University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2022

Programming Contest Platform

Dustin Black, Joshua Davis, Joseph Horner, Jack Malcom, Omar Moustafa

Abstract

The University of Arkansas currently uses a couple of archaic methods of judging its high school programming contests and similarly-structured classes.The existing proprietary solution limits contestants heavily on what programming languages they can use and forces several manual processing layers before the code can be tested. The online solution for classes like Programming Challenges is also very limited on what languages are allowed, and students have complained about trouble logging in and using the system. Our goal is to replace these systems with a single web-based application that is easy to use, lenient, and convenient.

Our approach to solve this problem is a flexible web app that allows users to create and join programming competitions as individually necessary. The front half of the application is written in the React framework for TypeScript, and the back half consists of a Java Spring Boot API, a PostGreSQL database, and a set of Docker images set up to compile and interpret many different programming languages. Together, these qualities create a much more user-friendly, versatile application that can be used not just by the University, but by anyone that wants to host a programming challenge.

1.0 Problem

Our solution is intended to replace two separate programs for two similar purposes. The first use for our platform is to judge programs submitted to solve challenging problems in the University of Arkansas' high-school-level programming competitions. The current solution forces participants to submit their uncompiled code directly to a locally hosted proprietary compilation service that implements PC^2, which is an extremely outdated solution that has very limited capabilities and requires each submission to be compiled and tested one at a time. It has not received a major update in fourteen years, and has not been updated at all since 2009.[1]

The second use for the platform is to offer an alternative to Online Judge in processing submissions for the University's Programming Challenges class. The Online Judge site was created back in the 1990s, and the front page admits that the current model of the site is over 15 years old.[2] Overall, the site is unfriendly to use, clunky, and full of problems. Students have even reported trouble creating accounts with their University email addresses, which makes the

issue of the professor attributing solutions to individual students difficult to resolve, as the site does not allow for group creation as our solution will.

2.0 Objective

Our project's main objective is to lay these older solutions to rest and finally apply modern usability standards to a similar application. Allowing any user to create and manage their own programming competitions as well as join others' competitions makes the site highly scalable. The user-facing half of the application was designed with user experience in mind and has been run past a user experience expert across its iterations. Users can rest easy with their data stored and used safely and responsibly, as our site authorization system is built on Auth0, which implements OAuth2 as an industry-standard protocol for access delegation.

The backend of the application is designed to be as fast and convenient as possible. Using Docker containers to compile code submissions allows us to provide a system that mimics as much of an actual development environment as possible, minimizing user error and friction in the setup process. This also allows for users to fill out their own test cases for the competitions they own, which will come in handy when they want to reward solutions that account for the most edge cases. Our target for the API and compilation integration overall is flexibility; the user should never be limited in what they can do with their programming competition.

3.0 Background

3.1 Key Concepts

Docker is a container technology. It uses a defined configuration to build virtual machines that can then be started, stopped, and configured on the host device. We have paired it with Kubernetes as a scaling solution for this project. This allows host machines and containers to communicate and adapt to their environment. For example, when under light load, the number of active containers can be scaled down, and under increased traffic the active containers can be scaled back up.

The *PostGreSQL* database is integral to the project's success. It mainly stores user account information and data on individual contests, including who the admin is, a list of participants, a list of problems within contests, and a list of submissions for each problem. It also draws connections between these fields as necessary and is structured under third normal form for low redundancy and low chance of data anomalies.

Spring Boot is a framework for Java centered around flexible APIs. Using this tool, we can take in large data structures organized in JSON from the frontend, implicitly convert them to Java objects for processing, query the PostGreSQL database, manipulate data, and return equally complex data by implicitly converting Java objects back to JSON.

Typescript is a sublanguage of Javascript that has become one of the most popular development languages over the last five years or so. It introduces a strict type system to Javascript, and many tools to take advantage of it such as interfaces, type unions, and filtering. One of the main advantages of this system is that the strict types make it much easier to introduce complex JSON objects into code without also risking the creation of unstable amounts of errors. Typescript is transpiled to Javascript at runtime, and the most popular implementation of the language creates a lengthy config to control the resulting Javascript output.

React is a Javascript and Typescript framework that allows developers to create components using a special flavor of Javascript called JSX. JSX differs from vanilla Javascript in that developers can insert HTML as an object in their code and make it dynamic using the Javascript portions of the component. Altogether, this creates a powerful environment for writing stateful, dynamic code.

3.2 Related Work

As discussed before, PC^2 is one of the technologies we hope to replace. It was perfectly acceptable for its time, but the project's updates have been inconsistent at best. The Association for Computing Machinery has widely switched to another solution as of 2010 that better suited their needs, and in 2014 PC^2 crashed nearly four hours into their Mid-Atlantic Programming Contest (one of the few cases where they still used it), for which a winner was never resolved.[3]

Online Judge could also stand for several improvements as previously stated. Students in the Programming Challenges class have reported many difficulties understanding the site, and plenty of inconvenient halts to their progress when it simply refused to cooperate. We hope that our solution prevents these types of disruptions.

There are also two popular websites for practicing software engineering interview questions to which we can draw many parallels. Binary Search[4] and Leetcode[5] are similar to our project in that they allow users to submit their code for evaluation through test cases, but different in that they take user submissions solely via having them type directly in the browser (as opposed to our flexible technique of still offering this option, but also allowing file uploads from the user, thereby allowing them to use their preferred development environment if they so choose) and users cannot host private competitions on those sites.

4.0 Design

4.1 Requirements, Use Cases, and Design Goals

Our project is an online platform where users can create and host programming competitions, which is able to handle many users submitting and uploading their code at the same time. The host of the contest is able to send out the ID of their contest so that users can easily register an account under that competition, which may eventually be expanded into a QR code generation or join-link system. The host will also eventually be able to configure many different settings such as the length of the contest, the maximum runtime of submitted programs, allowed languages, et cetera. The site also needs to be secure against malicious uploads that might target our compilation cluster for destructive purposes.

There are several requirements that we have aimed for the project to meet in order to assure a quality result. The main, broadest requirement was the creation of an online web page that allows users to both create competitions and join existing competitions. Each of those activities had their own requirements for full implementation. Creating a competition allows a user to name the competition, and will eventually allow them to create a list of teams for the competition and set up parameters that may be specific to that competition such as scoring methods. For joining a competition, a user is able to access the competition that they are trying to

join by inputting that contest's ID, provided by the organizer. In the future, we hope to expand this into a system that allows them to join contests either by viewing a list of them on the site, or by assigning each one a unique join link that the users may visit.

Our second priority was to provide infrastructure that can efficiently handle code uploads and run them in a way that is consistently performant. On top of that, the code also has its output checked, and its security is ensured. Since this needs to be done on our servers to guarantee submission integrity, we also check to make sure that user code doesn't pose a security risk. In order to fit these goals as well as possible, we have used Docker to implement this section of our architecture, which is a container system that allows us to create virtual machine images. Images are similar to ISOs, in that they contain an operating system and some code to be run. The key differences that Docker introduces are containerization and automation. Since Docker is intended to be an easy way to deploy code, most images are made by basing them on other "base" images and augmenting them with instructions specified in a *dockerfile*. The dockerfile is a reproducible way to "compile" an image before it is run that gives a list of terminal commands and launch commands needed to consistently create the runtime that will be used in production. As a standard example, a project based on Java might be based on an image that contains a minimal Linux distribution such as Alpine, the commands needed to install the project-specific dependencies, and a command to compile the code. Once that is done, running the image will call a specified launch command that will run the code compiled in the build process. In our case, we want a Docker image that builds our evaluation code, and heavily sandboxes the external resources the image has access to. This way we can evaluate user code while also maintaining security. We don't care what the user does in the container as long as it never escapes the container, at which point it would be indicative of an issue with Docker itself rather than the user's submitted code.

Another important requirement for our project was to secure our site. Running code remotely has many possible vulnerabilities, as malicious code could be extremely harmful to our website if we have no proper defenses. We have started by making the authentication process secure by using Auth0 to handle single sign on (SSO) verification for our page. Auth0 is a free service that implements Open Authentication 2.0 (OAuth2) verification which is an industry standard authorization protocol. By using Auth0, the majority of the work and security is handled for both signing in and registering a new account. It also stores the user information securely with a list of salted hashes for accounts, can handle user groups which will allow for admin accounts, and can easily support the addition of features like multi-factor authentication for added security.

The primary use case for the site is for the University to be able to host their programming competitions in a stable, decentralized manner. The current implementation requires all computers being used for the competition in the J.B. Hunt building to be temporarily booted into a customized Unix distribution with a carefully constructed firewall such that the enabled computers can only directly connect to the grading server over local area network, where the judges must still submit feedback manually. Many steps of this process can be automated or eliminated using our solution.

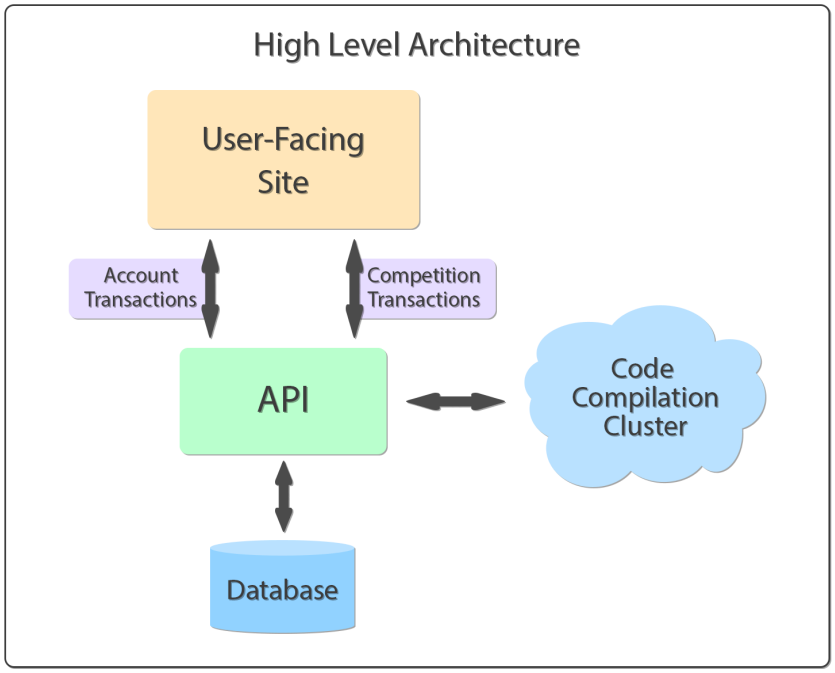
Our site will also end up being incredibly useful for anyone who wants to host or join a programming competition. Streamlining the process may make these competitions more common in general as well, which would allow for more in-depth analysis in situations such as programming-based scholarship applications or software engineering career positions. In these

cases, the panels that would normally ask for solutions and grade them individually could now set up a standard set of guidelines for submissions to be graded on and have it done automatically via our site. This also allows standardization of the testing environment for applicants, as well as centralization of their results, without any further effort on the organizers' parts: they simply tell their applicants to sign up for a free account on our site and join their contest.

The current website used by the University for the Programming Challenges course is Online Judge, which is a very outdated site. It lacks useful feedback, there is minimal language support, it doesn't allow instructors to create groups to add students, and the UI is extremely outdated. Our project aims to solve all of these issues by giving users access to better feedback through judge comments, adding support for more languages, having a system that supports the making of groups and adding members, and creating a modern UI by using the React NodeJS framework.

4.2 High Level Architecture

The site starts with a user-facing login, where the user can create an account and log in. These transactions communicate with the API to update the database as required with the new account information, or check that user credentials are valid and allow them to visit the user-facing contest/code submission section. The competitions loaded are the ones associated with the user in the contest database table, which are requested from the API to see which ones need to be accessed by said user when they login. The user sees a list of competitions, and when one is clicked they see the list of problems, each of which may then be clicked for further information and instructions. When a user then uploads their code or types it in the browser, it is sent to the API which will put it in a queue for the Docker image hosting the necessary code runner.



The API performs these checks with the compilation cluster asynchronously so that the site can display a notification to the user while their code is still compiling and running. Once it has been compiled and checked for malicious properties, it is run against test cases defined by the competition's admins one by one, which the API will be able to directly reference. These are passed in with the user's code to the Docker container as a string of command line arguments with which to run the code. The creation and formatting of these parameters is managed on the frontend web app, and they are stored in the database as a foreign key reference to an individual problem within a contest. This way, the test cases can be updated by the competition moderators at any time without interrupting the existing competition.

The final return from the API call to grade a submission is a JSON object containing a grade as a decimal number between zero and one representing what percentage of test cases the submitted code passed. We hope to also include a list of object pairs between failed test case IDs and suggestions (which the competition admins would be able to associate with some test cases ahead of time if they so choose) in the near future. As a very simple example, assume the problem is to design a function that multiplies a given number by two and returns the result. If the submitted code returns 5 instead of 6 on an input of 3, the competition admins could have a custom suggestion message returned as "double check your arithmetic operation, you're likely adding two rather than multiplying by two" which would help the competitors realize what needs to be changed about their solution. Keeping the test case parameters themselves out of forward-facing transactions entirely means that users will never be able to see what the test cases are unless they are an admin for that contest.

The competition admins are able to see all submissions by all competition entrants and what the grading system returned for them. Through this, they will in the near future also be able to attach more personalized feedback to individual submissions as they see fit (such as "you're very close, be careful with your decimal rounding"), which would be immediately visible by the submitter.

Users are also able to create contests, and they may also create new problems for contests in which they are an admin. To help with our mission of flexibility, we also hope to add numerous helpful filters that they could set, such as max runtime of the submissions, languages allowed to submit, contest start and end date, and who can participate. Users that join these contests are able to take advantage of all the features already discussed.

Frontend

The frontend is a React website written in Typescript. There are two reasons we want to provide a website instead of a full desktop application: development speed and accessibility. Web standards make it fast and easy to quickly iterate on our design, with React and Typescript only enhancing that. Using a web-browser-based technology also allows us to target virtually every modern device possible, making it easy for users to access, regardless of operating system or hardware. A website also has a subtle advantage of making the enforcement of our user flow easier. Each step in the flow can be represented as a page, or a component of one, which makes it simpler for the team to work on individual parts of the site without potentially breaking each other's code.

As previously stated, the core technology behind our site React, an open-source framework developed by Facebook to make dynamic web development easier. React apps are composed of small components that look and interact like standard HTML tags, that can store state and communicate with each other. We opted to use a version of React that is written in Typescript, which is then transpiled to Javascript that the browser can read at runtime. For code submissions, we use both Monaco, the text editor behind VS Code, and a default HTML file upload element. We also use Tailwind, a library that makes writing CSS faster, similar to Bootstrap. Finally, we use Axios, a library meant to make HTTP requests easier to read, create, and configure. Together, each of these technologies, along with some supporting libraries, come together to form the stack we use to make the frontend.

In practice, when a user goes to the site, they connect to the frontend container, and are served the code for the site. React then renders the site in the browser, and depending on where the user is on the site, a request is sent via Axios to fetch data for that portion of the site. If the request needs to be authenticated, then an additional request is made to Auth0 using their API to get a JWT that we use to authenticate the request. When the request is returned, React then updates the page state with the relevant information. This is the part where loading text gets replaced with the components you would expect. On the problem page, we send additional requests when making submissions, and use the data we get back from them to tell the user what percentage of the relevant test cases they are passing. This data is also shown on the list of problems itself, in the form of the problem background turning green when 100% of cases are passed.

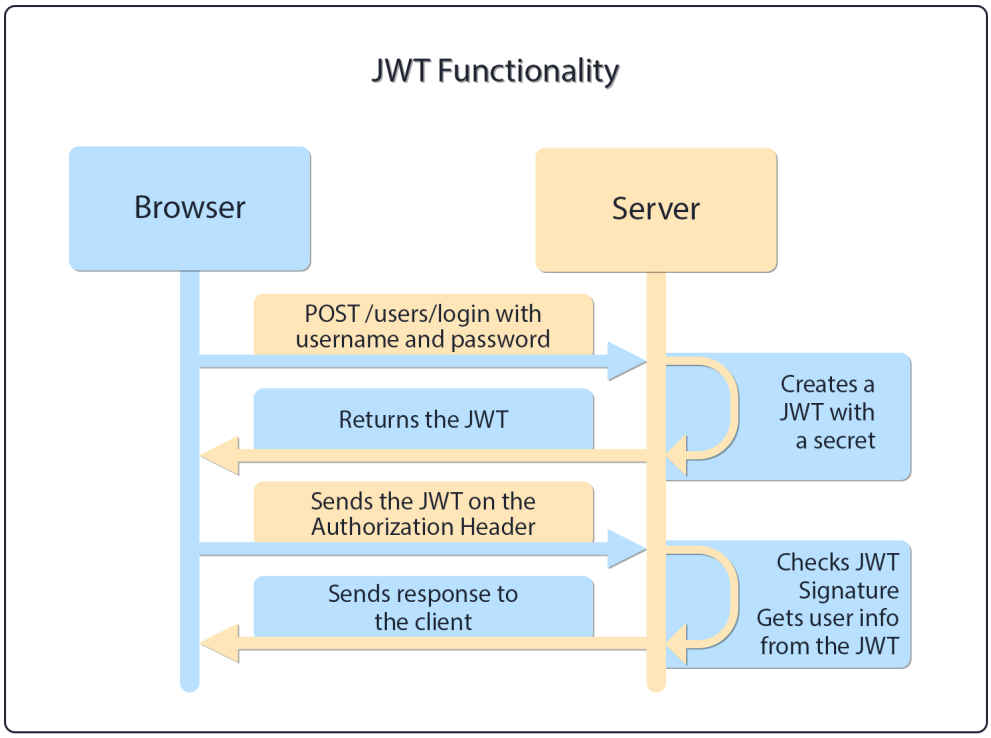
Backend

The backend consists mainly of two pieces: the Spring Boot REST API written in Java with Maven as a package manager, and the PostGreSQL database it communicates with. The database is currently hosted on Heroku for local testing purposes, but we have prepared a Docker image with the PostGreSQL engine that makes the transition to a privatized database an easy future step of simply copying the table schemas over from Heroku. The API itself is broken into endpoint controllers, DTOs, security classes, and a service, many of which integrate closely with the execution cluster.

The endpoint controllers are what actually accept incoming requests. These include both HTTP-GET and HTTP-POST type endpoints for each of the DTOs, as well as submission endpoints to handle communication with the compilation cluster. There are also a few specialized endpoints that help smooth the user's experience, such as one for joining contests and one for getting all the problems within a contest. The endpoints are easily manipulable from the frontend, and their simplicity allows for new endpoints to be added as necessary as the project progresses into the future.

The DTOs are simple Java objects that represent users, contests, problems, teams, submissions, and a few specialized wrappers. Each of these objects are automatically interpreted from JSON data and are remarshalled into JSON on their way back out from the endpoint controllers for ease of use on the frontend. The only reason the data spends any time in a non-JSON form is for the API to be able to manipulate its contents more quickly and conveniently as Java objects.

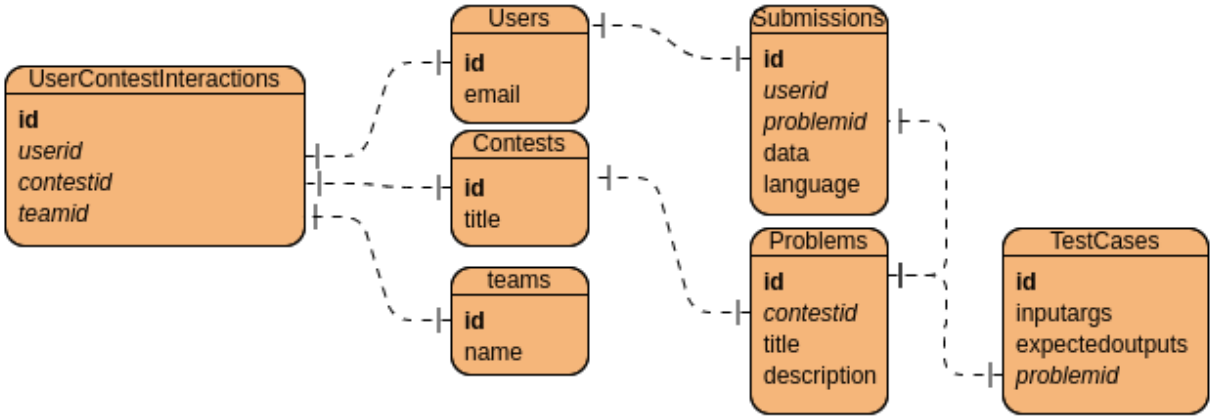
The security classes are what tie our API in with the Auth0 service to allow for access control. Each HTTP request coming from the frontend includes a JWT, or JSON Web Token, in one of its headers, which is used by the security classes to verify that the request is coming from a valid logged-in source. This is necessary because all of our endpoints manipulate sensitive user data, and only authorized users should be able to view the results of the API calls. On an invalid or missing JWT error, each endpoint returns a 403 unauthorized HTTP status code thanks to these security classes. Below is a diagram of how the transaction is modeled for maximum security.



The service class we have created employs Spring Web's built-in REST service class to allow our API to *make* HTTP requests as well as it accepts them. Through this class, the Spring API can communicate with the wrapper API that surrounds the code execution cluster. It wraps the language of the submission, the code itself, and the test case arguments into a nice little HTTP entity, and sends that off as a POST request. This is triggered multiple times in a row from the create-submission endpoint, running the code once for each applicable test case with the correct arguments.

The PostGreSQL database is where all of our information is stored. First we have the Users table, which ties into the Contests table in a many-to-many relational model. Because we're using PostGreSQL as the database engine, this many-to-many problem requires a third intermediary table, which we have aptly named UserContestInteractions. This manages the tying-together of user IDs, contest IDs, and team IDs, all of which is used to tell the frontend which users are participating in which contests, and under which teams. Because of that relationship, we also have the Teams table to keep track of team IDs and names. Separate from that Gordian knot of logic, we have the Problems table, which keeps track of all of the aspects of individual problems. Since problems and contests are in a one-to-many relational model, a third mediation table is not necessary between them. We also created a TestCases table, which stores

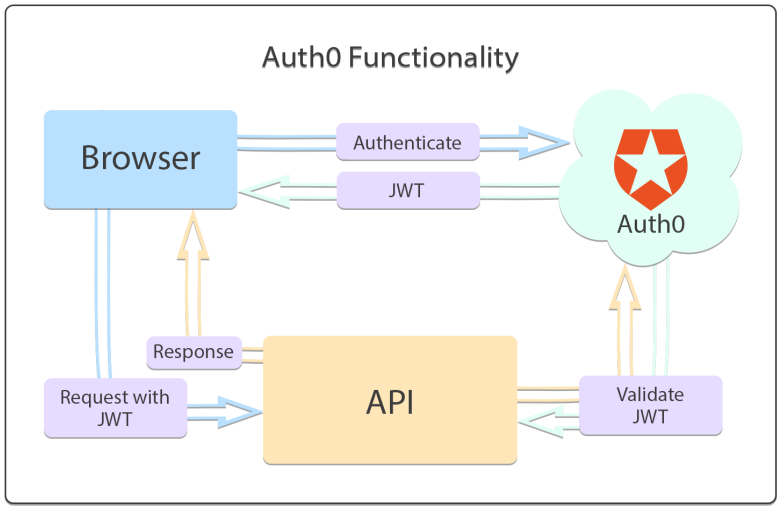
test arguments, their expected outcomes, and which problem they belong to. Finally, we have the Submissions table, which stores users' code submissions to each problem they attempt. It also holds what language they submitted in, and what proportion of the relevant test cases they succeeded with. These seven tables come together to host all of the data necessary to run the site. An ER diagram of our database setup is included below.



Integration between Frontend and Backend

Our frontend and backend are set up to follow a backend-for-frontend style architecture. This has a couple of key benefits. First, it allows us to deploy our entire application as microservices, small components of a full application that can be deployed to multiple servers at once. It also allows us to scale each component of our application as needed.

When a user logs into their account on the frontend, a JWT is created and signed through Auth0 before being sent back to the user. That JWT will continue to be used with the API for authentication and authorization purposes, allowing the user to access certain information while also protecting their routes and resources. JWTs introduce a granular security that allows us to specify a specific set of permissions per token, which overall improves debuggability. This model is shown below, with all explicit communications outlined.



Code Execution

When code is submitted for a problem, we need a way to actually execute it. If we relied on the users to execute the code on their devices, they could potentially figure out which test cases we use, or even fake their result entirely. Code runners solve this problem by acting as a secondary, non-public API in our system that allows us to run code on the backend, without creating potential security risks or exploits that could be used to gain an unfair advantage or interact with our system maliciously. We also wanted to create something stateless that would allow an arbitrary number of code runners to exist, since it would make scaling them up for larger contests easier.

The code runner is one of the smallest yet hardest to implement pieces of the entire project. However, using containers makes it possible to run user code in a sandboxed environment, where they only have access to the data and tools we want them to. In our implementation, we use Docker in Docker, or DinD, to allow the code runner to host containers inside of itself, something not normally allowed in Docker. This allows us to create Docker images for each of the languages we want to support, and run them as needed.

When we get a submission, it contains the language, code, and test cases we need to execute user code appropriately. The code and arguments are saved to the main runner instance, and then the code execution environment is started. It's important to note that a few things happen when we start the environment. First, this is the main point where we enforce security in the system, so networking and several basic Linux capabilities get disabled. Next, we mount the code as a bind volume to our container. This allows the container to access the submitted code and arguments. Finally, we tell the container to expose its shell. This gives us a way to read the output from the container so we can send it back to the Spring API.

Deployment (Containers)

We have two main ways to deploy our platform, one for development, and one for production. In development, we use Docker compose. This is a tool that rests on top of our already-existing docker containers, and defines how to set up the containers we want to run, and how they interact with each other. This is a great development option because it allows us to build directly off of what we already have. As previously stated, Docker also has the benefit of separating out each of our services into isolated containers. This makes building much easier as you don't need to have every toolchain used in every repository to build, and most of the configuration you might need is already done for you.

Our production deployment uses Kubernetes. Using MicroK8S, the Canonical distribution of K8S, we deploy our containers to a distributed network of Raspberry Pi devices that each can be assigned to run specific parts of our application. Currently we have stuck to running our code on Docker compose to keep development time fast, but the flexibility of our dockerized system will make moving it to Kubernetes simple. The main advantage of using Kubernetes is that it simplifies the process of scaling a microservices system like ours out to a distributed system of computers. It's also an industry standard, and that combined with containers gives us some added security.

4.3 Risks

Risk	Risk Reduction
Bad code compilation times	Appropriate backend resource allocation to optimize times
Poor User experience	User flow tests with outside participants to test and account for usability
SQL injection attack	Query parameterization and prepared statements, as well as sanitizing user input wherever possible
Malicious code execution	Scan code for vulnerabilities before running, use skeleton main function to run user code, and lock down docker container in strict sandbox
Cross site scripting (XSS)	Review and test code for XSS vulnerabilities, use good programming practice and review code
Password security	Store user passwords as salted hashes offsite through Auth0
Secure connection and transmissions	Use HTTPS to make sure that data isn't intercepted by malicious agents
Compromising code execution environment	Heavily sandbox arbitrary code execution as much as possible, remove network and OS access from user code, and use a new VM every time code is run.

4.4 General Task List Overview

1. Set up final architecture idea
    - Finalize architecture details and graphic
2. Get a site up and running with basic account creation and sign in process using OAuth2
3. Test signup/login capabilities
4. Start on contest page
5. Flesh out sub-pages for contests such as individual problems and rankings
6. Test contest creation, moderation, joining
7. Allow code file submission to problems
8. Compile user-submitted code on cloud cluster
9. Test full competition capabilities
    - Run mock competitions to verify functionality with multiple users
    - Test compiler thoroughly
10. Prevent possible attacks through submitted code
    - Verify website integrity/security
11. Test user flow
    - Have a new user test the site's flow by navigating through it, then address any feedback they've provided to improve the layout of the site.
    - Fix any errors and implement improvements before moving on.
12. Document full site

13. Extensive testing
    - Stress tests - large number of users, test advanced algorithms or special case scenarios, etc. (Try to break it, address issues as necessary).
    - Mostly done along the way, no explicit dates set out for it.
14. Project report and submission

4.5 Schedule

For the most part, our schedule was simply broken down into biweekly sprints in which we decided what was most necessary to get done during that time, as we are following an agile style of development. However, we did have general guidelines for when the stages of the project needed to be finished. These timelines certainly ended up overlapping as different parts of the team began working on different parts of the project, but this staging of the project into discrete pieces allowed for less context-switching and a more efficient workflow.

Each task is visible with its corresponding assignee(s) on one of our GitHub Project boards.[6] Please note that tasks marked without date ranges (on either April 9th or April 23rd) actually did take place entirely on that day. Our group made the decision to meet up for hackathon-esque long coding hours on those two dates. One may also notice a gap between March 8th and April 9th. This is due to our spring break, midpoint project presentations, and midterm exams for other classes, all of which took temporary precedence over this project.

Tasks	Dates
1. Set up final architecture idea	11/8-12/10
2. Get a site up and running on localhost with Typescript React and setup to work with tailwind CSS	1/22-1/24
3. Get frontend running on a Docker container	1/27-2/3
4. Create skeleton backend components	1/18-2/8
5. Dockerize backend	1/18-2/8
6. Create K8s cluster	1/18-1/8
7. Create login page, and setup Auth0 group for the website, allowing User signup and login through Auth0	2/3-2/9
8. Create basic database structure, including Users table, Problems table, and Contests table	2/8-2/15
9. Explore Auth0 integration with Spring Boot	2/8-2/15
10. Add page routing and set up an Auth wrapper to secure routes from unauthenticated users, add Contest and Problems page	2/8-2/21

12. Coordinate many-to-many relational model for Users and Contests tables	2/15-2/22
13. Auth0 dependency injection with Maven for Spring Boot	2/15-2/22
14. Integrate Auth0 with basic User API endpoints	2/22-3/1
15. Add dynamic routes for multiple Contests and Problems, create API models and add styling	2/21-3/5
16. Write out API endpoint documentation for frontend utilization	3/1-3/8
17. Finish off Problem and Contest endpoints with Auth0	3/1-3/8
18. Research code runner structure	3/18-4/20
19. Add basic API requests to frontend	3/9
20. Add a header to the site, and add remaining API models to frontend app	3/10
21. Update landing page, add logo to site	4/9
22. Give Auth0 appropriate scope	4/9
23. Add get-usercontests API endpoint	4/9
24. Research reformatting of Problem Submission acceptance	4/9
25. Restructure database to fit final schema, add Submissions table	4/9
26. Restructure Problems DTO and API endpoints	4/9
27. Add Submissions DTO and API endpoints	4/9
28. Create API endpoint to get all Problems in a Contest	4/9
29. Research Auth0 user identification	4/10-4/16
30. Add Problem API endpoint to get data from ID	4/10-4/16
31. Add API endpoint to get a User's Submissions for a specific Problem	4/10-4/16
32. Add wrapper-API call to interface with code runner	4/17-4/22
33. Write code runner	4/20-4/23
34. Add User admin check for contests	4/23

35. Add Test Case submission for Problem creation	4/23
36. Simplify User API endpoints to not store data that Auth0 handles for us	4/23
37. Add create problem and create contest requests to frontend	4/23
38. Integrate code runner, browser editor, and file upload	4/23
39. Add fresh logo to header	4/23
40. Update API documentation	4/23
41. Add page refresh on creation of new contest/problem	4/24
42. Add joining of contests to frontend functionality	4/24
43. Assemble final presentation slides	4/24-4/25
44. Assemble final project report	4/24-4/25

4.6 Deliverables – This project was broken up into several components in order to make it easier for the team to delegate work. As such, our deliverables follow similar categories.

- React: Front-end written using Typescript
- Spring Boot: Back-end written in Java
- Docker/Kubernetes container deployment
- PostgreSQL database
- User account system that compliments Auth0's authentication system
- Final report, including documentation of site functionality and mapping
- Demo of final website including prepared competition for viewers to experience site flow

As the project is entirely web-based, there will be no need for installation or configuration scripts. This level of personalization will not be necessary often enough or by a low enough level of user for us to consider allowing individuals to personalize the boot options for the site, API, or compilation cluster. We will, however, provide the dockerfiles for the containers to be runnable.

5.0 Key Personnel

Dustin Black – Black is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed related courses up to and including Software Engineering, Database Management Systems, Mobile Programming, and Algorithms, and is currently enrolled in Information Security, Computer Networks, and Wearable and Ubiquitous Programming. He was a software engineering intern with Affirma Consulting for nearly two years, and is currently a software engineering intern with SupplyPike. He is responsible for much of the API development and database structuring.

Joshua Davis – Davis has prior experience in front-end related website work and is a senior Computer Science major in the Computer Science and Computer Engineering Department at the

University of Arkansas. He is currently enrolled in Information Security, Computer Networks, and Wearable and Ubiquitous Programming, with related courses like Database Management, Algorithms, and Software Engineering completed. He is contributing mainly to the back-end of the project, UX design, and any necessary graphics or illustrations.

Joseph Horner – Horner is a Computer Science major with experience in mobile app and web development. He is currently enrolled in Formal Languages and Computability. He is currently employed at J.B. Hunt as a software engineering intern. He is contributing primarily to the back-end for the project, creating endpoints to query the database for the front-end. He also made minor contributions to the front-end creating some of the routes that were used.

Jack Malcom – Malcom is a Computer Science major with previous personal and professional frontend experience at Affirma, Supplypike and Tesseract Ventures, with current employment at Hathaway. He is currently enrolled in Formal Languages and Computability, and has completed relevant coursework such as Software Engineering, Algorithms, and Mobile Programming. He has a strong background in web-based technologies, as well as Linux administration and cloud hosting. He is focusing on the code compilation cluster for the project.

Omar Moustafa – Moustafa is a senior Computer Science major in the department with previous experience making secure web apps with a University research team. He is currently enrolled in Wearable and Ubiquitous Programming, and has completed relevant courses including Algorithms, Mobile Programming, Database Management, and Software Engineering. He is working primarily on the frontend of the application.

Dr. Wing Ning Li, Professor and Project Champion – Dr. Li completed his bachelor's degree in Computer Science at the University of Iowa in 1982. He also completed his master's degree in Computer Science at the University of Minnesota in 1985, followed shortly by his Ph.D. from the same institution in 1989. His research interests include design automation, algorithmic analysis, and parallel computing.

George Holmes – Holmes has been with the University of Arkansas for over 40 years. He was originally a student starting in 1978, then performed academic advising between the years of 1985 and 1994. From then into the 2000s, he was an instructor in the CSCE department, and since 2005 he has been the master scientific research tech for the department. He will assist this project by helping us understand what technology we'll be replacing and how our project needs to be used.

6.0 Facilities and Equipment

We have used four Raspberry Pi 4 8GB SBC units in order to deploy our project. This was in order to guarantee that our code evaluation is as consistent and fair as possible to the end user. We want to make sure that each submission of user code gets its own device to run on. This is to avoid a case where a single device may only be running one user's code while another may run several at once, which could severely impact run time. By making sure that each user runs their code on dedicated CPU cores, we will greatly improve the security guarantees we can provide, and will also improve evaluation integrity. As for facilities, we have done much of the work for our project in physical proximity in the PRIME Lab on campus, and would like to thank the faculty, staff, and volunteers who maintain the space for providing such a wonderful environment.

7.0 References

[1] *A History of CSUS' PC^2*, Sacramento State, http://pc2.ecs.csus.edu/pc2history.html

[2] *Yearly Statistics*, Online Judge, https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=23

[3] *PC²*, Wikipedia, https://en.wikipedia.org/wiki/PC%C2%B2#cite_ref-1

[4] Binary Search, https://binarysearch.com/

[5] Leetcode, https://leetcode.com/

[6] *Hatstone* Projects board. Github, https://github.com/orgs/Hatstone/projects?type=beta