Design Document

Abstract

The annual Spring Programming Contest hosted by Bucknell has a costly amount of manual work involved for judges and participants alike. Because the judges have to go through the timely process of downloading, compiling and testing every submission, Bucknell is forced to limit the amount of teams that can participate. In addition, the scores are computed by hand and displayed on a whiteboard in the classroom. Our goal with this project was to give Bucknell a convenient and well-designed platform that automates these tasks and allows the organizer to communicate with the participants. As a result, judges will have an easier time running the contest and more teams will be able to participate.

Table of Contents

System Design

MERN Stack

The website is built on top of the MERN stack using the MERN boilerplate. The stack breaks down into the following:

- MongoDB - A noSQL database
- ExpressJS - Backend framework for routing and redirecting
- React - A frontend framework developed by Facebook which allows us to split up our interface into modular components that are only rendered when necessary
- Node.js - Event-driven I/O server-side JavaScript environment.

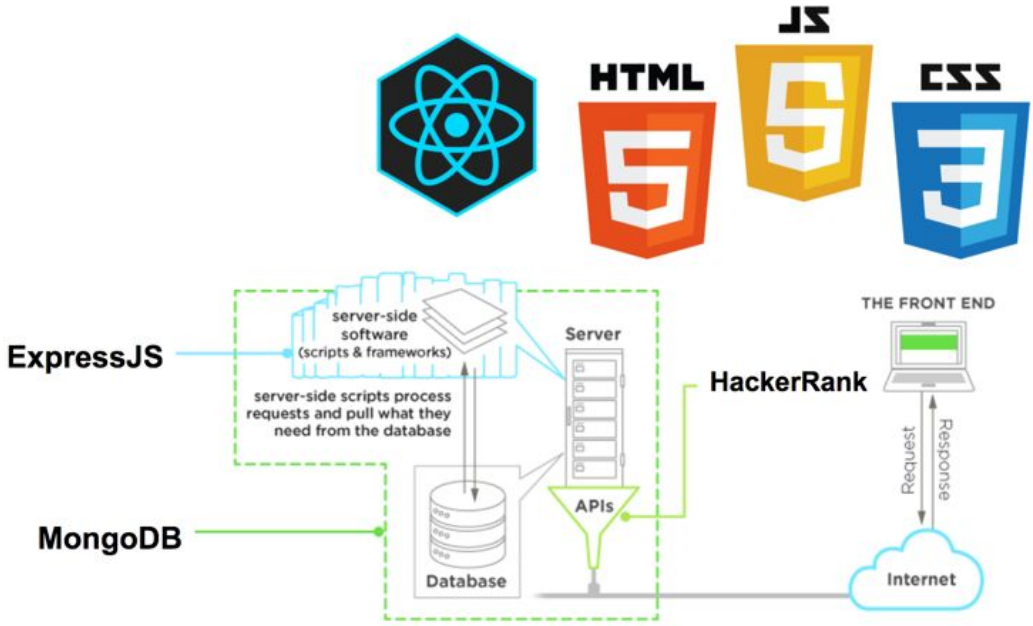

Figure 1: System Design Diagram

There were a couple of considerations that went into our decision to choose the MERN stack. First off, we wanted a web-based solution so that any number of participants, regardless of their location, could join a contest. We decided to use the MERN stack because the entire stack uses JavaScript. Firstly, as full stack developers, we would have an easier time contributing to the project at any point of the stack (frontend/backend/database) since everything was in the same language. If we had chosen Ruby On Rails as our solution, we would need to context-switch from JavaScript in the frontend to Ruby in the backend. This was not an issue with the MERN stack. Additionally, JavaScript is the world's most popular language (according

to Stackoverflow/GitHub usage and frequency). This popularity is reflected by the large online JavaScript community, which has a lot of resources and tutorials that we were able to use.

External APIs and Libraries

Besides the stack frameworks, our website relies on several existing APIs and libraries. The website's biggest dependency is the HackerRank API. It helped us provide the functionality for participants to submit source code, which is compiled and run against a list of contest creator's provided input test cases. We chose HackerRank API because it helps us accomplish our highest priority user story of automating the compilation and testing of submissions. Without any additional work, HackerRank API supports about 50 languages and also provides useful feedback when errors occur.

As the name suggests, the API is provided by HackerRank, a company used for programming interviews and practice coding challenges. Early on in the semester, when we were considering system design, we assessed the risks associated with using an external API for a critical functionality. Since large companies like Amazon and Facebook have been using HackerRank in the hiring of programmers for many years now, we do not expect the endpoint going away. Yet, in the future, it is possible for the API to change in which case we would have to change our code as well. Despite these risks, we think the time saved using HackerRank API along with the benefit of its extensive language support were too valuable to pass up. If HackerRank is no longer available there are many other alternatives. A simple Google search of "compile code API" will yield a good number of alternatives to HackerRank API. As a last resort, one would have to replicate the API's functionality and send the code to our server, compile and run it. There would be a lot of effort involved in assessing incorrect submissions that Hackerrank does for us and the speed of our server would be a greater concern.

In addition to HackerRank API, we used several libraries for different functionalities of the website. We used CodeMirror for the text editor on the problem page because it provides the capability of syntax highlighting for different languages - a feature that ties in nicely with HackerRank's multiple language offering and is highly valuable for participants' experience. We used simple-react-pdf, to display PDFs for both the participant and the judge to view problem prompts. In order to display a clean and elegant scoreboard, we used chart.js. Additionally, on the website, you will find multiple pages that have beautiful tables, which were created using the react-table component.

For security measures, we used bcrypt for hashing user's passwords, so we are not simply storing the raw passwords into our database. We also used jsonwebtoken and jwt-decode, an open, industry standard RFC 7519 method, for user verification between the server and the client. Additional libraries, react components, and dependencies can be found in the package.json file.

Database Design

User Document

The user document contains basic information about the user, such as the username and password, which is inserted when a user first registers on the website. The document also contains the field for capturing ids of contests that the user has created, createdContestsID, and pairs of ids of contests and teams that the user has participated in, participatedContestsID. In addition, the document also contains several flags such as isFirstTimeUser, for telling whether or not they should be displayed initial website greetings, loginAttempts, and lockUntil.

Contest Document

The contest document contains an entry for each contest that is created. Each contest will contain a list of team objects where all information relevant to the teams is kept. The contest object also has a list of problem subdocuments that contain problem names and their corresponding file names. The actual input, output and PDFs for problems are downloaded to folders on the server. Most operations on the contest document happen from the contest controller, except for messaging which is handled by the messaging controller and a few updates that result from a change when a judge gives feedback.

Submission Document

The submission document is the source of information for the judge's table and the individual submissions pages that can be accessed by selecting one of the submissions in the table. Submissions are created in the contest controller, but all other submission operations, feedback changes, deleting submissions, etc. are called from the submissions controller.

Chat

Since we did not have a lot of time and we were unable to integrate socket.io into our project, our implementation uses polling for handling team and judge messages. When the team is on the problem page, it will send a request to the server about every 10 seconds to receive all new messages for the team. This implementation seems to be working fine, however to optimize the performance it would be better for the team to send the number of messages it has to the server with its request for messages so the server only has to send over new messages the team does not have. The judge of a contest will poll about every five seconds to check if it has a new message. If the judge has a new message, the judge will see a notification saying such when it is not on the submissions page. Clicking on the notification will bring the judge to the submissions page

where they can see messages from the teams using the chat sidebar. Clicking on a team in the sidebar opens up a small chat window that works just as the team chat works.

Authentication

The figure below gives a basic overview of how the authentication for the website works. When the user clicks login on the website, a post request is made to the backend api /auth with the username and password. On server side, a json web token (JWT) is created, and then returned to the browser. All the routes from now on are verified by keeping the JWT in the user's browser's local storage. In addition, every time a user navigates to a new page, the jwt is sent in the authorization header. The server should check the jwt signature, get the user information and send the user to the appropriate page. Currently, this step of checking the authorization header upon every request is not done, and therefore the api endpoints are open. We encourage the next person who carries this project forward to complete this.

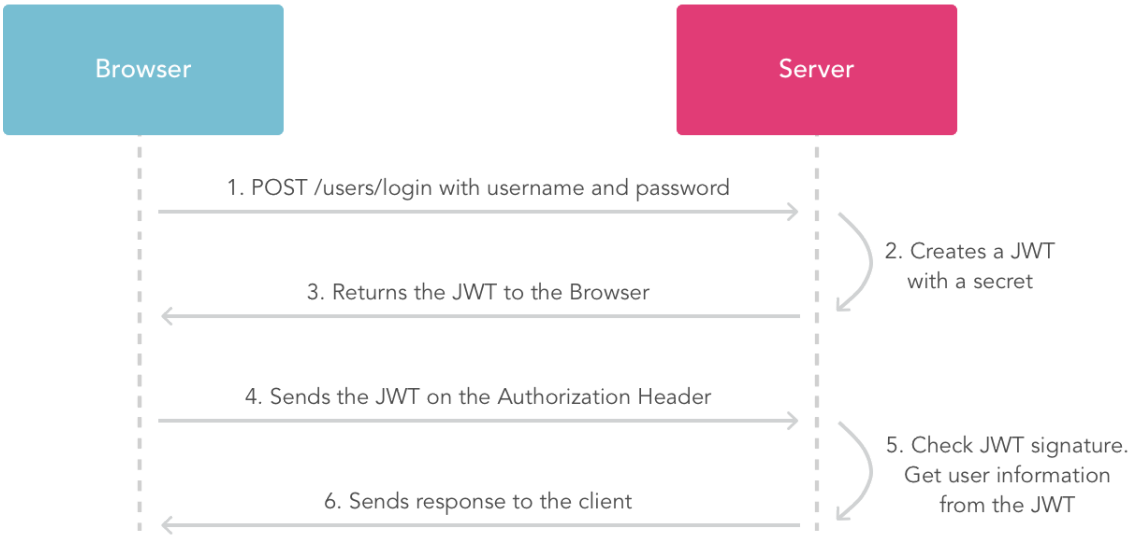

Figure 2: Authentication Architecture

Testing

There is a severe lack of testing in this project. There are a few tests written using Ava for some of the backend controllers, but no testing exists for the frontend. Ideally every function should have at least one test, but we had some trouble running the tests. Only one of our group members was able to get the test running on his laptop (OS windows 10, node 6.9). We could not solve the issue, but encourage anyone continuing this project to please add more test coverage and fix the the issue that makes npm crash during testing.

Scrum

These are the major user stories we had set out to accomplish at the beginning of the year per request of our client, Professor Wittie:

- Completed
    - Allow participants to submit code from multiple operating systems
    - Maintain a scoreboard for each team in each division (novice or expert)
    - Automate testing of submissions against correct output
    - Allow judges to view results of testing
    - Create chat client for teams to communicate with judges and forum for judges to broadcast to all participants
- Not Completed
    - Allow participants to solve non-programming logic problems

At the beginning of the semester, none of the team members had experience with full stack web development. Hence, as indicated by the big spike early on in the burndown chart below, we spent much of the first month going through tutorials and becoming familiar with Javascript, the React framework, and the mongoose library for database endpoints. Throughout the rest of the semester, we had broken down the work between the team members as follows, Andrew was responsible for the endpoints and the database, Ryan was responsible for the components on the problem page, i.e: the code editor, and the hackerrank call, Yash was responsible for user schema, the authentication, and several of the front-end components, and Courtney was responsible for the scoreboard page and some of the other front-end components.

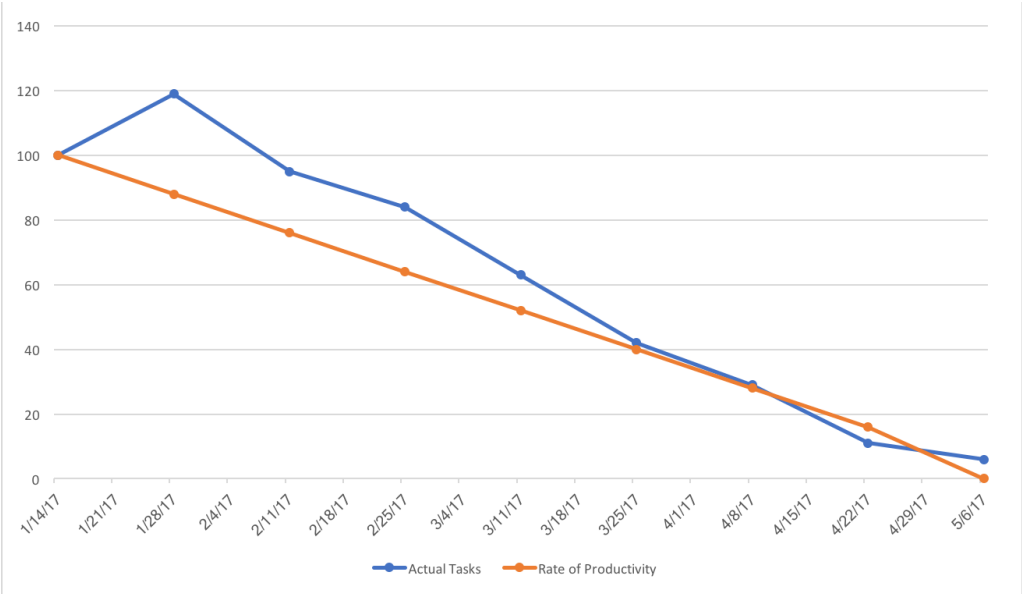

Figure 3: Final burndown chart

Limitations & Future Improvements

Limitations

Currently, there is concurrency issues with the website. For example, two judges should not change the the feedback for a specific submission at the same time either from correct to incorrect or from incorrect to correct. This will cause the score for that team to be off by one.

Future Improvements

One aspect that was not required by our client, but would be of use to others would be to allow multiple users on the same team. Currently, our system considers each user as representing one team, but in a new system, users would not only join a contest, but also join a team within the contest. Similarly, there is one judge for each contest. Having users be able to join as judges would allow the contest to scale and manage more participants. Additionally, as noted in the scrum, the team did not provide a easy solution for hosting logic problems on the website; this can be brainstormed by a future team.

As security was not our client's top priority, there exists a handful of security holes in our system that should be addressed. Going through this list of Node.js security features should tie up any loose ends. Also, we should remove any identifying hashes from the url and pass those identifiers to the application in a more discrete fashion. For example, if you navigate to the problem page, the url is contest/{$contest_id}/problems/{$team_id}/{$problem_number}. Ideally, it should be problems/{$problem_number}, and the contest_id and team_id are handled in the backend, out of sight from the user. A future team should also consider adding on integration for sign-in through Gmail/Bmail accounts.