DOCUMENTATION

PROJECT LIGHTNING

PRODUCED BY TEAM WINNERS

COMPSCI320: INTRODUCTION TO SOFTWARE ENGINEERING

Table of Contents

SECTION 1

Introduction to Project Lightning

Project Lightning is a Minimum Viable Product (MVP) implementation of a RESTful API backed software system developed by a team of students enrolled in UMass Amherst's COMPSCI320: Introduction to Software Engineering in the Fall 2022 semester for ISO New England.

1.1 Rationale

The software system is geared towards addressing two client proposed user cases as under:

- Use Case 1: Users are presented with a dashboard that summarizes time series data in a visually appealing way using charts and graphs.
- Use Case 2: Users are presented with an interactive GUI that facilitates comparison and modeling of time series data.

1.2 System Description

The software system is comprised of the following main components:

- The system encapsulates a REST API that processes user requests and accesses data in a secure manner.
- The system includes a database that stores the data securely and supports operations such as querying and accessing data.

- The system includes an interactive and visually appealing GUI that provides ease of access to the users.

1.3 Downloading Project Lightning

Project Lightning is available open source and is available on all types of operating systems. To download, click the url to the the official github repository: https://github.com/nishant1549/lightning

1.4 Installing Dependencies

A list of dependencies is provided in the package.json file in the code repository, both for the client as well as the server in each of the directories.
Navigate into each of the directories and run the following command:

```
npm install
```

1.5 Build Instructions

The system is built using Create React App framework. Clone the repository on your local system and make sure all dependencies have been installed.
To begin development, run the following command inside the project directory, once each for the server and the client:

```
npm start
```

Open http://localhost:3000 to view it in your browser.

1.6 Contributors

- Atif Abedeen
- Aadit Bhatia
- Colin Genta
- Nishant Jain
- Matt May
- Gabe Sussman
- Manan Talwar
- Alan Zheng

SECTION 2

The Client Directory

The client directory in the code repository contains crucial front-end code files. The important files in this directory with their significance is explained below.

2.1 Home.js

This file contains code functionality to the homepage that is used to route to use cases 1 and 2.

2.2 Filter.js

This file contains code functionality for the filtering page to enable the user to select what specific filters to apply to the nodes as a part of the use case 2 implementation.

2.3 Graph.js

This file contains code functionality to generate graphs and plots from the node data by using the highchart API and display them to the user through the interface as a part of use case 1 implementation.

2.4 Validate.js

This file contains code functionality for the validation page developed as a part of the implementation for use case 1. The code displays 4 plots based on node data with date, LMP, scenario, and name filters, as well as statistics about the data in the plots.

2.5 UC2.js

This file contains code functionality for the page displaying statistics based on nodes filtered by Filter.js as a part of the implementation for use case 2.

SECTION 3

REST API

The section provides an overview of the RESTful API that underlies the backend of the project.

3.1 API Frontend Methods

The following functions are defined in getFromServer.mjs.

1. pullInit
   (a) Input: None
   (b) Output: Promise of object mapping each database field to what datatype is stored for that field.
2. pullNodes
   (a) Input: string representing filters
   (b) Input Description:
      - Starts with ?
      - Each filter is then &FIELD=value
      - Range filters include &FIELD=range before the start and end.
      - Dates formatted as mm/dd/yyyyT00:00:00.000z
      - No range for dates, a single date is converted to start and end with times, while two dates automatically become a range.

   (c) Output: Promise of object containing node objects

       Note: Each node object maps each field of a given node to the associated value

3. aggregateNodes

   (a) Input: A string representing the filters

   (b) Input Description:

       - Starts with ?
       - Each filter is then &FIELD=value
       - Range filters include &FIELD=range before the start and end.
       - Dates formatted as mm/dd/yyyyT00:00:00.000z
       - No range for dates, a single date is converted to start and end with times, while two dates automatically become a range.

   (c) Output: Object of key: field contained in the nodes and value: all the values of the field.

4. get

   (a) Input: String of a field in the database

   (b) Output: Array containing all values of the given field

3.2 API Backend Methods

1. get

   (a) Input: String collections (node table to query), JSON Object Filters (Array indicating that conditions must be met for a node to be returned.)

   (b) Output: Returns an array of JSON Objects related To the filters' JSON Object

3.3 API Backend to Frontend Calls

1. /init

   (a) Purpose: Returns an init object used subsequent computation.

   (b) Return Value: JSON Object formatted as under:

```
{
 Fieldname: val
 RelatedObj/Fieldname : val,
...
}
```

(c) Example:

```
{
 PNODE_NAME: "string",
 Generator/Fuel: "string",
 PERIOD_ID: "date",
 LMP: "number",
 ...
}
```

2. /get or /getoption

   (a) Purpose: Returns an array of unique values corresponding to the requested field.

   (b) Input Formatting: See the examples below.

   (c) Return Value: Array of requested values.

   (d) Example:

```
By fieldname:
    Format: /get/PNODE_NAME ==> [ "NAME1" , "NAME2"]
Relatedobj/fieldname:
    Format: get/Generator/Fuel ==> [ "Oil" , "Solar"]
Relatedobj/Relatedobj/fieldname:
    Format: get/Scenario/Group/fieldname ==> [val, val2]
```

3. /filter?query

   (a) Purpose: Returns results corresponding to a query

   (b) Return Value: An array of nodes and associated fields

   (c) Query Formatting Guidelines: See examples below.

   (d) Example:

```
Every node where the value is equal to the value specified:

a) Exacly one value
Format: /filter?Fieldname=val
Example: /filter?LMP=30 ==> [ { LMP:30, SCENARIO_ID: val, ... } ]

b) One of Multiple Values
Format: Fieldname=val1&Fieldname=val2&...
Example: /filter?LMP=30&LMP=31 ==> [
{ LMP:30, SCENARIO_ID: val, ... },
{ LMP:31, SCENARIO_ID: val, ... },
...
]

c) Range of Values (inclusive)
Format: /filter?Fieldname=range&Fieldname=minval&Fieldname=maxval
Note1 : Any extra fields are ignored.
Note 2: If a maxval is not provided, minval is treated as
one value (1a).
Example: /filter?LMP=range&LMP=30&LMP=31 ==> [
{ LMP:30, SCENARIO_ID: val, ... },
{ LMP:30.2, SCENARIO_ID: val, ... },
{ LMP:30.6, SCENARIO_ID: val, ... },
{ LMP:31, SCENARIO_ID: val, ... },
...
]
```

SECTION 4

PRISMA

PRISMA is an object relational mapping (ORM) software that allows seamless flow of data between different database services by mapping the data into object oriented relational format. Helpful links are linked below:

- For description, check `https://www.prisma.io/`
- For installation and setup, check `https://www.prisma.io/docs/getting-started/quickstart`
- For complete documentation, check `https://www.prisma.io/docs`

4.1 PRISMA Integration with the Server

This section contains step-by-step instructions to set up PRISMA with the backend server.

1. Identify and paste connection url into the .env file.
2. You must now create a PRISMA schema file. The PRISMA Docs have detailed information on creating this file. Visit Prisma DB Schema page (`https://www.prisma.io/docs/concepts/components/prisma-schema`) for help configuring the Prisma schema file. A temporary file has been created at server/prisma/prisma.schema.

   Note: Only small changes will need to be made to convert the current MongoDB schema to transition to a SQL-based schema. An initial schema is being provided with the project.

Note: If necessary the following command may be helpful.

```
npx prisma db pull
```

3. Once the schema file is updated, run the following command to package connection URL (with encryption).

```
npx prisma generate
```

4. Configure Host IP settings in the server/server.js file.
5. Clone the server directory to the Host.
6. Configure "Access-Control-Allow-Origin" in all server side calls. This must be done in 3 main calls: /init, /get, and /filter. See warning below for more information.
7. Run the following command to begin the server.

```
npm start
```

WARNING: Set "Access-Control-Allow-Origin" only for the client host's address for each of the server side API Calls. This will keep anyone but our client application from querying the server. Without configuring this setting you will be opening up querying requests to the internet, opening doors to security vulnerabilities.

4.2 PRISMA Integration with the Client

This section contains step-by-step instructions to set up PRISMA with the frontend client.

1. Configure server IP settings in client/src/getFromServer.mjs, so the application can query server data.
2. Select a subdomain to host the application.
3. Modify the package.json scripting and the .env file that configures the clients' Host information. For a guide, reference React's Deployment information (`https://create-react-app.dev/docs/deployment/`).
4. Run the deployment scripts.