Prototype Findings

Overall Findings

Through the prototyping process, we found that the structure of the prototype models that we made should be sufficient to use for the production codebase in the next sprint. The structure of a web app using a React JavaScript client communicating with an Express JavaScript server through HTTP requests worked well. It allowed us to relegate database communication entirely to the server app, allowing the client and server to be more separate outside of the requests made to each other.

We also made a separate prototype LTI application that used an Express JavaScript server. This app did not have a significant amount of structure, but prototyping it helped us to learn how to successfully send and receive an LTI launch request from Canvas, which we can use in future sprints with the actual production application.

React JavaScript worked well for the client, as we were able to use the Blueprint.js library to standardize the look of the prototype component we made, and it will allow us to keep the appearances of components uniform and integrate other React libraries in the future if needed.

Using Express JavaScript for the server also worked well. We were able to use the “pg” library to interface with our PostgreSQL database and make queries. We were also able to use the “ims-lti” library to process LTI launch requests from Canvas in a separate Express application, which means that Express should work well for the server application in the eventual production environment.

Client Summary

The prototype of the client mainly consists of the App.tsx and multipleChoice.tsx files. A React JavaScript program that communicates with the server and runs on localhost port 3000 is the main program for the client. At start-up, it makes a GET request to the server requesting the data needed to render the multiple-choice question, and it does this through the useEffect function when the component is initially rendered. Once it receives the response from the server, it will then update the component to reflect the data retrieved from the database.

When the user clicks an option and hits the Submit button, the client sends the user’s selection to the database through a POST request, and the server will insert the value into the database. The server will then send back the user’s response, and the client will display the info from the response on the page. We found that this overall flow worked well for the client, and we plan on continuing with this structure in the production codebase next sprint.

Server Summary

The prototype of the server is an Express JavaScript application that performs request routing, resource fetching, and database interaction. The server by default runs on localhost on port 9000. The program contains two main routes currently for requests. A GET request to the “/api/questions” endpoint will return the questions for an exam as well as the available answers to those questions if there are any. For prototyping purposes, there is only one question in the database, but the production grade application will support pulling all questions for an exam from the database.

A POST request to "/api" occurs when the user on the front end submits their answer for the multiple-choice question. When the server receives the request, adds a row to the "StudentResponse" table of the database. We found that this worked well for inserting response data, and we are planning on using this general structure in the initial production codebase next sprint. As of this sprint, the feature does not support recognizing different users, questions, exams, or courses, so the SQL statement inserting the row will remain the same aside from the answer selected by the front-end user.

LTI Summary

The simple LTI application we made was just to get familiar with the flow of LTI launch requests from Canvas and how to use the "ims-lti" library to handle them. We were able to successfully create a simple "Hello World" LTI application and have it display in a Canvas assignment. With this knowledge, we should be able to integrate the real application with LTI standards and have it display within an assignment once the production codebase is established.

Database Summary

The database structure that we chose should work for the application as we build it out further. We were able to insert rows into the table and submit a test response, although the data was hard coded. We may need to add an "Organization" table to the database to keep track of information regarding different LTI provider institutions, but that will likely be determined once the application's features are more fleshed out.