InSpire - Adding a Web Server

Stephen Palagi - Kevin Chan - Nicholas Merlino - Michael Gallant - James Morris - Josh Silvia

Routes

HTTP route: GET /students/:studentid/enrolledCourses

- What it does: Takes in a student's id and returns an array of the currently enrolled courses sorted by date, then start time
- What mock server function it replaces: getCourseObjects
- What resources it operates on: students, and within that student his/her courses
- Who is authorized to see this request: The logged in student can only see his/her own account

HTTP route: GET /students/:studentid

- What it does: Takes in a student's id and returns the their information
- What mock server function it replaces: getStudentInfo
- What resource it operates on: The 'students' collection
- Who is authorized to see this request: Only the logged in student

HTTP route: GET /professor/:professorid

- What it does: Takes in a professor's id and returns their information
- What mock server function it replaces: getProfessorInfo
- What resources it operates on: The 'professor' collection
- Who is authorized to see this request: Any logged on student can trigger a request to get professor information via looking at a class

HTTP route: POST /search

- What it does: Takes a JSON object from the body and uses its properties to search for courses that match what the user entered.
- What mock server function it replaces: getSearchResults
- What resources it operates on: courses
- Who is authorized to see this request:

HTTP route: POST /addclass

- What it does: Enrolls a student in a class given the student ID and course ID in the URL
- What mock server function it replaces: enrollInClass
- What resources it operates on: courses, students
- Who is authorized to see this request: Only the student being enrolled in the class

HTTP route: POST /dropclass

- What it does: Removes a student from a class given the student ID and course ID in the URL
- What mock server function it replaces: dropClass
- What resources it operates on: courses, students
- Who is authorized to see this request: Only the student being dropped from the class

HTTP route: POST /resetdb

- What it does: Resets the server database back to it's 'initial state' or 'initialData' and prompts the browser for a page refresh
- What mock server function it replaces: resetdb
- What resources it operates on: 'initialData'
- Who is authorized to see this request: The admin should be the only authorized user to reset the database. But for the purpose of this submission, the 'main user' *Jason Jackson* is the only authorized user we have currently, so their account is able to see this request.

Client Error Handling

Failed HTTP requests such as network failures and client/server errors are handled by displaying a bootstrap warning alert with a small amount of information related to the error. Currently if more than one error occurs then the latter one is the only one displayed by the warning banner. If no Error is present then the banner will be hidden, otherwise it will appear at the top of the webpage.

JSON Tokens and Authentication

For the purpose of this submission the JSON web token authentication is done with an unencrypted token that the client provides for permission. In the future this will be changed to an encrypted token. The current token is created by stringifying the JSON element for the user id and then use base64 encoding on the string to place it in the header.

Client Reset Database Button

The client reset database button communicates with the server to reset the database back to it's initial state. There is an HTTP route on the server that resets the database. We included the reset database button to appear as a link in the user info panel. Once the button is clicked, the browser prompts the user that the database was reset and a page refresh is necessary for changes to take place.

Contributions

Kevin Chan - Helped with Unofficial Transcript modal, implemented getShoppingCart() and getAvailableCourses() http routes, various styling changes, helped wire the calendarBlocks to server with Stephen, refactored the functionality of the AvailableModal so it used the modal template/factory class, wrote the functions in util.js
Stephan Palagi - Unofficial Transcript modal, Added functionality to shopping cart, implemented http routes for getCourseInfo() and getProfessorInfo()
James Morris - Set up initial server code. Programmed search function, adding and dropping classes.
Nick Merlino - Created the basic modal factory, and then created the Final Exam Schedule modal spawned by the modal factory with the route for getCourseObjects() and authorization which it depends on. Removed sidenav and important notices, and created “User info” to handle their old functionality. Created the “logout” button using react-router to a blank page. Various styling changes. Minor database changes.
Joshua Silvia - Implemented reset database button. Set up initial client code. Various styling changes.
Michael Gallant - Error Exception Handling and JSON web token authorization. Added hide element function to util.js as well as minor database changes.

Lingering Bugs / Issues / Dropped Features

- Navbar (scrapped)
- Important Notices (replaced with UserInfo pane)
- Currently have backend logic for adding and dropping classes, but overall structure of component hierarchy is preventing sensible execution of the code. Requires significant refactoring.
- Login/logout is unimplemented, as not required for this submission, but for now we have a link to open up a new page to demonstrate that there will be some action