Solana Code Analysis as a Service

Critical Design Review

Rust Cleaners

Arsh Kabarwal

Joseph Carpman

Kipp Corman

Olivia Ornelas

Department of Computer Science

Texas A&M University

03/20/2022

Table of Contents

1 Introduction (1 page; 5 points) - Done

1.1 General scope and problem background

With the emergence of smart contracts and programs on the blockchain, there is a learning curve for creating secure contracts and programs. It can be difficult to understand and comprehend all the security measures that need to be present in the programs. Our service will help educate developers with many of these pitfalls by evaluating the numerous pitfalls for the user of the service. Since code that is submitted to the blockchain can not be amended or changed, it is important that the code is secure so malicious attackers can't take advantage of the flaws in the unsecure code

1.2 Needs statement

The current barrier to entry of blockchain smart contract development is too high. The difficulty of modifying or updating deployed smart contracts demands robust code review tools to catch programming errors before deployment. Currently, code review tools that analyze code meant for blockchain requires the user to download the tool. Additionally, most tools require the user to send portions of their code directly to the company in order to get a quote for how much the tool will cost. Our project aims to create a low cost tool that doesn't require the user to download anything in the form of Software-as-a-Service.

1.3 Goal and objectives

The goal of this project is to implement a Software-as-a-Service tool which allows users to quickly and clearly review their code before it is deployed to the blockchain. This tool should be as user friendly as possible to lower the very high barrier of entry to smart-contract deployment. The minimum viable product would allow users to submit and run their code, and then visually show what lines have potential security/performance issues. In addition to this, the product will review a user's code using multiple tools and present the results in an easy to understand way. Additional features we plan to add are an interface that highlights code where errors are prevalent, a progress report, and an area to view history and progress over time.

Objectives for this project include the ability to reduce the cost of using a blockchain code review tool by 20% and identify at least as many vulnerabilities in their code as the existing products do. Additional objective include:

- The product will be able to review code using at least 3 code review tools and produce results from each on the same dashboard.
- The product will be able to accept code from at least file upload and github url.
- The product will preserve up to 10 prior submissions for comparison.
- The product will preserve code review report data for historical modeling.

1.4 Design constraints and feasibility

One constraint in our project is that we are limited to using open source static analysis tools that already exist. We do not have the time or expertise to create our own, which means we are relying on others. For our objective of showing as many problems in the submitted code as existing tools, this entirely depends on how effective the tools we end up finding are. Another constraint will be a time constraint. This project has to be completed within about 3.5 months which can cause a lot of features and functionalities to be left out or uncompleted. Finally, we are potentially financially constrained. If this product ends up gathering a lot of users over time, there would be additional costs to start up more servers so that the users don't have extremely slow responses. All of their past entries would still need to be stored, and over time with people possibly submitting entire codebases this could be quite large too.

1.5 Validation and testing procedures

Majority of the testing will be done within the Rust Cleaners team. Edge cases will be tested for every functionality to make sure that there aren't any huge bugs in any of the code. In addition, if time permits, there will be user testing where selected users will use the interface. The users will then determine if they enjoyed using the service and potentially recommend any functionalities that they would like to see.

2 Proposed design (5-10 pages; 60 points)

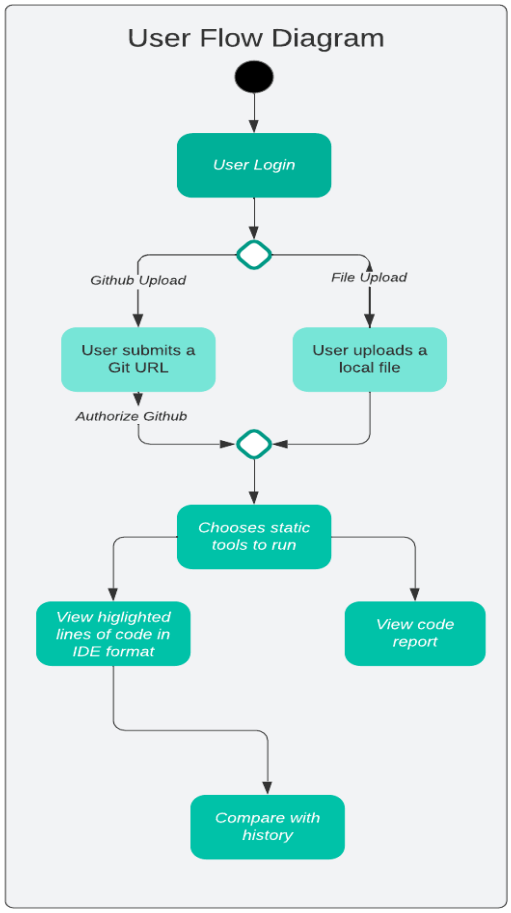
2.1 Updates to the proposal design (if applicable)

A new alternative that was evaluated is Remix IDE for Ethereum. Remix IDE is an open source tool that allows developers to write and test Solidity contracts for Ethereum. This service has a file structure interface that will be utilized in the Solana Code Analysis as a Service. Once a file is clicked on, the contents of the code is displayed on the right hand side as shown in the image below. Solana Code Analysis as a Service will display the results of the tools highlighted on the code to allow the user to easily see where the vulnerabilities lie.

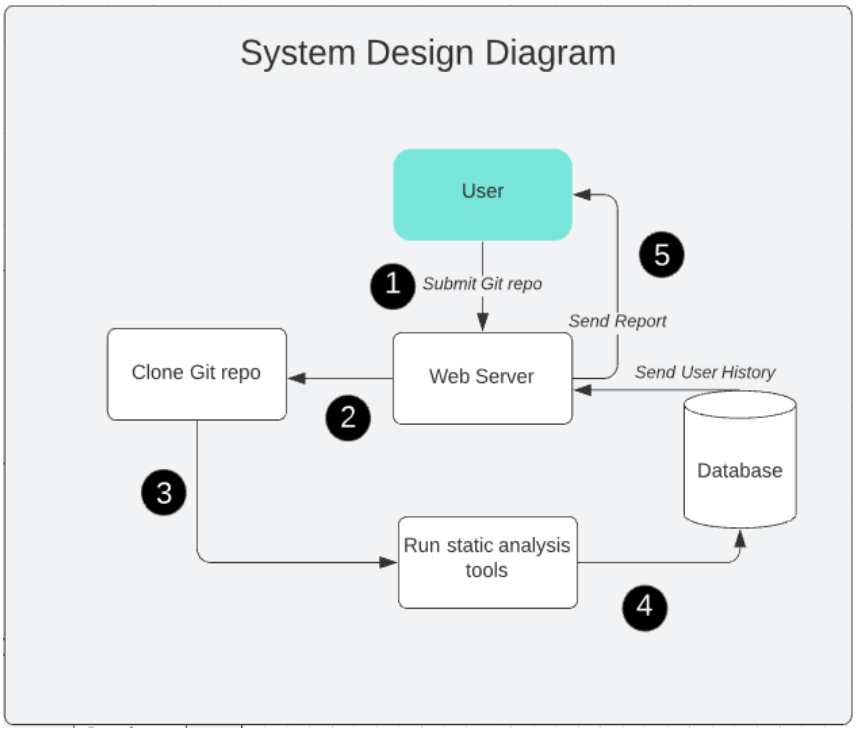
Image of Remix IDE's interface

2.2 System description (1-2 pages; 30 points)

The system description will consist of a high-level block diagram, and a functional description of the different parts and interfaces.



The image above is the updated user flow diagram for the Solana Code Analysis as a Service project. An added component is the ability for the user to choose which static tools they want to use. There will be a checkbox functionality that displays all the available tools they can run. If multiple tools are chosen on the same code, a carousel feature will enable the user to toggle between the different results from the different tools. The user will be able to comfortably see where most of the errors and vulnerabilities lie. A code report will also simultaneously be created which will establish how secure the code is and also display the raw results of the static analysis tools. The results of the tools will be able to be compared with previous submissions of the code to display how the code has been affected by recent changes.



The image above is the updated system design diagram for this project. The main difference here is that we have decided against using a separate server to handle the code analysis tools due to simplicity and the goals of our project. The original reasoning was to create a more distributed system so that it could scale better, but in the short time span of this project we feel that our time is better spent adding more features to the website. If additional scalability is needed, we plan to horizontally scale by using more AWS servers rather than distributing our tasks across multiple systems.

As before, the web server will be hosted with AWS Elastic Beanstalk and will include a Docker container that runs all of the code analysis scripts. Once the user sends in their Git repository, the server will clone it locally, run all of the code analysis tools on it, and save the results in our database (AWS DynamoDB). The database will store the commit ID and Git repo being analyzed so that we can match it up to past requests made for the same repo. This is useful in order to cache results to avoid rerunning the same exact commit ID multiple times for different users, and to help us show more useful information by comparing with past versions that the same user has already run. The web server will then pull these results and display it in an easy to read format. This format will look similar to an IDE where the file directory is displayed on the left, and by clicking on individual lines of any file you are able to see what warnings are coming from each section of the code. Finally, all of this information as well as a summary of their code will be returned to the user.

2.3 Complete module-wise specifications (4-6 pages; 30 points)

The Solana Code Analysis as a Service uses a Docker container to contain all of the dependencies to run all the different tools. This saves plenty of time for users by preventing the need to download every dependency to run the different tools, and allows us to easily add additional code analysis tools by just adding them to our container. To run all of the tools on the user provided code, bash scripts are used to run every command in a command line interface. This automates the process of manually typing out the commands. The bash script locates where each static tool should be run. For example, the cargo audit tools need to be run where there is a Cargo.lock file and clippy needs to be run where the rust files are. The script stores the output of each tool in a different json file.

Then, Javascript is used to read through the json files and print out all the key information in a readable format for the user. Since the results from the tools are all stored into json, the key-value pairs are run through a for loop to store the most important information. A javascript file is created for every static tool to read the output of the tool. Therefore, each tool has a json file, where the tool's output is stored, and a javascript file, where the output is parsed through.

At the completion of this project, there will be many different static analysis tools that the user can choose from. Currently, the user can only choose between cargo audit and clippy. Cargo audit analyzes security vulnerabilities for crates in Cargo.lock files. Clippy interprets and catches mistakes in Rust code. Soteria, a tool that identifies and eliminates security vulnerabilities built specifically for Solana contracts, will be added in the near future along with other tools. These tools will be chosen based on their accuracy and reliability in providing important information on possible vulnerabilities.

The front end is done using React. React is used to create the website for the service. The website currently takes in a GitHub url and clones the repo using a middleware function to automatically download the Github repository. Prime React is used for user interface components for the website. It was used to output a directory tree of the uploaded repository. The directory tree allows the user to click on different branches of the repository to see where different files are stored. The checkbox for the analysis tools was also created with Prime React. Future user interface elements will also use Prime React to ensure continuity.

The website is hosted on Amazon Web Services. AWS allows the website to be viewed with a specific link. The database to store all the information will also be stored on AWS. DynamoDB will specifically be used because it is a noSQL database that is included in the AWS free tier. Although the data we plan to store is technically structured, the main thing that will take up space is just the user's code and the code analysis results. With lots of users, this could end up being a lot of text and noSQL is better for scalability. In our database we will be using the Git repo (partition key) and commit ID (sort key) combined for the primary key. The partition key ensures that anything with the same value will go on the same partition, so by using the Git repo we will be able to easily check if any other commits for the same repo have been run before without looking across database partitions. The database objects will store information about the Git repo being analyzed as well as what code analysis tools were run and what the output was. Since some of the tools take a long time to run, and we plan to only add more, having a caching system is important. Finally, though using the database in this way greatly reduces the runtime and improves our overall user experience, some users may not want their results to be stored, especially if their code is not public. The user will have the option of not storing anything, though that also means that all added functionality based on past runs will not be shown.

Though this feature is only currently in progress, below is an example of what a database entry might look like:

```
{
    "AnalysisResults": [
        {
            "RequestMetadata": {
                "UserID": ...,
                "GitRepo": ...,
                "GitCommitID": ...,
                "ToolsRan": ...,
            },
            "CodeResults": {
              "File": {
                "Name": ...,
                "Directory": ...,
                "Warnings": {
                  "Line": ...,
                  "WarningText": ...,
                  "Fix": ...,
                  ...
```

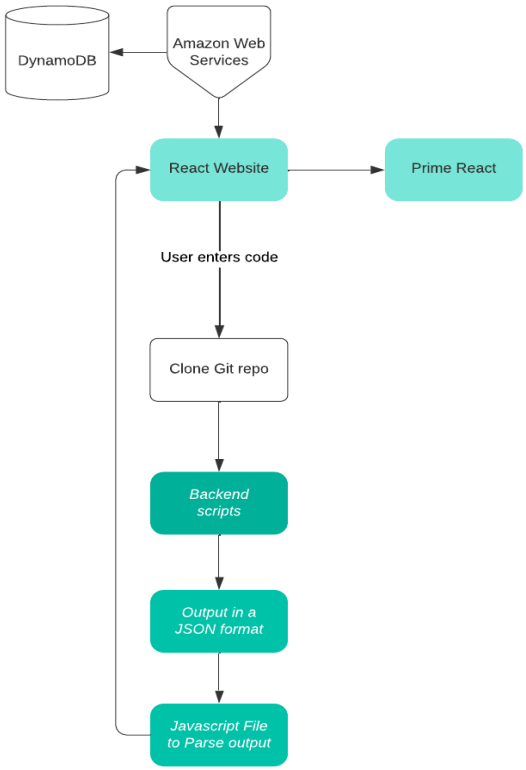


Image of Subsystem hierarchy

3 Project management (3-5 pages; 30 points)

Arsh is the project lead and was in charge of creating backend scripts to run the code-review tools. These scripts essentially combine the results from all of the different tools so that they are all in the same format for easy display on the frontend. He also created checkboxes on the frontend that allow users to select which tools they want to receive results for. Furthermore, Arsh worked on helping Kipp create the Docker Container to hold all the dependencies for these tools. As the project lead, he is responsible for scheduling regular team meetings and turning in weekly reports.

Kipp has experience in distributed systems and networking communication between different services so he will be in charge of system design. Kipp started by creating a Docker container that includes the dependencies for all of the code analysis tools that would be used by the web server. He then focused on backend component functionality. He set up the DynamoDB database and created a framework for what entries would look like. Then, he worked on a file directory tree that shows on the frontend and displays file's content with language syntax.

Olivia has the most experience with developing web applications for classes and on club project teams, and will be in charge of software design. Olivia worked on printing console output to the frontend in an organized way. Because most of the data is sent to the server, she focused on making sure the data is visible on the website, provides a loading page, and prints onto a component instead of just plaintext.

Joseph has the best understanding of blockchain and has experience working with database systems, so he will be in charge of technical reporting and help with software design. Joseph started by making the backend and the basic fetch requests and responses that the rest of the web app is based on. The first task was to accept the git repo URL and use it to automatically download the given repo to a remote directory. Next, Joseph made the backend run the scripts made by Arsh and Kipp and return the output to the frontend. After this task was done and the demo given, Joseph focused on helping other team members interact with the back end and worked to format the output of the scripts to be better standardized and displayable.

The management approach that we decided on in the original project proposal is working for the most part. However, there have been some small adjustments we have made so far. We have continued to use our bi-weekly meetings in class in order to address concerns anyone has with their current tasks or to get input from other team members. Additionally, whenever we are planning to have a demo in our Monday meeting we will meet shortly the night before to make sure that everyone is on the same page. We have also maintained using GroupMe to keep in constant contact with each other, as well as sending out updates whenever an important change is pushed or someone believes that something in the code should be changed. This is not something that we anticipated, but has become very useful. Based on each of our expertise areas listed above, we can suggest changes to some aspects of the project. Additionally, by messaging each other about what big changes have been made, we are essentially doing component testing. If the next person who works on it does not see the result that they have shown in our chat, then clearly something was done incorrectly.

We have also kept up to date with our Gantt charts and have used that to keep track of who is working on what and what the current progress for each task is. However, this ended up being slightly annoying because the tasks we were going to work on each week changed very frequently. We had originally planned to leave too much frontend development for the latter end of the project, and quickly realized that this would hurt our demos. This resulted in a restructuring of our task schedule. Something we started doing in addition to the Gantt chart was assigning everyone tasks at the beginning of each week after having our meeting with the professor and TA, and writing them down separately. This way we were positive about what everyone was doing even if our chart was slightly out of date at times.

For brainstorming sessions and overall resource sharing using a shared Google drive has been very successful and we will continue to use it. The same GitHub organization is used for version control, and all finance decisions are talked about as a team beforehand.

3.1 Updated implementation schedule (1 page; 10 points)

Our schedule consists of four different phases. The first phase is to create the minimum viable product which proves the validity of our idea and allows the user to upload their code and see a result. The second phase involves adding features that we feel are necessary in order to compete with many of the competitors we analyzed when first crafting the idea. This includes combining results from multiple code analysis tools, hosting on a website so that the user doesn't have to download anything, and showing them when multiple tools pick up on the same warnings. Our third phase is for additional features that improve the product but aren't as important. This is where we will allow users to create accounts to store their history, allow them to automatically analyze their GitHub repos whenever they push to it, and output some sort of summary of how well written their code is. Finally, our fourth phase is to polish the application and make sure all functionality is there. There are not many features listed under this time, as it is mainly to ensure that we can get everything we want to have working in the short time frame. If it is already done, we can work on a stretch goal.

This section shows the progress we have made on the UI. Currently, the user can see the code they submitted and look through the files with a directory tree. The output for the code analysis tools is then displayed below it, allowing them to easily see what caused the errors.

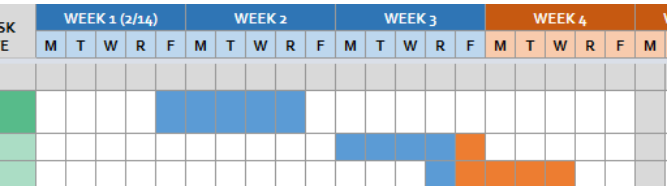
TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 1 (2/14)					WEEK 2					WEEK 3					WEEK 4					
				M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M
User Interface																								
Basic code upload to web app (GitHub)	Olivia	4	100%																					
View code analysis results on web app	Olivia	5	100%																					
File directory tree for code uploaded	Kipp	3	100%																					
Display user's code on website	Kipp	1	100%																					
Let user choose which analysis tools to use	Arsh	2	50%																					

In the next few weeks, we want to take this further by allowing users to see the code analysis results on top of their code, and combine the results so that they can see what percentage of the tools found each warning. Finally, when the database is fully implemented and user accounts are set up there will be additional pages on the frontend for that.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 5 (3/21)				WEEK 6					WEEK 7					WEEK 8					WEEK 9					WEEK 10				
				T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
Show code analysis results on top of their code		5	0%																												
View analysis report		3	0%																												
User profile page		3	0%																												
See past code submissions and history for repos		4	0%																												
More polished frontend		3	0%																												

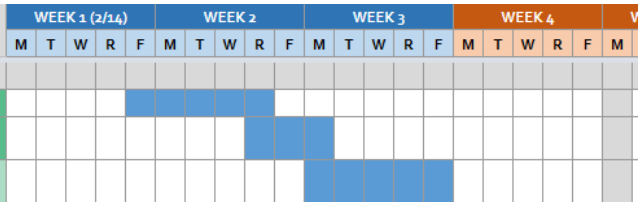
All of the pipeline tasks came early on, but some of the components are not quite put together yet. We have hosted a website and created a docker container for analysis tools, but the docker container is not currently set up on our AWS server with the website. The database is fully set up, but not fully utilized yet.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE
Pipeline			
Create Docker container with all code analysis tools	Kipp	3	100%
Host website	Joseph	5	50%
Set up database	Kipp	5	50%



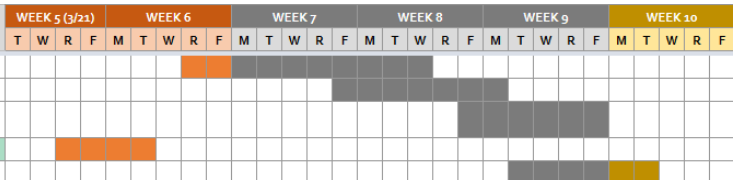
On the backend, our website will clone the repo/branch submitted by a user and run the code analysis tools on it. It will also run through the repo and create a directory tree json file which is used by the frontend to allow the user to search through their files.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE
Backend			
Clone GitHub repo link locally	Joseph	5	100%
Turn local repo clone into format for the directory tree	Joseph	3	100%
Send code analysis request/result to database	Kipp	4	50%



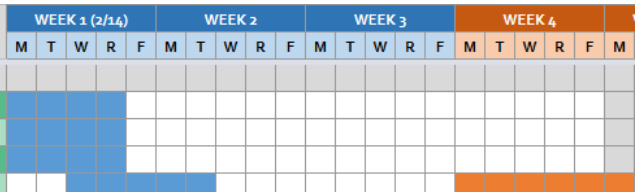
In the future we plan to create the backend infrastructure for accounts, and use the database to create history and cache results.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE
Account login/logout/creation		5	0%
User code analysis history		5	0%
Show user diffs for past submissions of same repo		4	0%
Cache results if same repo submitted often	Kipp	3	50%
Auto analyze repo whenever pushed		5	0%



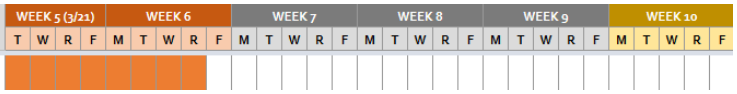
For code analysis tools, we have gotten quite a few to successfully run locally so far and created a script to combine all the results and display it on the website.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE
Code analysis			
Run cargo-audit on code	Arsh	3	100%
Run Soteria (free version) on code	Kipp	4	50%
Run Clippy on code	Arsh	3	100%
Write script to combine tools	Arsh	5	50%
Send uploaded code to server	Joseph	2	100%
Send test report back to the web server	Olivia	2	50%



The main thing left to do here is to figure out when the different tools are actually warning the user about the same thing. We want to combine them to emphasize that it might be an important warning, and to not give the same information twice.

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE
Highlight tools finding the same warnings as each other		5	0%



Another important part of our schedule that isn't written on the Gantt chart is testing. Testing will be done periodically as the components are created. In addition, extensive testing will be done in the last few weeks to ensure that every edge case is taken care of.

3.2 Updated validation and testing procedures (1 page; 10 points)

The ability of the user to submit their code and receive informative results will be validated by user testing. Users will utilize the service and determine how they liked the service and whether there should be any other added functionalities. The Rust Cleaners team will observe the user while they navigate through the website and write down their observations. For the user testing, there will be at least 3 users evaluated. The analysis of the user's experience will allow the team to observe how a user would interact with the structure of the website. Since this project is a Software as a Service, it is important that user feedback is taken into account.

The file upload and github upload and report history feature will be tested as described in the Project Proposal. The file upload and github upload tasks will be tested by submitting files of different lengths along with identical files hosted on github. This feature will be validated if the result of each file is identical to its GitHub counterpart. To test the code and report history feature, a new user account will be created and 11 versions of the same code will be submitted. The system will pass this test if the last 10 code versions are available for comparison and the report history is viewable in both text and graphic format.

3.3 Updated division of labor and responsibilities (2 pages; 10 points)

Provide a detailed list of deliverables (and due dates) for each team member.

Arsh will work on adding more tools to the scripts and interpreting the results of the tools. He will also help speed up the run time of the process of running the tools so the process is more efficient, and help with frontend development regarding tool selection.

Some specific tasks and due dates are:

- Implement a fully functioning checkbox for tools on the frontend: due 3/27
- Add Soteria as a working tool: due 4/4
- User file upload functionality: due 4/11
- User testing and maintenance: due 4/18

Kipp will mostly work on the backend and interactions with the database, and help with displaying code analysis results on the frontend since he worked on the file output. He will also maintain the Docker container. Some specific tasks and due dates are:

- Help combine the code analysis tool results with the file outputs on the frontend: due 3/30
- Store code analysis results in the database, and use cached result instead of rerunning: due 4/4
- Show user's past submissions on their profile: due 4/11

- Show user diffs if they submit an updated version of a repository they have submitted before (what issues were fixed, what wasn't): due 4/18

Olivia will focus primarily on the frontend, working on how to display all of the information from the code analysis tools in a user friendly way. She will also work on creating additional methods for uploading code and receiving results. Some specific tasks and due dates are:

- Combine the code analysis tool results with the file outputs on the frontend: due 3/30
- Create the carousel element to switch between pages: due 4/4
- Create an automated report that states how secure we think the code is and displays raw results: due 4/11
- Styling all the UI elements: due 4/18

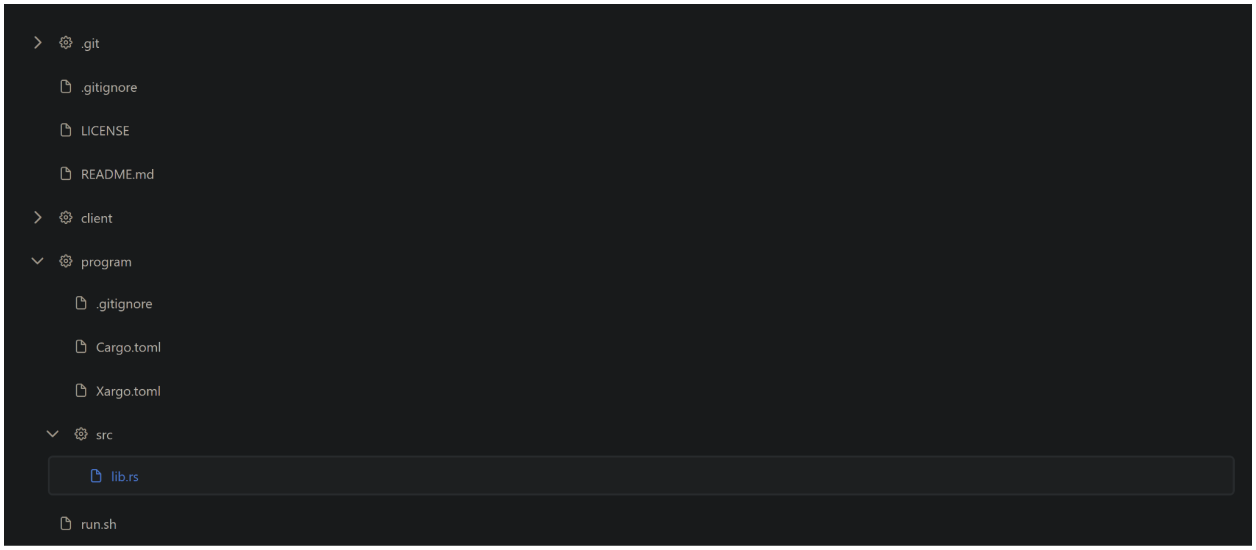
Joseph will start by finishing up the work we have done with AWS, finalizing our web server and all of the components we need on it. He will then help doing backend development and user accounts. Some specific tasks and due dates are:

- Get code analysis tools onto AWS web server: due 3/27
- Functionality for the user to ask for specific file testing: due 4/4
- Increased efficiency in subsystem to prevent long wait times: due 4/11
- User testing and maintenance: due 4/18

4 Preliminary results (1-2 pages; 5 points)

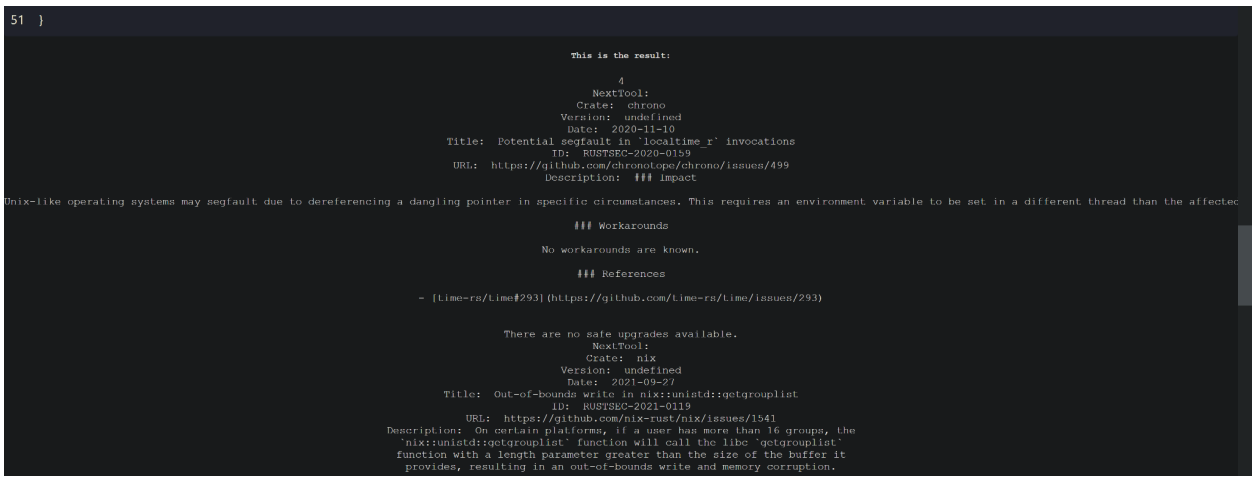
Provide any test results and demo of completed parts of the system at the time of the CDR.

This is the first page of our website that prompts the user to enter a Git repo link and choose which tools to run.



Once submitted, the user is shown all of the files inside of the repo they cloned.

By clicking on any file, they can view the file with its corresponding language syntax highlighted like an IDE.



Finally, the code analysis output can be seen below the code. We are working towards combining this output inside of the code to further benefit the user and have the warnings be more visual than just text.