Spaceacademy Project Description

This project is a REST API built using Express.js as the back end framework, Node.js as a JavaScript runtime environment, and MySQL as the database management system. The front end is made simply using HTML, CSS, and a few scripts using JavaScript.

The premise of this website is that it’s used to keep track of the courses at a fictional “Space Academy” and the different personnel involved with those courses (and their details). Thus, the database contains tables for students, courses, professors, and departments, which each have their own attributes. Specifically, there are two tables called “course enrollments” and “professor assignments” which keep track of which students are registered to which courses, and which professors are to teach which courses, respectively. These are intersection tables, meaning they facilitate the many-to-many relationships between the respective entities that these tables capture. For example, in the case of course enrollments, one student can enroll in many courses, and a course can have many students enrolled in it.

For the majority of the tables in the database, there are required relationships, meaning that the foreign key for a table cannot be null and must have an associated value. However, there is one “NULLable” relationship, which is that courses are not required to have a department (i.e. it can be set to null). The database also has cascade operations, which means that when an entity is deleted, all other rows in other tables that referenced this entity are also deleted (if specified).

With regards to normalization, this database is in second normal form. It reaches first normal form because no table column has tables as values (i.e., each table represents a single entity or idea). It reaches second normal form because it does not contain any partial dependencies (each table contains its own unique primary key). I am not sure if this database is in third normal form (3NF).

The web application is connected to the database using the “mysql2” library in Node. I have a “pool” object (mysql.createPool) that creates a connection to the database, and that pool object is exported to other files in the project so that the connection can be utilized for querying the database.

In terms of general structure of the web application, there is a central app.js file that imports the other components of the database and essentially “runs the web application” and listens on port 8081. The HTTP request methods are split up into several route files (.js) that are used by app.js. These HTTP requests serve up HTML files, or rather .hbs files, which is the format I needed to use because I am using handlebars, a JavaScript templating engine. Handlebars generates HTML pages using a central template and allows for directly inserting back end code and values onto these HTML pages with ease.

In terms of scope, this website has a page for each of the six tables in the database and allows users to do CRUD operations for each of these tables (create, read, update, and delete). These operations are facilitated by their corresponding HTTP request methods, i.e. POST, GET, PUT, and DELETE, respectively. To use the “students” entity as an example:

* Users can create a student by filling out a web form with the pertinent information; upon submission, a POST method is called that creates a student in the database and then sends back the updated web page.
* Users can “read” all the students in the database by simply going to the URL for the students page, which triggers a GET method that serves up the page with all of the rows from the students table.
* Users can update a student by clicking the update link in the row of the students table, which first autofills that student’s information into a web form (which the user can then edit). Upon submission, a PUT method is called to update that database entry and the webpage itself (using AJAX; the page doesn’t refresh in this case).
* Users can delete a student by clicking the delete link in the row of the students table. After confirming, a DELETE method is called to delete that database entry and then refresh the page.

This website’s information is sanitized using a few methods on both the front end and back end. On the front-end, there are various data validation points, such as using regex to prevent users from entering in strings that include quotation marks (“, ‘) and having number fields that have minimums and maximums. All fields are required as well, so no entry will have an insufficient amount of data. On the back end, to further prevent attacks such as a SQL injection, all query strings are constructed using placeholders with the question mark format (?) instead of template literals (i.e. “This is an example ${myVariable}!”). This ensures that any input that has made its way to the back end is going to be treated like a string, instead of allowing a user to close off or create additional SQL queries.

Ideally, I would have liked to have made it so “multiple statements” for SQL queries were disabled, but I have a method that resets the contents of the database to its original, default values that makes use of multiple SQL queries at once (to delete and re-create the entire database).

Finally, I deployed this web application using Amazon Web Service, specifically their two services called Relational Database Service (RDS) and Elastic Beanstalk. The former is a service that simply hosts the MySQL database that this web application uses. The latter is an “orchestration service” that uses several other services for hosting this website (i.e., EC2, S3, autoscaling, which respectively is the virtual computer running the web server, storage space, autoscaling with traffic). All I had to do was update the MySQL connection in web application so that it pointed to the URL of the database (and I provided the proper username and password for accessing this database). The database goes up, then the web application goes up, the web application could find the database, then everything works!