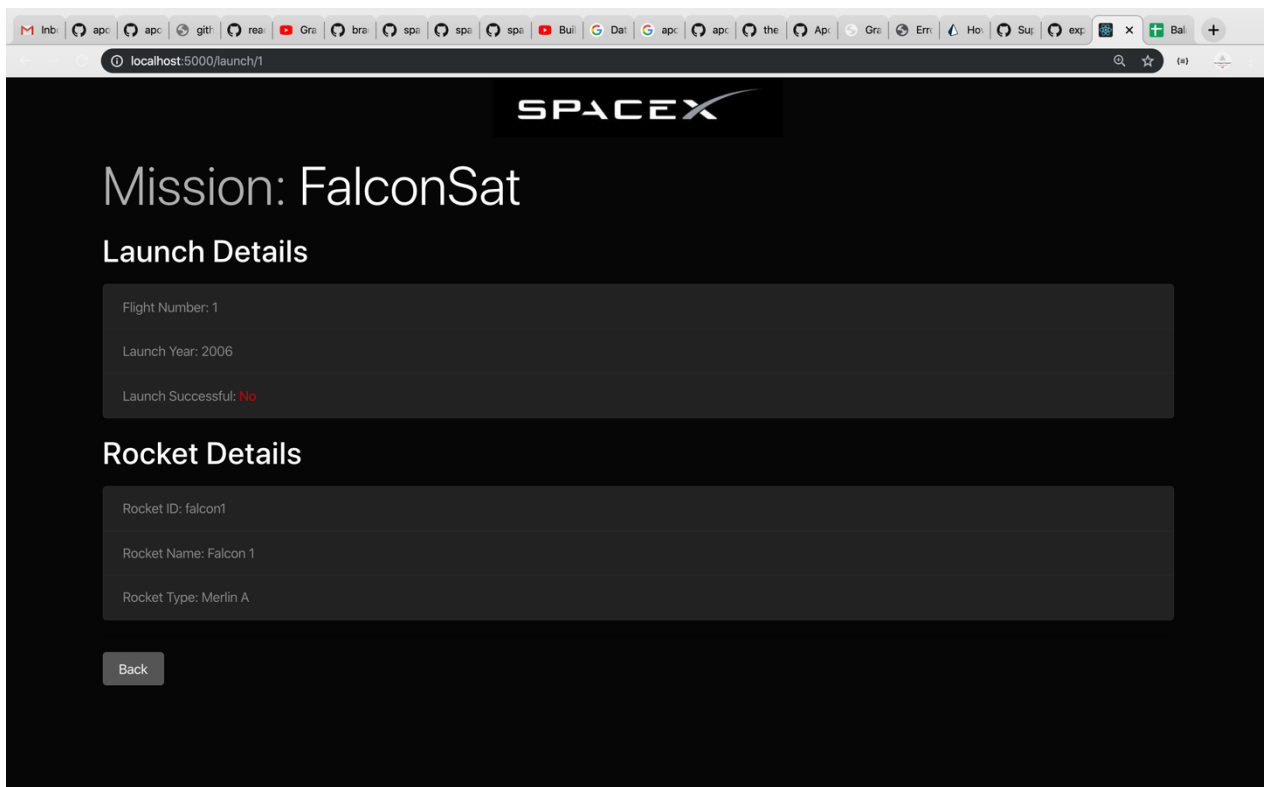
export class Launches extends Component {
  render() {
    return (
      <Fragment>
        <h1 className="display-4 my-3">Launches</h1>
        <Query query={LAUNCHES_QUERY}>
          {({ loading, error, data }) => {
            if (loading) return <h4>Loading...</h4>;
            if (error) console.log(error);

            return (
              <Fragment>
                {data.launches.map(launch => (
                  <LaunchItem key={launch.flight_number} launch={launch} />
                ))}
              </Fragment>
            );
          }}
        </Query>
      </Fragment>
    );
  }
}

export default Launches;

```

**Step 8:** For the next component we will receive the data for all the launches and its details, from the GraphQL API.





## Conclusion:

We have successfully created an application using GraphQL and Apollo Client to fetch and display the space launch data.

edureka!