

CS 628: Full-Stack Development – Web App

City University of Seattle
School of Technology & Computing
Professor Sam Chung

HOS09: MERN

Samantha Hipple
August 28, 2023

PLEASE NOTE

Screenshots in this guide may differ from your environment (e.g., directory paths, version numbers, etc.). When choosing between a stable or most recent release, we advise you install the stable release rather than the best-testing version. Additionally, there may be subtle discrepancies along the steps, please use your best judgment to complete the tutorial. If you are unfamiliar with terminal, command line, and bash scripts, we recommend watching [this video](#) prior to moving forward with this guide. Not all steps are fully explained. Lastly, we advise that you avoid copy-pasting code directly from the guide or GitHub repositories. Instead, type out the code yourself to improve familiarity.

More information on this guide can be found under the related module in [this repository](#).

SECTION CONTENTS

1. Accessing GitHub Codespaces
 2. Project initialization
 3. Setting up the React router
 4. Creating the components
 5. Connecting the front and backends
-

SECTION 1. ACCESSING GITHUB CODESPACES

GitHub Codespaces is an online cloud-based development environment that allows users to easily write, run and debug code. Codespaces is fully integrated with your GitHub repository and provides a seamless experience for developers. In order to access Codespaces, users only need a GitHub account and an active internet connection.

After downloading the current HOS assignment, in the top-right corner of the repo, click on the `<>` **Code** drop-down menu and select **Create codespace on main** as shown in the following image. The free and pro GitHub subscriptions include free use of GitHub Codespaces *up to a fixed amount of usage each month*. In order to avoid unexpected charges, please review the [billing information](#).

NOTE: *This tutorial is an extension of the previous guide in the HOS08 module. Please create a copy of the backend application from HOS08 and move it into your HOS09 repository, or plan to execute the backend from the HOS08 directory if you are using Codespaces.*

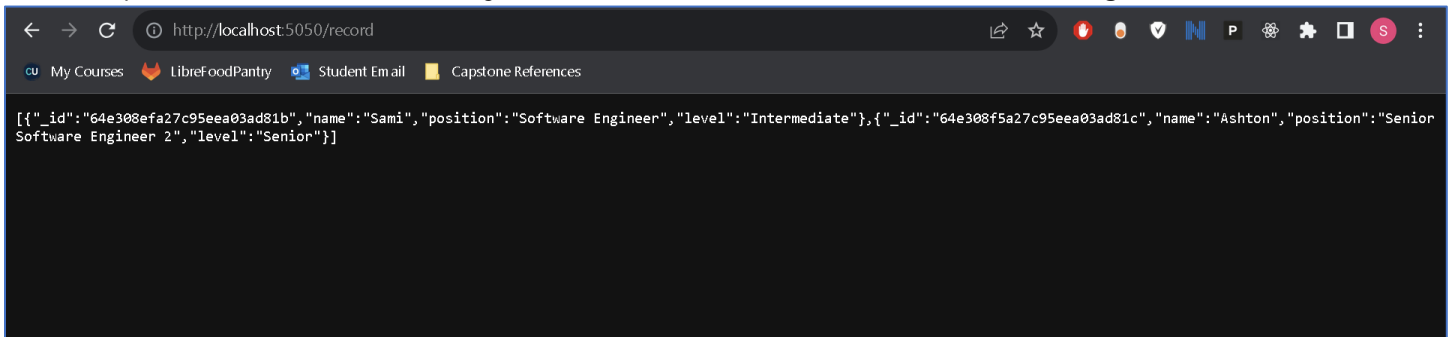
SECTION 2. PROJECT INITIALIZATION

Before we begin creating our frontend, we must make sure that the backend server from the previous module is up and running and test our API.

1. Navigate to the appropriate backend directory.
2. Start the server using the command `node server.mjs`.

NOTE: *If you are using Codespaces, you must make the port running your backend server public, as discussed in the previous module.*

3. Open the local address in your browser to check if the API is working.



Once everything is in order with the backend, we are ready to start creating the frontend of our web application!

4. Open a new terminal and initialize the frontend using the following commands:

```
>>mkdir mern
>>cd mern
>>npx create-react-app client
```

Next, we'll navigate to the **client** directory in the terminal and proceed to install two extra dependencies essential for our project.

5. Execute the following commands:

```
>>cd client
>>npm install react-router-dom bootstrap
```

Bootstrap enables rapid deployment of templates and components for your new web app, sparing you from starting from scratch, while **react-router-dom** installs React router components tailored for web applications.

SECTION 3. SETTING UP THE REACT ROUTER

We will be using the **BrowserRouter** component reviewed in previous models to ensure that our UI stays synchronized with the URL, enable smooth transitions when moving between components.

1. Replace the code in **index.js** with the following:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
import { BrowserRouter } from "react-router-dom";

ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById("root")
);
```

SECTION 4. CREATING THE COMPONENTS

Now, we need to create the components for our application. Ensure you are under the **mern/client** directory for the following steps.

1. Use the following terminal commands to create our UI component files:

```
>>mkdir src/components
>>cd src/components
>>touch create.js edit.js navbar.js record-list.js
```

2. Open **create.js** and add the following code:

```
import { useState } from "react";
import { useNavigate } from "react-router";

const Create = () => {
  const navigate = useNavigate();
  const emptyForm = { name: "", position: "", level: "" };
  const [form, setForm] = useState({ emptyForm });
  // method to update the state properties
  const updateForm = (value) => setForm(
    (prev) => ({ ...prev, ...value })
  );
  // method to handle form submission
  const onSubmit = async (e) => {
    e.preventDefault();
    // when a POST request is sent to /create URL, we add a new record to the database
    const newPerson = { ...form }
```

```

    await fetch("http://localhost:5050/record", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(newPerson)
    }).catch((error) => window.alert(error));
    setForm({ emptyForm });
    navigate("/");
  }
}

```

// display the form that takes the input from the user

```

return (
  <div>
    <h3>Create New Record</h3>
    <form onSubmit={onSubmit}>
      <div className="form-group">
        <label htmlFor="name">Name</label>
        <input
          type="text" className="form-control" id="name"
          value={form.name}
          onChange={(e) => updateForm({ name: e.target.value })} />
      </div>
      <div className="form-group">
        <label htmlFor="position">Position</label>
        <input
          type="text" className="form-control" id="position"
          value={form.position}
          onChange={(e) => updateForm({ position: e.target.value })} />
      </div>

```

```

      <div className="form-group">
        <div className="form-check form-check-inline">
          <input
            className="form-check-input" type="radio" name="positionOptions"
            id="positionIntern" value="Intern"
            checked={form.level === "Intern"}
            onChange={(e) => updateForm({ level: e.target.value })} />
          <label htmlFor="positionIntern" className="form-check-label">Intern</label>
        </div>
        <div className="form-check form-check-inline">
          <input
            className="form-check-input" type="radio" name="positionOptions"
            id="positionJunior" value="Junior"
            checked={form.level === "Junior"}
            onChange={(e) => updateForm({ level: e.target.value })} />
          <label htmlFor="positionJunior" className="form-check-label">Junior</label>
        </div>
        <div className="form-check form-check-inline">
          <input
            className="form-check-input" type="radio" name="positionOptions"
            id="positionSenior" value="Senior"
            checked={form.level === "Senior"}
            onChange={(e) => updateForm({ level: e.target.value })} />
          <label htmlFor="positionSenior" className="form-check-label">Senior</label>
        </div>
      </div>
    </div>

```

```

    <div className="form-group">
      <input type="submit" value="Create person" className="btn btn-primary" />
    </div>
  </form>
</div>
);
}

export default Create;

```

The provided code will function as a creation component for our records, allowing users to generate new entries. Through this component, a create command will be sent to our server.

3. Open `edit.js` and add the following code:

```

import { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router";

const Edit = () => {
  const emptyForm = { name: "", position: "", level: "", records: [] }
  const navigate = useNavigate();
  const params = useParams();
  const [form, setForm] = useState({ emptyForm });

  useEffect(() => {
    const fetchData = async () => {
      const id = params.id.toString();
      const response = await fetch(`http://localhost:5050/record/${id}`);
      if (!response.ok) {
        const message = `An error occurred: ${response.statusText}`;
        window.alert(message);
      }
      const record = await response.json();
      if (!record) {
        window.alert(`Record with id ${id} not found.`);
        navigate("/");
      }
      setForm(record);
    }
    fetchData();
  }, [params.id, navigate]);

  // methods to update the state properties
  const updateForm = (value) => setForm((prev) => ({ ...prev, ...value }));
  const onSubmit = async (e) => {
    e.preventDefault();
    const editedPerson = { name: form.name, position: form.position, level: form.level }
    // send a POST request to update the data in the database
    await fetch(`http://localhost:5050/record/${params.id}`, {
      method: "PATCH",

```

```

    body: JSON.stringify(editedPerson),
    headers: { 'Content-Type': 'application/json' },
  });
  navigate("/");
}

```

// display the form that takes input from the user to update the data

```

return (
  <div>
    <h3>Update Record</h3>
    <form onSubmit={onSubmit}>
      <div className="form-group">
        <label htmlFor="name">Name: </label>
        <input
          type="text" className="form-control" id="name"
          value={form.name}
          onChange={(e) => updateForm({ name: e.target.value })} />
      </div>
      <div className="form-group">
        <label htmlFor="position">Position: </label>
        <input
          type="text" className="form-control" id="position"
          value={form.position}
          onChange={(e) => updateForm({ position: e.target.value })} />
      </div>

```

```

    <div className="form-group">
      <div className="form-check form-check-inline">
        <input
          className="form-check-input" type="radio" name="positionOptions"
          id="positionIntern" value="Intern"
          checked={form.level === "Intern"}
          onChange={(e) => updateForm({ level: e.target.value })} />
        <label htmlFor="positionIntern" className="form-check-label">Intern</label>
      </div>
      <div className="form-check form-check-inline">
        <input
          className="form-check-input" type="radio" name="positionOptions"
          id="positionJunior" value="Junior"
          checked={form.level === "Junior"}
          onChange={(e) => updateForm({ level: e.target.value })} />
        <label htmlFor="positionJunior" className="form-check-label">Junior</label>
      </div>
      <div className="form-check form-check-inline">
        <input
          className="form-check-input" type="radio" name="positionOptions"
          id="positionSenior" value="Senior"
          checked={form.level === "Senior"}
          onChange={(e) => updateForm({ level: e.target.value })} />
        <label htmlFor="positionSenior" className="form-check-label">Senior</label>
      </div>
    </div>
  </div>
  <br />

```

```

      <div className="form-group">
        <input type="submit" value="Update Record" className="btn btn-primary" />
      </div>
    </form>
  </div>
);
}

export default Edit;

```

The **Edit** component will act as an editing component for our records, adopting a layout like the **Create** component, and ultimately sending an update command to our server.

4. Open **navbar.js** and add the following code:

```

import { NavLink } from 'react-router-dom';
import "bootstrap/dist/css/bootstrap.css";

// display the Navbar
const NavBar = () => (
  <div>
    <nav className="navbar navbar-expand-lg navbar-light bg-light">
      <NavLink className="navbar-brand" to="/">
        
        </img>
      </NavLink>
      <button className="navbar-toggler" type="button"
        data-toggle="collapse" data-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation" >
        <span className="navbar-toggler-icon"></span>
      </button>
      <div className="collapse navbar-collapse" id="navbarSupportedContent">
        <ul className="navbar-nav ml-auto">
          <li className="nav-item">
            <NavLink className="nav-link" to="/create">Create Record</NavLink>
          </li>
        </ul>
      </div>
    </nav>
  </div>
);

export default NavBar;

```

Within the **NavBar** component, we will have a navigation bar that links us to the necessary components.

5. Open **record-list.js** and add the following code:

```
import { useEffect, useState } from "react";
import { Link } from "react-router-dom";

const Record = (props) => (
  <tr>
    <td>{props.record.name}</td>
    <td>{props.record.position}</td>
    <td>{props.record.level}</td>
    <td>
      <Link className="btn btn-link" to={` /edit/${props.record._id}`}>Edit</Link> |
      <button className="btn btn-link"
        onClick={() => { props.deleteRecord(props.record._id); }} >Delete</button>
    </td>
  </tr>
);

const RecordList = () => {
  const [records, setRecords] = useState([]);
  // effect to fetch the records from the database
  useEffect(() => {
    const getRecords = async () => {
      const response = await fetch(`http://localhost:5050/record`);
      if (!response.ok) {
        const message = `An error occurred: ${response.statusText}`;
        window.alert(message);
      }
      const records = await response.json();
      setRecords(records);
    }
    getRecords();
  }, [records.length]);
  // method to delete a record
  const deleteRecord = async (id) => {
    await fetch(`http://localhost:5050/record/${id}`, { method: "DELETE" });
    const newRecords = records.filter((el) => el._id !== id);
    setRecords(newRecords);
  }
  // method to map out the records on the table
  const recordList = () => (
    records.map((record) => (
      <Record record={record} deleteRecord={() => deleteRecord(record._id)} key={record._id} />
    ))
  );
};

// display the table with the records of individuals
return (
  <div>
    <h3>Record List</h3>
    <table className="table table-striped" style={{ marginTop: 20 }}>
      <thead>
        <tr>
          <th>Name</th>
          <th>Position</th>
```



```

      <th>Level</th>
      <th>Action</th>
    </tr>
  </thead>
  <tbody>{recordList()}</tbody>
</table>
</div>
);
}

export default RecordList;

```

The provided code will function as a viewing component for our records, retrieving all database entries through a GET method.

6. Replace the code in **App.js** to match the following:

```

import { Route, Routes } from 'react-router-dom';

import NavBar from './components/navbar';
import RecordList from './components/record-list';
import Edit from './components/edit';
import Create from './components/create';

const App = () => (
  <div>
    <NavBar />
    <Routes>
      <Route exact path="/" element={<RecordList />} />
      <Route path="/edit/:id" element={<Edit />} />
      <Route path="/create" element={<Create />} />
    </Routes>
  </div>
);

export default App;

```

SECTION 5. CONNECTING THE FRONT AND BACKENDS

So far, we have successfully created our UI components and established a connection between our React app and the express app we built for our API backend using **fetch**, a built-in function designed to simplify the handling of HTTP requests. This is implemented in the **Create**, **Edit**, and **RecordList** components.

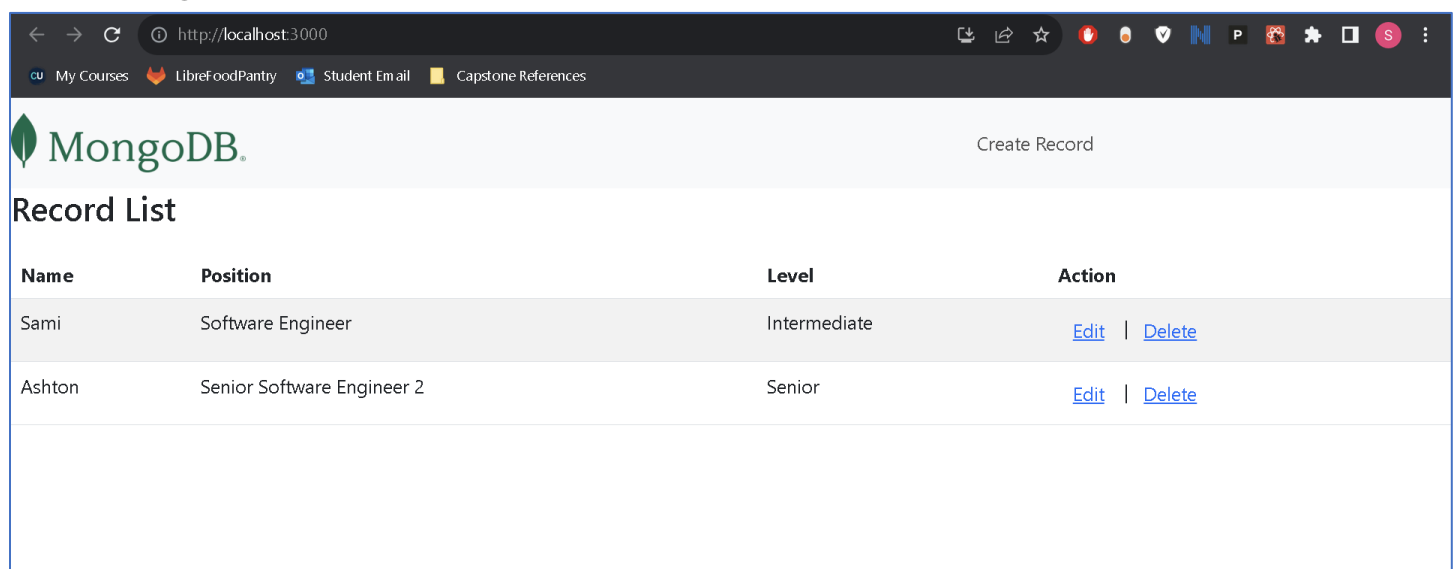
In **create.js**, we enhance the **onSubmit(e)** block with code that triggers a POST request to the **/create** endpoint, adding new records to the database. Similarly, in **edit.js**, we insert code within the **onSubmit(e)** block, as well as beneath the constructor, to make updates to the database. In **record-list.js**, we use the GET method from **fetch** to retrieve records from the database.

The last thing we need to do to connect our client and server side applications is update the placeholder URL in all of the **fetch** functions with the local address of our backend API. Once all of the components have been updated, we need to navigate to the **client** directory in our terminal and run the **npm start** command.

NOTE: *Ensure your express API is running in a separate terminal from your React app. If not, open a new terminal and navigate to the appropriate directory then run the command **node server.mjs**.*

After refreshing your browser, a successful connection between the front and backends of our application, we will be able to view a display of the records in our database, generate new records and perform updates or deletions through the frontend.

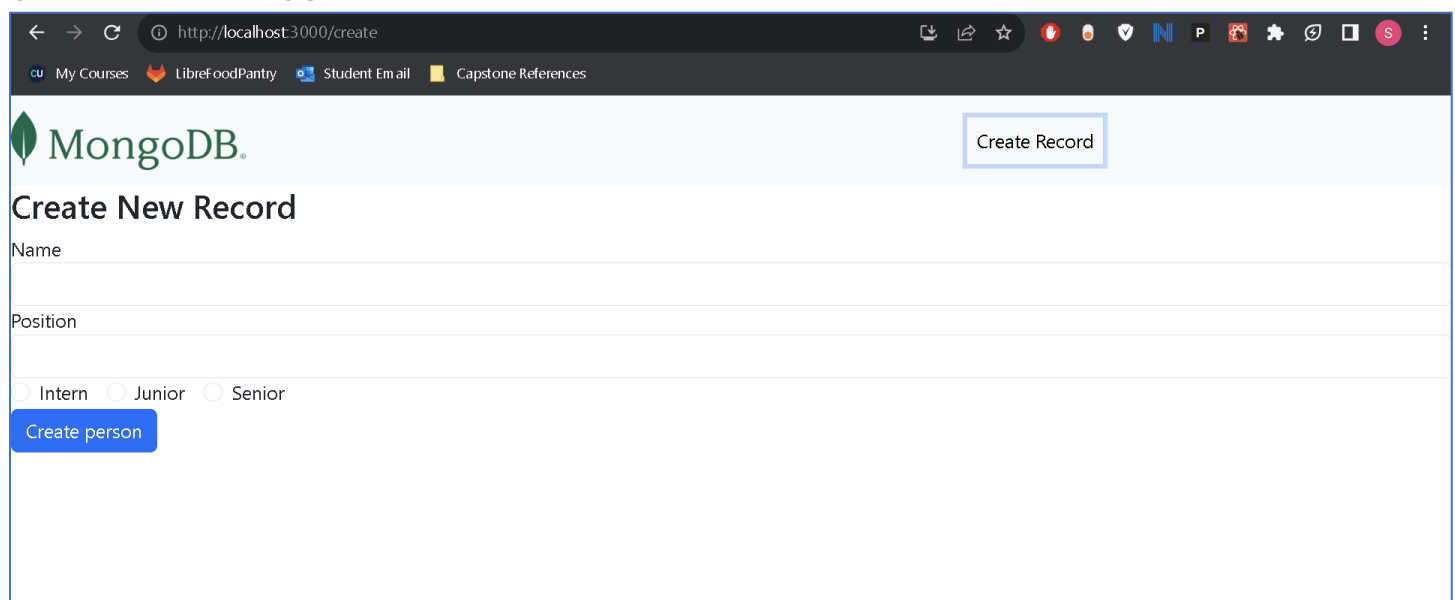
INITIAL LOAD



The screenshot shows a web browser at `http://localhost:3000` displaying the MongoDB Record List page. The page has a dark header with the MongoDB logo and a 'Create Record' link. Below the header, the title 'Record List' is followed by a table with four columns: Name, Position, Level, and Action. The table contains two records: Sami (Software Engineer, Intermediate) and Ashton (Senior Software Engineer 2, Senior). Each record has 'Edit' and 'Delete' links in the Action column.

Name	Position	Level	Action
Sami	Software Engineer	Intermediate	Edit Delete
Ashton	Senior Software Engineer 2	Senior	Edit Delete

CREATE NEW RECORD



The screenshot shows a web browser at `http://localhost:3000/create` displaying the MongoDB Create New Record form. The page has a dark header with the MongoDB logo and a 'Create Record' link. Below the header, the title 'Create New Record' is followed by a form with three input fields: Name, Position, and Level. The Level field has three radio button options: Intern, Junior, and Senior. A blue 'Create person' button is at the bottom of the form.

Create New Record

Name

Position

☐ Intern ☐ Junior ☐ Senior

Create person

NEW RECORD ADDED

← → ↺

http://localhost:3000

📄

🔖

☆

🔥

🍌

🛡️

📺

P

🔌

🔍

🖱️


🌐

⌂

S

⋮

cu My Courses LibreFoodPantry Student Email Capstone References

MongoDB

Create Record

Record List

Name	Position	Level	Action
Sami	Software Engineer	Intermediate	Edit Delete
Ashton	Senior Software Engineer 2	Senior	Edit Delete
New Record	Superman	Junior	Edit Delete

EDIT RECORD

← → ↺

http://localhost:3000/edit/64e308efa27c95eea03ad81b

📄

🔖

☆

🔥

🍌

🛡️

📺

P

🔌

🔍

🖱️


🌐

⌂

S

⋮

cu My Courses LibreFoodPantry Student Email Capstone References

MongoDB

Create Record

Update Record

Name:

Sami

Position:

Software Engineer

☐ Intern

☐ Junior

☐ Senior

Update Record

UPDATED RECORD

← → ↺

http://localhost:3000

📄

🔖

☆

🔥

🍌

🛡️

📺

P

🔌

🔍

🖱️


🌐

⌂

S

⋮

cu My Courses LibreFoodPantry Student Email Capstone References

MongoDB

Create Record


Record List

Name	Position	Level	Action
Sami	Software Engineer	Intern	Edit Delete
Ashton	Senior Software Engineer 2	Senior	Edit Delete
New Record	Superman	Junior	Edit Delete

RECORD DELETED

← → ↺ http://localhost:3000

My Courses LibreFoodPantry Student Em ail Capstone References

MongoDB

Create Record

Record List

Name	Position	Level	Action
Ashton	Senior Software Engineer 2	Senior	Edit Delete
New Record	Superman	Junior	Edit Delete

PUSH YOUR WORK TO GITHUB TO SUBMIT