There will be three steps

1. Server side (Back-end)
2. Data Base
3. Client side (Front-end)

1.Server side (Back-end):

Create a directory for your server-side code. You can name it "server-side":

Initializing npm
In the "server-side" directory, initialize npm by running the following command:

```
npm init -y
```

This command will generate a package.json file with default settings.

Initialize TypeScript in your project by creating a tsconfig.json file. Run:

```
tsc --init
```

Installing Required Libraries

Install the necessary libraries using npm. Run the following commands one by one:

- To install Node.js types for TypeScript:
  `npm install -D @types/node`
- To install the Express library:
  `npm install express`
- To install the CORS library:
  `npm install cors`
- To install the SQL library for Node.js (msnodesqlv8):
  `npm install msnodesqlv8`

Creating the Main Server File
Inside the "server-side" directory, create a file named index.ts

```
const express = require("express");
```

This imports the 'express' module, which provides a framework for building web applications in Node.js.

```
const cors = require('cors');
```

Imports the 'cors' module, which is used to handle Cross-Origin Resource Sharing and allows the server to respond to requests from different origins.

```
const app = express();
```

Creates an instance of the Express application.

```
app.use(cors());
```

Adds the 'cors' middleware to the Express application, enabling CORS for all routes.

```
const sql = require('msnodesqlv8');
```

Imports the 'msnodesqlv8' module, which is used to interact with SQL Server databases.

```
const
connectionString="server=serverName;Database=dbName;Trusted_Connection=Yes;Driv
er={SQL Server Native Client 11.0}";
```

Defines the connection string for the SQL Server database. This string includes the server name, database name, authentication mode, and driver details.

```
app.get('/data', ({ req, res }:any) => {
```

Defines a route for handling GET requests to the '/data' endpoint.

```
var query = 'SELECT * FROM Customer';
```

Defines an SQL query to select all rows from the 'Customer' table.

```
sql.query(connectionString,query, (err:any,rows:any)=>{...}
```

Executes the SQL query using the connection string and handles the result.

```
app.listen(4000,()=>{...}
```

Starts the Express server on port 4000. When the server is successfully started, it logs a message

After add code in index.ts file then run your back end project by running below command

```
tsc
```

```
node index
```

Your project will run on below url

http://localhost:4000/data

2. Database setup:

Step 1: Download and Install Microsoft SQL Server:

- Visit the official Microsoft SQL Server Downloads page: https://www.microsoft.com/en-us/sql-server/sql-server-downloads
- Choose the edition of SQL Server you want to download. For most users, the "Express" edition is a good choice as it's free and suitable for learning and small applications.
- Follow the on-screen instructions to download the installer executable (typically an .exe file).
- Run the downloaded installer executable.
- During the installation process, choose the appropriate installation type and follow the prompts. You can typically choose the default settings for most options.

Step 2: Connect to SQL Server using Windows Authentication:

- After installing SQL Server, you'll need to connect to it using a SQL Server management tool. One popular choice is "SQL Server Management Studio" (SSMS), which can be downloaded separately from:

  https://www.microsoft.com/en-us/sql-server/sql-server-downloads

- Open SQL Server Management Studio (SSMS) after installation.
- In the "Connect to Server" dialog:
  - Server Type: Choose "Database Engine."

- ❖ Server Name: Enter the name of your SQL Server instance. This could be the server's name or its IP address.
- ❖ Authentication: Choose "Windows Authentication."
- ❖ Click the "Connect" button. If the connection is successful, you should be connected to the SQL Server instance.

Step 3: Create a Database and Table:

In SQL Server Management Studio:

- In the "Object Explorer" panel on the left, right-click on "Databases" and choose "New Database."
- Give your database a name, e.g., "storeDb."
- Click "OK" to create the database.

With the "storeDb" database selected:

- Right-click on "Tables" under your new database and choose "New Table."
- Design your table by adding columns. For example, you can add columns like "CustomerID," "FirstName," "LastName," etc.
- Set the appropriate data types and constraints for each column.
- Click the "Save" button to save the table.

Your "storeDb" database with the "Customer" table is now set up. You can use this database to interact with your backend code.

3. Client Side (Front-end)

Create a New React App

Open your terminal or command prompt.

To create a new React app, you can use npx (a package runner tool that comes with npm) with the following command:

```
npx create-react-app my-react-app
```

Replace my-react-app with the desired name for your project. This command will set up a new React app in a folder with the specified name.

Editing the App

Open the src/App.js file in a text editor or an integrated development environment (IDE).

You can start editing the App component or create new components in separate files within the src directory.

```
const [customers, setCustomers] = useState([]);
```

This code snippet uses React's useState and useEffect hooks to fetch customer data from an API and manage the state of the customers array. Let's break it down step by step:

const [customers, setCustomers] = useState([]);

useState([]) is a React hook that initializes the state variable customers with an empty array. It also returns a function setCustomers that can be used to update the state.

```
useEffect(() => {...}
```

useEffect is another React hook that allows you to perform side effects in functional components. It takes two arguments: a function to run and an array of dependencies.

Inside the useEffect function:

- fetch('http://localhost:4000/data'): This initiates a network request to the specified URL (http://localhost:4000/data), which is assumed to be an API endpoint that returns customer data.

- .then(response => response.json()): This is a promise chain that processes the response from the API. It uses the .json() method to parse the response and extract the JSON data.

- .then(data => { ... }): This promise is triggered when the JSON data is successfully extracted from the response. The parsed data is then passed to the provided function.

- Inside the { ... } block, console.log("Data from API:", data); logs the fetched data to the console for debugging purposes.

- setCustomers(data);: This line updates the customers state with the fetched data. When setCustomers is called, React re-renders the component with the updated state, causing the table to display the fetched customer data.

Run your front end project with following command

```
npm start
```