record-log
Software Design Specification

1. SDS Revision History

Date	Author	Description
5/13/24	EH	Initial document creation
5/13/24	JO	Modifications to section 4. Software Modules
5/19/24	EH	Updates made for new project idea
5/20/24	EE	Added UML diagrams and revised module language
	EE	Refactored to update naming
6/1/24	JO	Updating SDS for final submission
6/2/24	EE	Refreshed diagrams, module names, and module descriptions
6/3/24	EE	Finished updating descriptions

2. System Overview

The record-log system architecture is organized into four main categories: the User Logging Interface Module, Entry Views Module, Entry Database Module, and Database / Server Interaction Module . Utilizing the MERN (MongoDB, Express.js, React, and Node.js) technology stack eased the development of a highly modular full-stack application. The User Logging Interface and Entry View Modules are written in React, with a stateful, ‘component-based’ design philosophy that makes for highly reusable parts in the user interface. The database server integration module utilizes Node and Express for handling HTTP GET, PUT, POST, and DELETE requests between the Entry Views and the Entry Database. The Entry

Database uses MongoDB to store organized data about song entries that are logged by the user.

3. Software Architecture

Overview

The architecture of record-log is designed to utilize the MERN stack in a way that facilitates a natural flow between the client and server side operations. The application developers and maintainers should be provided with highly cohesive, loosely coupled modules, and the MERN stack should create a robust and scalable web application.

User Logging Interface Module

The User Logging Interface Module provides an area for user navigation and a place in the application for all view routes to have an area to be displayed. This is the primary method of interaction between application and user.

Entry Views Module

The Entry View and Logging Interface will be what the user interacts with when using record-log. It will display the software modules and send and receive API calls to and from the Express.js server intermediary.

Entry Database Module

This module consists of the MongoDB server, which holds collections of song entries in a formatted document-style manner. This interfaces with the interaction module to send and receive information for persistent storage.

Database and Server Interaction Module

This module will handle the general logic for database operations and client-server communication. They will send and receive data to and from MongoDB in the entry database. The Entry Views module will use this module's methods to populate components that will then be displayed via the User Logging Interface Module.

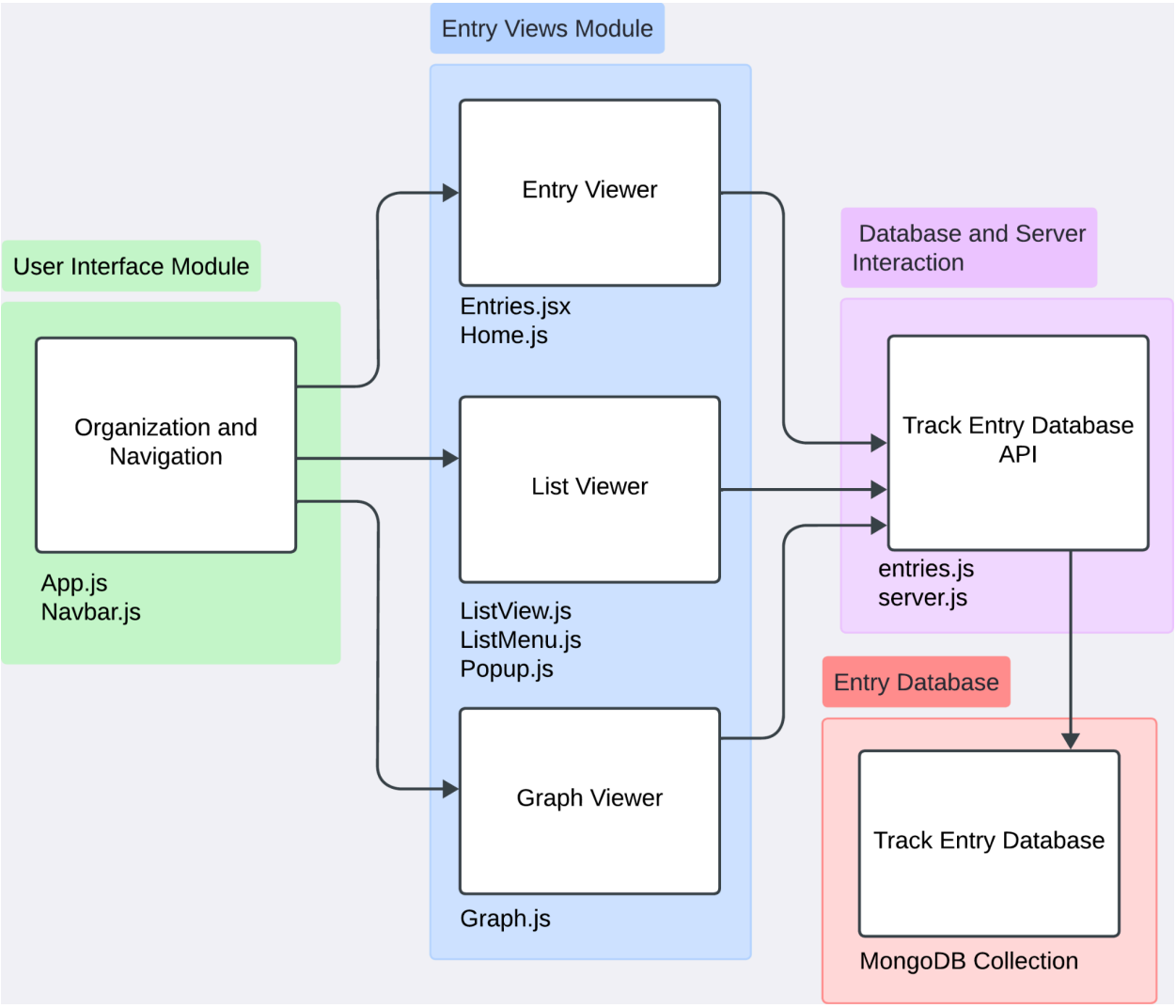

Figure 3.1 Software Architecture Diagram: This figure shows the abstracted layout of the system modules, with the User Interface Module interfacing with all three view formatting components in the Entry Views module, the view components interfacing with the Database and Server Interaction module, and the API in the interaction module depending on the Entry Database module.

4. Software Modules

4.1 User Interface Interaction Module

4.1.1 Module Role and Primary Function:

The User Interface Interaction module provides a final endpoint for the application user to interface with the rest of the program. App.js and Navbar.js, the two components within the User Interface Module, provide the user with the navigation options to swap between views in the <Navbar /> component, as well as the route for each view to be seen in the browser in the <App /> component.

4.1.2 Interface Specification:

This module interfaces with the Entry Views Module, as each view provides the User Interface module with a single method {Home(), ListView, Graph()} that returns the fully populated component to be rendered.

4.1.3 Static Model

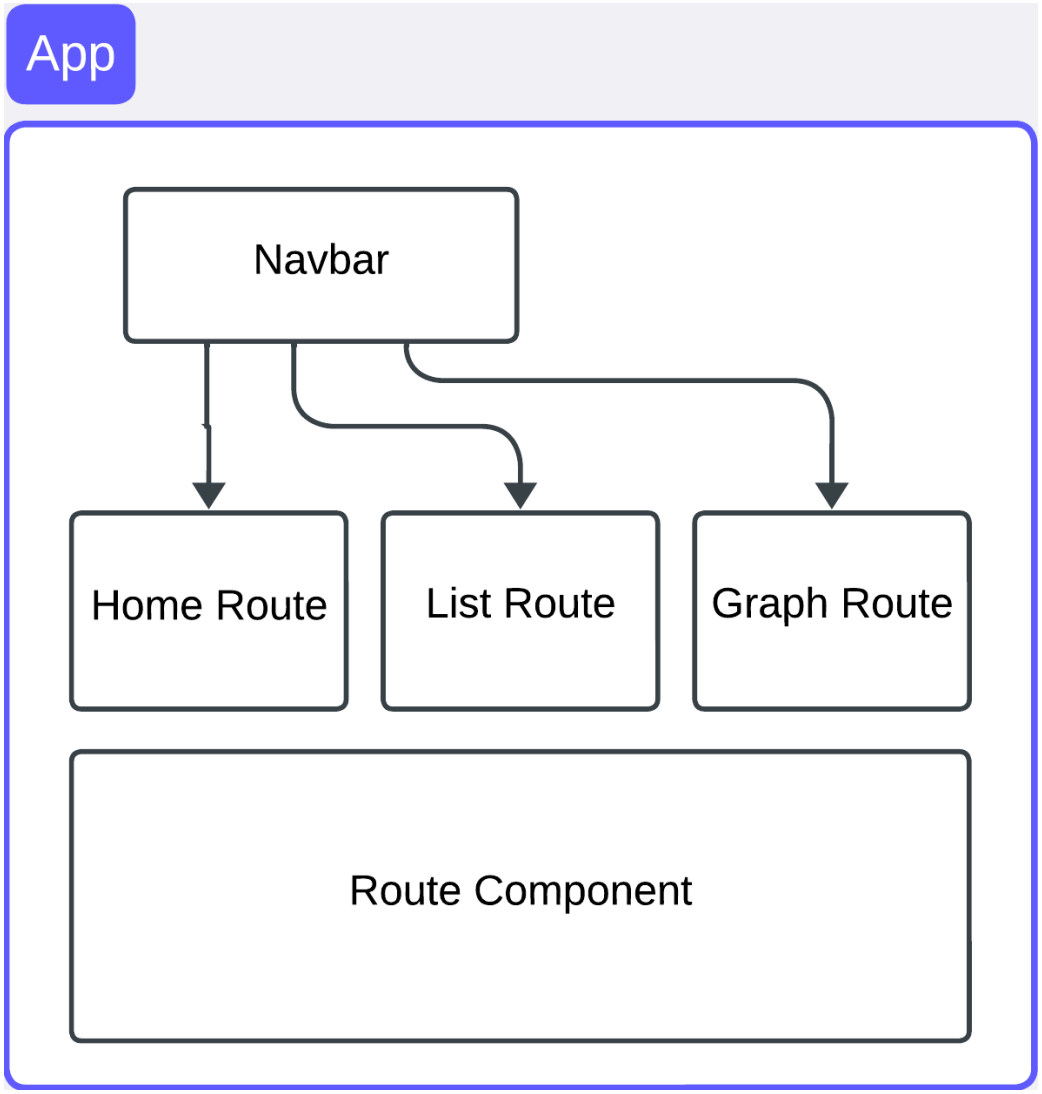

Fig. 4.1 Static Model: This diagram represents the structure of the User Interface Module, with the Navbar containing references to each view mode route, and the component supplementing the route with its view.

4.1.4 Design Rationale

This module was designed to contain both the navigation and view in order to motivate modularity of the different view options. Due to very low levels of coupling between the User Interface Module and the Entry Views, new views can easily be added and made accessible by the User Interface Module.

4.1.5 Alternative Designs

Initially, the User Logging Interface module would have included every data view component--as if the ‘route’ components within Figure 4.1 were instead fully fledged components embedded into the module. We found that this increased render time for each page and slowed performance, and it limited the modularity of the system. Redesigning the User Logging Interface module allowed for easy implementation of the AboutText.js file, as a new route was quickly added to the interface and navigation bar.

4.2 Entry Views Module

4.2.1 Module Role and Primary Function:

The Entry Views Module contains the formats in which the data in the Entry Database can be displayed via the User Logging Interface. The module provides methods for each of the given views to be used by the UI.

4.2.2 Interface Specification:

The Entry Views Module interfaces with the Database and Server Interaction Module (D/SIM) with the transfer of entry information such as an entry_id, entryName, thought, etc., from the Entry Database to the Entry View’s internal data structures. Specifically, the Entry Views Module uses route.route(‘/’) to fetch all entries from the database, route.route(‘/add’) to create new entries, route.route(‘/:id’).get, .delete, and route.route(‘update/:id’) to fetch, delete, and update entries in the database, all methods provided by the D/SIM.
The Entry Views Module interfaces with the Interaction Module to make sure the lists are appropriately displayed in the user interface with their correct entries and information. The module provides each view as a publicly exported function that is accessed in the form of a Component (Home.js exports const function Home() → <Home />, ListView.js → <ListView />, Graph.js → <Graph />).

4.2.3 Static Model:

See Figure 4.2. The Entry Views module is composed of Graph.js, Home.js, Entry.jsx, ListView.js, ListMenu.js, and Popup.js.

4.2.4 Design Rationale

This module was designed to manage the individual representations of the user’s data while minimizing *direct* interaction with the database. The view components simply extract or send data in organized

formats (JSON or Mongoose Schema, respectively), whilst the more complex parsing and retrieval is done by the Database and Server Interaction Module.

4.2.5 Alternative Designs

The Entry Views Module was, as mentioned in section 4.1.5, initially a part of the User Interface Module. This design was deprecated in favor of the separate, cohesive module that contains the individual views.

4.3 Entry Database Module

4.3.1 Module Role and Primary Function:

This module handles the persistent storage of track entry data on the MongoDB Atlas server. Being a largely third-party implemented module, the database module provides two methods that are used for connection to itself and formatting of the data that is sent to it.

4.3.2 Interface Specification:

This module interfaces with the Database and Server Interaction Module, which provides assistance with database management methods. The module provides the mongoose.connect() and mongoose.Schema methods, which are utilized by the D/SIM to query data out of the Entry Database.

4.3.3 Design Rationale

This module was designed to be as simple as possible while reliably providing the desired functionality. This module is coupled with the Entry Database API module which allows the transfer of data from the User Navigation module to the database.

4.3.4 Alternative Designs

Initially, this database was going to be self hosted on a University of Oregon development server, ix or ix-dev. Due to some system limitations, this wasn't found to be reliable for the purpose of this project. As a result, the application now connects to MongoDB Atlas to store user data.

4.4 Database and Server Interaction Module

4.4.1 Module Role and Primary Function:

This module is intended to facilitate data transfer between the Entry Views module and the Entry Database module. It consists of externally available API functionality, composed of the following database interaction methods: add, delete, save, and update.

4.4.2 Interface Specification:

This module interfaces with the User Logging Interface module and the Entry Database module. The module uses Mongoose, an ORM (Object Relational Mapper) to make requests to send and receive data from the Entry Database Module. The module also provides the User Logging Interface with HTTP

methods to send and receive data to the Interaction Module for extended interfacing with the Entry Database. As detailed in section 4.2.2, the Database and Server Interaction Module provides the routes (‘/’, ‘/add’, ‘/delete-lists/:id’, ‘/update/:id’, ‘/:id’) for the Entry Views Module to access data. This module connects to the Entry Database module via MongoConnect, and uses the Mongoose Schema specified by the route model to pull information out of the database to populate these route requests.

4.4.3 Design Rationale:

This module was designed to reduce the complexity of DB queries by the Entry Views Module. The module’s API provides easily intelligible and usable methods of handling data with a limited amount of coupling. Providing the Entry Views with an API for accessing data eased the process of development, and reduced technical overhead that would have slowed development if every view was required to directly access the database server.

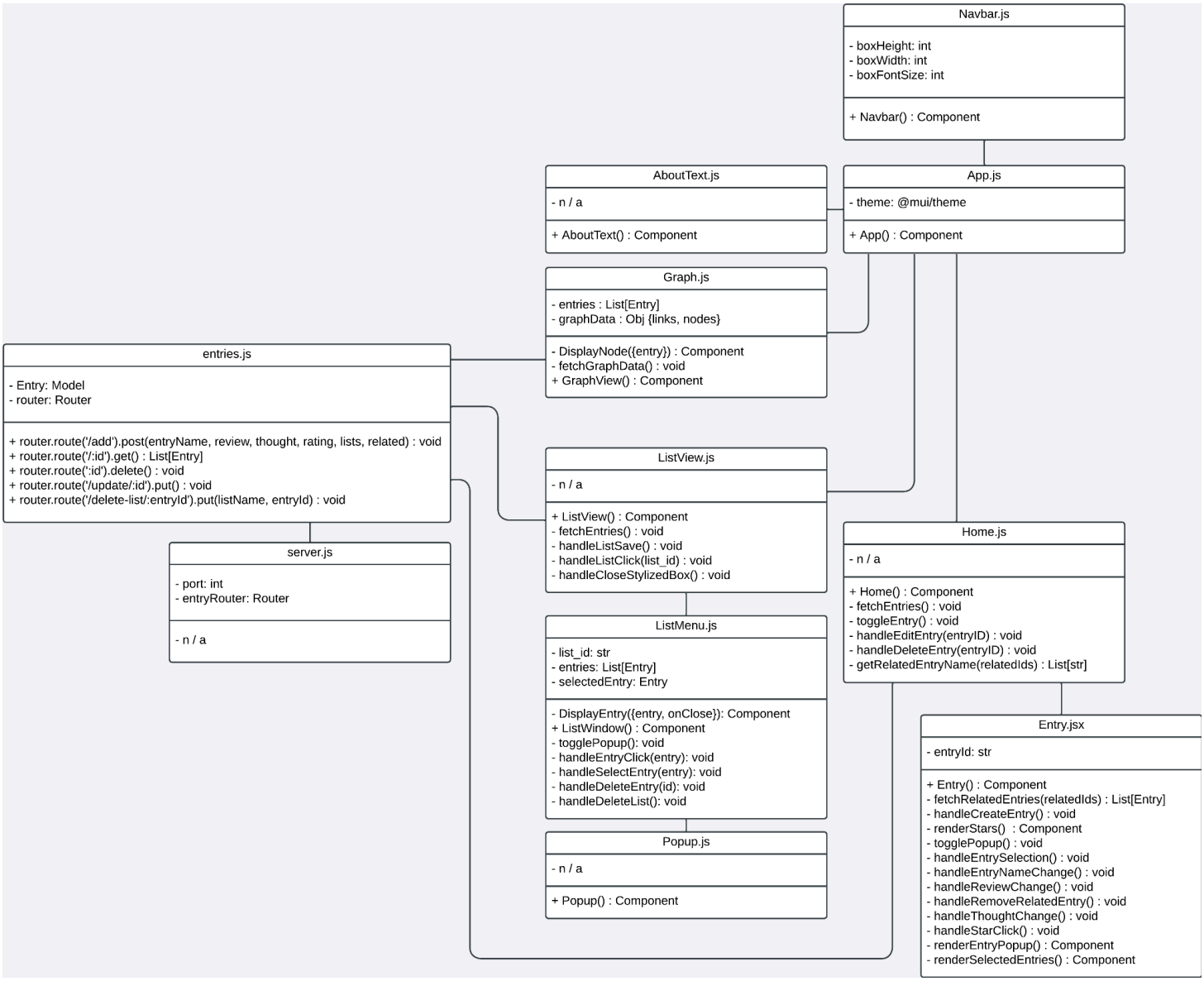

Fig 4.2 System Class Diagram: This diagram shows the relations between each class in the system, with the edges representing interfacing between classes.

5. Dynamic Models of Operational Scenarios (Use Cases)

5.1 Software Use Cases

Song Review: After running the software, Users can create entries to organize and review different musical pieces they have listened to. They can give their thoughts on the song, as well as save specific information and thoughts about the film, such as BPM,

songs that may transition well to another, or a note of the song's Genre. After creating entries, the user can create specific lists to help organize their entries. A list can save entries that come from the same genre or have similar characteristics.

Playlist Creation: After logging many tracks and taking note of meaningful relations between songs, the user can visit the Graph View to gain inspiration for a path between a number of songs. They can keep this path in mind when they visit List View, then create a new list, and add the songs along that path.

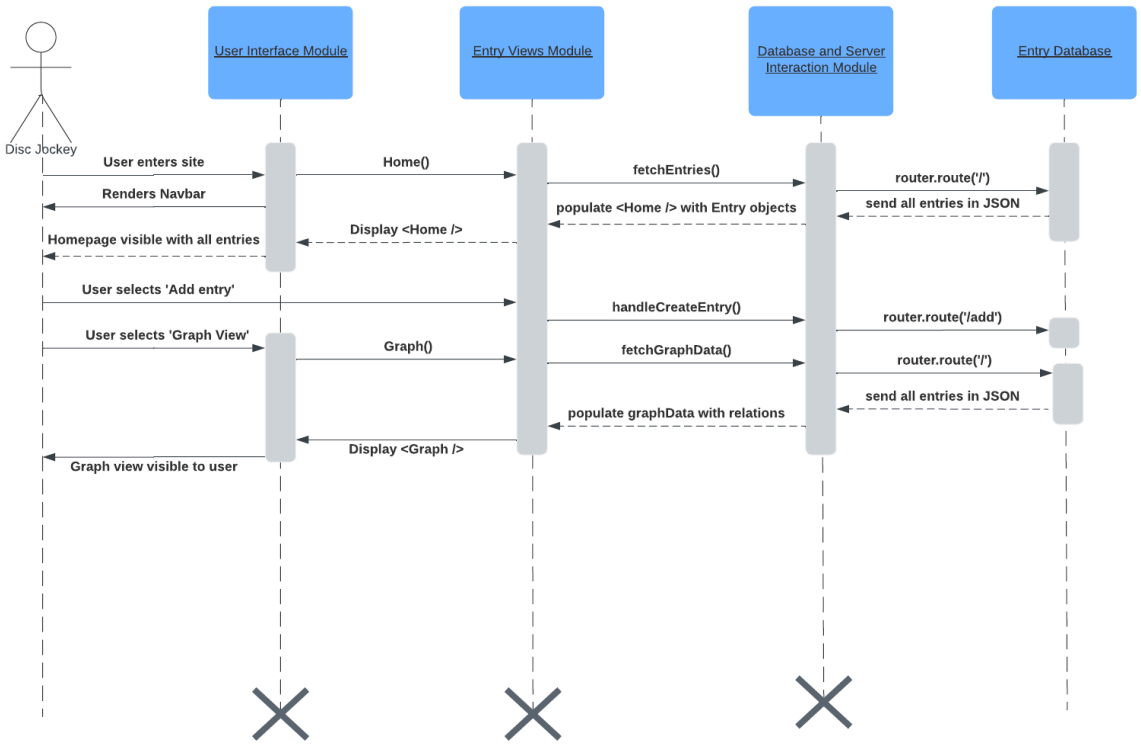

Fig 5.1 Sequence Diagram: This is a dynamic model of a disc jockey that has listened to a new song and wants to log it on record-log. First, they enter the site. This prompts the User Interface Module to request the Entry Views Module for a <Home /> component. To populate the component with all the entries in the database, the module calls fetchEntries() to get the entries in a JSON (JavaScript Object Notation) format. The server interaction module sends these to the view module with organized Entry objects, which are then used to populate the Home view. Finally, the user interface module displays this populated component to the user. Next, the user selects 'Add entry' on the home page. This immediately prompts the Entry Views module to make a call to handleCreateEntry, which then prompts the database interaction module to send a router.route('/add') PUT request to the database. Finally, the user selects 'Graph view,' which prompts for Graph(), which calls fetchGraphData, which then makes a get request to the database, and the information cascades back to the user.

6. References

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young.
IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255
Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM, 15*(12), 1053-1058.

7. Acknowledgements

This software design specification builds on the template from https://classes.cs.uoregon.edu/24S/cs422/Templates.html