CrimLog Design Report

Melanie Spence & Elijah Olmos

CST-451 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Revision: 1.0

Date: November 20, 2022

ABSTRACT

CrimLog is an open-source software protocol that utilizes Web 3.0, near-field communication (NFC), and non-fungible tokens (NFTs) to create a student attendance solution that is unparalleled in the market. Unlike other products currently available, CrimLog provides students custom minted NFTs for each class. Due to NFTs immutable nature, once a student is marked as present there is no way for that to get revoked, which makes CrimLog the most reliable student attendance solution.

A student will enter the classroom and tap their NFC card against an NFC reader, which will provide them with their NFT and mark their attendance. Behind the scenes, the NFC reader utilizes the NFC card's unique identifier to associate the card with a particular student via their student ID number. This student ID number is added to the queue for the class, which is visible by the professor. Additionally, throughout the duration of the class, a professor may manually issue a NFT attendance mint for a student; however, if a manual action is not performed, NFTs will be minted automatically for the queue at the end of the class. To ensure students attend for the duration of the class, a professor may remove a student from the queue prior to the automatic trigger. Once the trigger is activated, the student's digital wallet will be pulled and a custom NFT asset with classroom details will be minted into the student's wallet. Finally, all attendance transactions will be viewable on a web dashboard and will include links to blockchain transactions. CrimLog is the ultimate student attendance software solution with superior functionality and support that is not present elsewhere on the market.

History and Signoff Sheet

Change Record

Date	Author	Revision Notes
November 20, 2022	Melanie Spence & Elijah Olmos	Initial draft for review/discussion

Overall Instructor Feedback/Comments

Overall Instructor Feedback/Comments

Integrated Instructor Feedback into Project Documentation

☐ Yes ☐ No

Table of Contents

Design Introduction

This design report provides essential information for project design and development by outlining the design of CrimLog with diagrams showing the project flow, proposed user interface, logic for the program, display of technologies, and other design entities. The diagrams will provide a high-level design of the proposed solution, providing insights on the design and execution of CrimLog. Additionally, these diagrams and flow processes will show how the application will operate from the user's perspective. Furthermore, the diagrams and flow process will display the behind-the-scenes operations of the product and the procedure the product must undertake to properly function. These high-level reports will allow one to view the anticipated design of CrimLog.

All external project deliverables will be referenced within this design report to provide further insight into the design of this solution. Diagrams will depict the logical and physical design of the system, illustrated with block diagrams, component design, solution architecture and configuration, non-functional requirements, schema design, frameworks, and third-party libraries. Additionally, details of the Proof of Concept (POC) implementation will be provided to present relevancy through rationalization for the usage of specific technologies. Moreover, logical and physical solution designs along with detailed technical designs, including sitemaps, API design, schema design, and flow charts. Furthermore, technical risks will be assessed, and external resources will be provided. Therefore, this Design Report will provide the information necessary to understand the overall design of CrimLog.

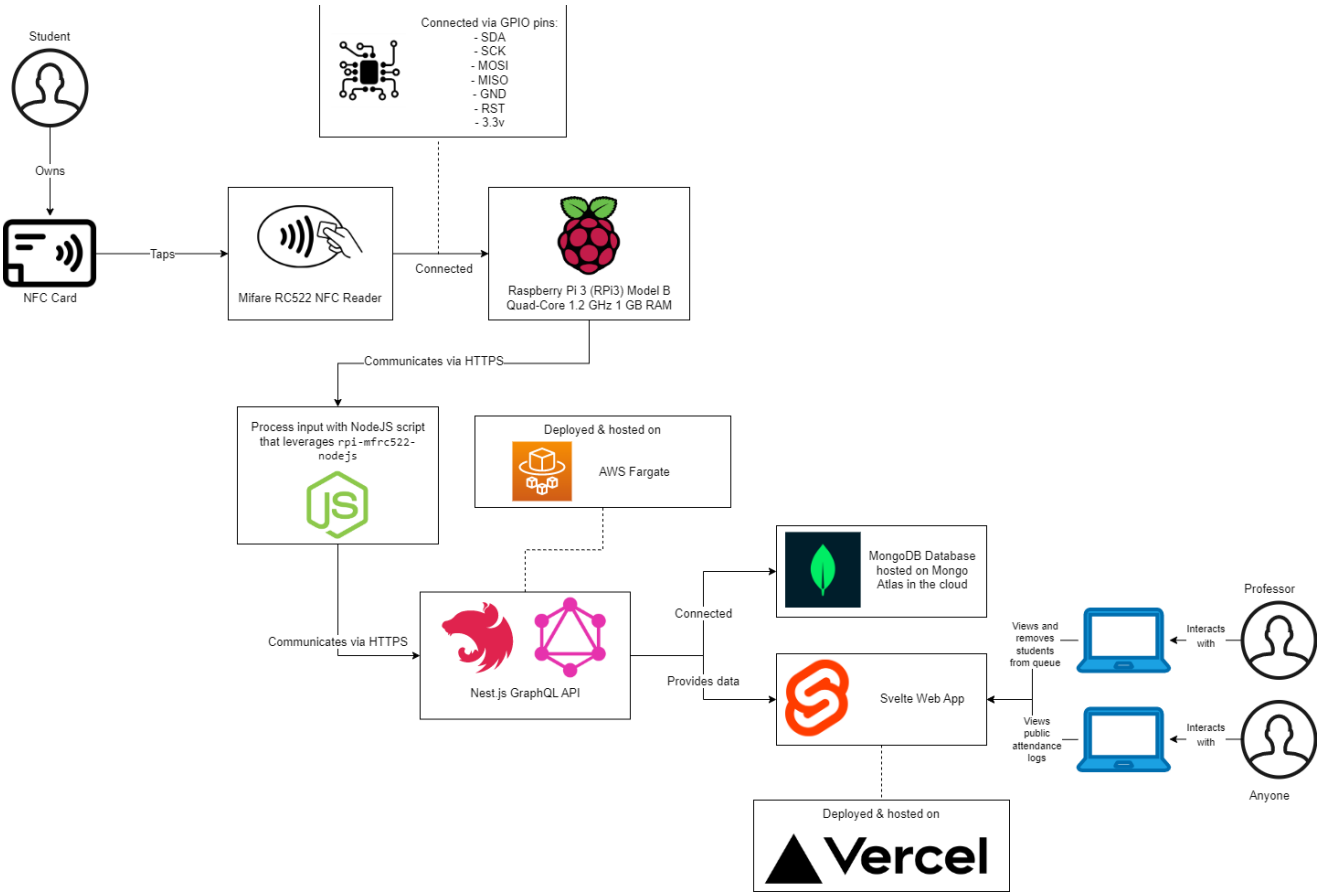
Deliverable Acceptance Log					
ID	Deliverable Description	Comments	Evaluator (internal or external as applicable)	Status	Date of Decision
1	API Design	API Design utilizing GraphQL Documentation	External - https://bit.ly/crimlog_apidesign_graphql	Complete	2022-11-04
2	Project Proposal	The initial Project Proposal for CrimLog	External - https://1drv.ms/w/s!AiCF1hcKxI7hhnas8G8Toba3xbQA?e=jpCqar	Complete	2022-09-25
3	Requirements	The milestone requirements for CrimLog	External - https://1drv.ms/w/s!AiCF1hcKxI7hhn2eba5x8WDpqiZd?e=8gyaQI	Complete	2022-10-16
4	User Stories	The user stories for CrimLog	External - https://1drv.ms/x/s!AiCF1hcKxI7hhn9p1P17ajzvei6E?e=VGYfq9	Complete	2022-10-16

Proof of Concepts		
Description	**Rationale**	**Results**
1. Raspberry Pi	Connects hardware NFC reader and software NodeJS script. Vessel for receiving NFC card input, processing the data, and outputting a result	NFC card input data is read and processed, displaying a specific output depending on the NFC card, including the NFC card unique identifier and the cardholder's name.
2 - NFC Card Reader	Hardware that connects the Raspberry Pi that receives card input and transfers the information to the Raspberry Pi	NFC Card Reader receives NFC card input data and sends it to the Raspberry Pi
3 - Node.JS Script	Runs on the Raspberry Pi. This script processes the NFC card input data from the NFC Card Reader and provides a specific output.	NFC card input data is read. The Node.JS script processes the NFC card data to display the NFC card's unique identifier and the cardholder's name.
4 - NFC Cards	Provides input for the NFC Card Reader. Each NFC Card is unique.	NFC card is tapped against the input reader, which provides the NFC card unique identifier, which is processed.

Hardware and Software Technologies
1 – Raspberry Pi 3 (RPi3) Model B Quad-Core 1.2 GHz 1 GB RAM
2 – Mifare RC522 NFC Reader
3 – NFC Card
4 – NodeJS 16.18 LTS
5 – Svelte 3.51
6 – SvelteKit 1.0
7 – Tailwind CSS 3.1
8 – DaisyUI 2.31
9 – Vercel
10 – GitHub
11 – TypeScript 4.9
12 – Nest.JS 9.1
13 – GraphQL 16.6
14 – MongoDB 6.0
15 – Apollo GraphQL Client 3
16 – Prisma 4.6
17 – MongoDB Atlas
18 – Docker
19 – AWS ECR
20 – AWS ECS
21 – AWS Fargate
22 – AWS Route53
23 – AWS CloudWatch

Block Diagram:

This diagram provides a high-level view of the overall technological design for CrimLog.

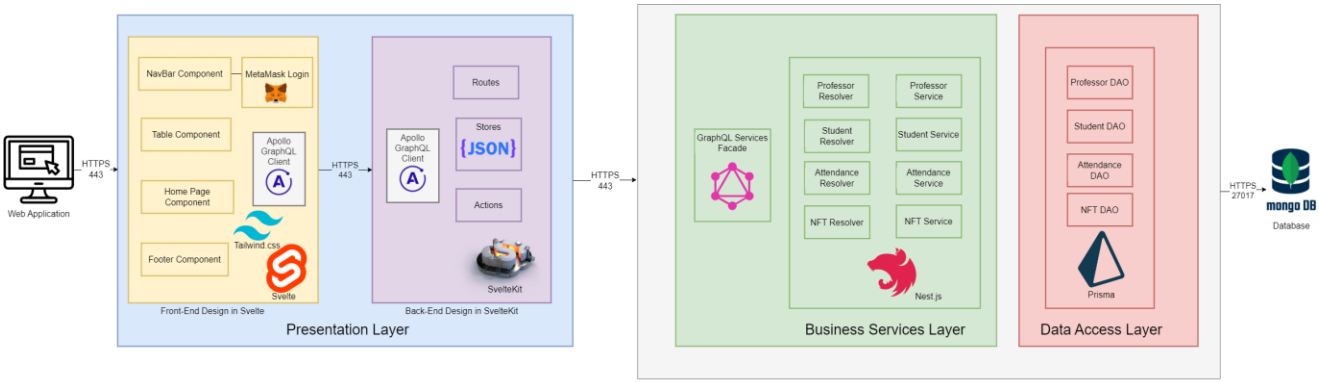


For a larger view of the diagram, please visit: https://bit.ly/crimlog_blockdiagram_tech

This diagram shows a high-level view of the process flow including technological design for CrimLog. A user with an NFC card will tap the card against an NFC Reader. This NFC reader is connected through GPIO pins to the Raspberry Pi. The Raspberry Pi will communicate with a NodeJS Script, that will process the input of the NFC card. The script will then take the processed input and communicate the output with the Nest.js GraphQL API. This API will be hosted in AWS Fargate. Additionally, the API is connected and retrieves data from a MongoDB Database, which will be hosted on Mongo Atlas. Additionally, the API provides data to the Svelte Web Application hosted on Vercel. Professors will be able to interact with the web application to view and remove students from the queue, which will send data to the API to update the database. Additionally, all users can interact with the web application and view the public attendance logs, which will retrieve information from the MongoDB database via the API.

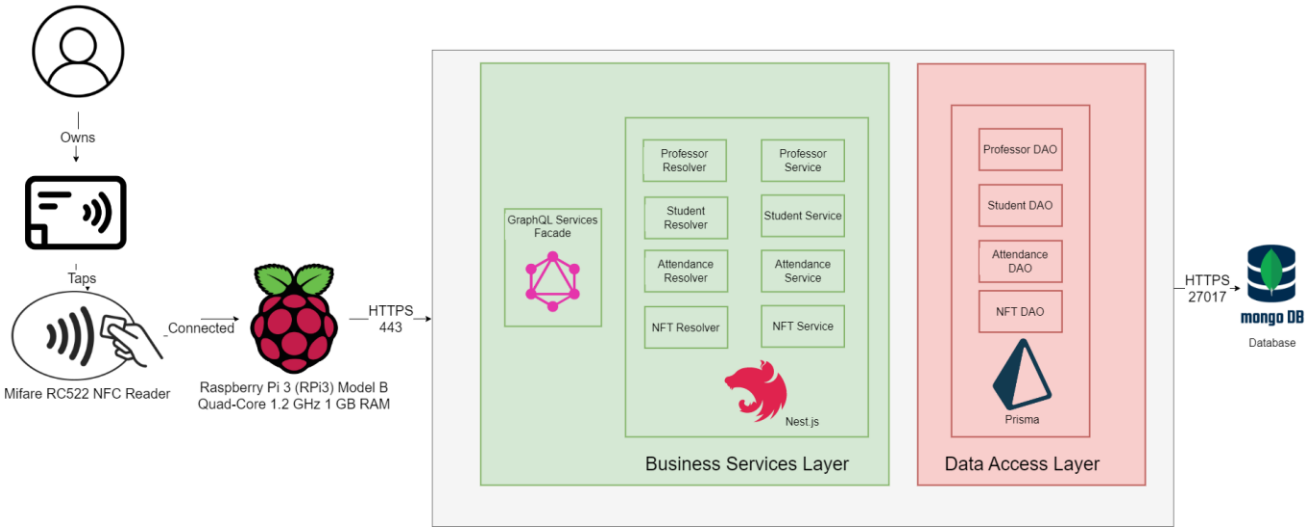
Logical Solution Design:

This diagram provides a high-level view of the web application for CrimLog utilizing Nest.js, GraphQL API, Svelte, and MongoDB.



For a larger view of the diagram, please visit: https://bit.ly/crimlog_logicaldiagram

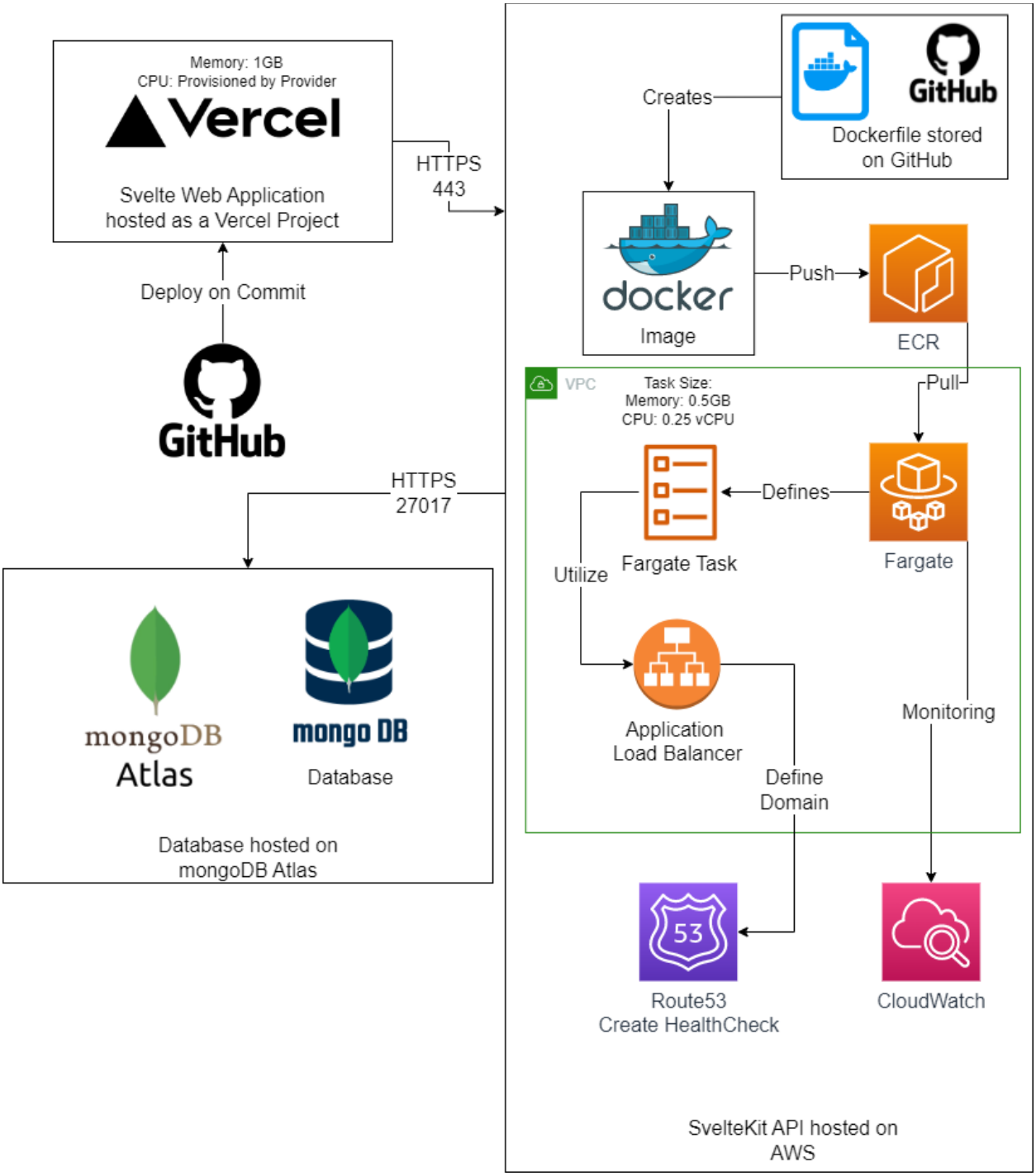
This diagram outlines the software solution and design for the web application portion of CrimLog. Utilizing the N-Layer design, the Presentation layer will utilize Svelte components with Tailwind CSS and Svelte kit routing and stores in tandem with the Apollo GraphQL Client to communicate with the web application and business service layer. The business service layer will utilize GraphQL services façade and Nest.js services, with a component service for each data access component. The data access layer will utilize Prisma for each DAO component, which will be communication and querying the MongoDB database directly. These connections will all be established through HTTPs.



For a larger view of the diagram, please visit: https://bit.ly/crimlog_pi_logicaldiagram

This diagram outlines the software solution and design for the Raspberry Pi NFC portion of CrimLog. A user taps an NFC card against the NFC reader, which is connected to a Raspberry Pi. This Raspberry Pi will communicate over HTTPs to the Business Layer. A NodeJS Script leveraging rpi-mfrc522-nodejs will process the tap and communicate with the GraphQL API endpoints. From this point, data will be pulled in the same way described above between the business services and data access layers, allowing for code reusability.

Physical Solution Design:



For a larger view of the diagram, please visit: https://bit.ly/crimlog_hardwarediagram

The Svelte Web Application will be hosted as a Vercel Project. Vercel connects to a GitHub repository and automatically publishes and deploys changes when code is committed to the repository. The Vercel project will be configured with 1GB memory while the CPU is provisioned by the provider.

The MongoDB Database will be hosted within mongoDB Atlas. The mongoDB database within mongoDB Atlas will be configured manually in the mongoDB Atlas console.

The API built within SvelteKit will be hosted within AWS and configured through the AWS Management Console. Within GitHub, CrimLog will store a Dockerfile, which creates a Docker Image. This docker image will be pushed and stored within AWS elastic container registry (ECR). CrimLog will also be leveraging the AWS elastic container service (ECS). A cluster is created and configured to utilize Fargate to run the Docker image alongside a virtual private cloud (VPC). Additionally, a task definition will be created, which defines what Docker image should be run and how it should run, which once again will be configured to be utilized with Fargate. This task will utilize the smallest amount of memory and CPU to reduce costs, with a configuration of 0.5GB memory and 0.25vCPU CPU. Moreover, a service, security group, and load balancer will be configured for the cluster, once again for Fargate, where CrimLog will utilize the already created VPC. Once completed, CrimLog will have a public task running in Fargate, which makes it accessible to other users since it is in the cloud. However, rather than having people access the task-instances directly, CrimLog will utilize Route53 to create a domain name to direct to the load balancer to expose the site on the Internet. Now, CrimLog will be hosted on AWS utilizing Fargate and be accessible with a specific domain name configured in Route53.

Detailed Technical Design

General Technical Approach:

Details of the design and approach will be outlined in the Detailed Technical Design section of this design report, providing a detailed view of key technical design decisions such as chosen technologies and frameworks, details of the team's proof of concept, a database design, a diagram showing the process flow of the application, a sitemap, user interface designs, component diagrams, service API documentation, information about the non-functional requirement which is external API documentation, operational support design, and other supporting documentation.

Design & Approach:

CrimLog approaches design with one mission in mind, Keep it Clean n KISS (keep it simple stupid) it (KICK it). The KICKit mentality focuses on keeping code and infrastructure practices simple. Rather than getting tangled in unnecessary complexity, CrimLog will utilize best practices and technologies within the industry to deliver code that is clean, accomplishing the necessary tasks while cutting the fluff. A KICKit focus helps CrimLog stay on the cutting edge of technology. Using KICKit, code is adaptable, flexible, and dynamic, so as the industry changes, CrimLog changes alongside it. CrimLog is not simply another participation software solution, CrimLog is THE participation software solution.

CrimLog approaches logical and hardware solution diagram design with the KICKit mindset. The logical solution design comprehensively displays the design and configuration of the software utilized to create the web application. Additionally, a diagram is created to show the interaction and relationship between the Raspberry Pi hardware and software components. Furthermore, the hardware solution displays the cloud hosting providers and configurations utilized by CrimLog to make the program accessible on the Internet. The KICKit mentality is used throughout the planning process because the configuration and design emphasize ease of use, simplicity, and clean practices. Rather than create over-complex solutions, CrimLog strives to design solutions that are powerful and efficient.

Key Technical Design Decisions:

CrimLog is leveraging near-field communication (NFC) technology, which is a cross-platform software that is also used by some of the biggest companies in the industry including Apple, Google, and Bank of America. Additionally, CrimLog is further setting itself apart by using Web3 technologies and concepts through non-fungible tokens (NFTs). To record participation, a user will tap an NFC card against an NFC reader, which will process this tap and add the user to a queue, viewable on a web application. Once in the queue, a professor can remove a user or mint an NFT for them. Once the class ends or when issued by a professor, an NFT will be minted for each student, which will act as their record for participation. Since CrimLog is leveraging NFTs, the student's participation is immutable, eliminating human error and unauthorized tampering.

CrimLog utilizes both hardware and software to create its participation software solution. The hardware components explained above will be connected to software components. A script created in JavaScript will run in a NodeJS environment on the Raspberry Pi whenever an NFC tap is inputted. This script will process the input and transport it between business and data access layers to update the database. Once this database has been updated, the presentation layer will detect the changes by polling the API and update the display of the web application to match the database. The business layer utilizes a GraphQL service façade, with services created in Nest.js. The Data Access layer will have data access objects created with the Prisma Client. This layer communicates with the service layer and the MongoDB database, querying the database to retrieve and modify data. The presentation layer will include a front end and back end created in Svelte with Tailwind CSS and SvelteKit respectively. The back end communicates with the business service layer and defines the stores, actions, and routes for the web application. Both the back end and front end will utilize the Apollo GraphQL client to handle data. The back end and front end communicate with each other in the presentation layer to display and modify data. The front end houses the DaisyUI components that make up the website and create the user interface that the user can view and interact with. Some components within this front end include a home, navigation bar, met mask login, and table view. This allows the web application to accurately display information within the database.

Technologies & Frameworks

Near-Field Communication (NFC):

Purpose – Allow students to conveniently mark their participation within a class and eliminate the professor having to track participation manually.

Chosen – Chosen for its convenience. Additionally, its growing popularity and the accessibility it provides for the project.

Raspberry Pi 3 (RPi3) Model B Quad-Core 1.2 GHz 1 GB RAM:

Purpose – Allows hardware and software components to be connected.

Chosen – Chosen for its ability to easily connect hardware and software components.

Mifare RC522 NFC Reader:

Purpose – Allows the Raspberry Pi to receive NFC card inputs.

Chosen – Chosen to provide compatibility between NFC cards and the Raspberry Pi and necessary because NFC was chosen as the technology for students to input their participation.

NFC Card:

Purpose – Allows the student to conveniently mark their participation within the class. Provides input for the NFC Card Reader.

Chosen – Chosen to provide students with an easy way to mark their participation. Allows NFC to be utilized and provides the input for the NFC Card Reader, which allows the Raspberry Pi to operate.

NodeJS 16.18 LTS:

Purpose – Process the NFC input of the Raspberry Pi. Once the input has been processed, this script alerts the API that as can has been made to begin the process of inputting the student into the queue.

Chosen – Chosen to provide a way for the Raspberry Pi hardware to communicate with the API software. NodeJS was chosen because team members of CrimLog have experience with the software and its compatibility with the Raspberry Pi Hardware.

Svelte 3.51:

Purpose – Create the front end of the web application for CrimLog.

Chosen – Chosen because the framework reduces the amount of boilerplate code written by developers.

Tailwind CSS 3.1:

Purpose – Provide styling to the front-end web application components for CrimLog

Chosen – Chosen because the framework helps developers write and maintain CSS code within the application.

DaisyUI 2.31:

Purpose – DaisyUI is a Tailwind CSS component library that provides further styling options for CrimLog

Chosen – Chosen because the component library helps developers utilize Tailwind CSS for faster development, cleaner HTML, and more customization.

SvelteKit 1.0:

Purpose – Create the back end of the web application for CrimLog.

Chosen – Chosen because it is the fastest way to build web applications in Svelte.

TypeScript 4.9:

Purpose – TypeScript will be utilized to create the CrimLog API

Chosen – Chosen because it excels at longevity and eliminates technical debt in environments where numerous developers contribute to the codebase, like in open-source software projects.

Apollo GraphQL Client 3:

Purpose – Manages local and remote data with GraphQL.

Chosen – Chosen to help structure code in a predictable and declarative manner that is consistent with modern development practices. Additionally, allows the front end and back end to manage data with GraphQL.

GraphQL 16.6:

Purpose – Query the CrimLog API.

Chosen – Chosen as the query language because it provides CrimLog the exact data requested. GraphQL was chosen for its speed and predictable results, providing the developer more control of data.

Nest.JS 9.1:

Purpose – Create services found in the business layer.

Chosen – Chosen because it is lightweight and promotes code reusability, reducing the amount of programming for CrimLog developers.

Prisma 4.6:

Purpose – Create data access objects found in the data layer. Retrieve data from the MongoDB database.

Chosen – Chosen because it provides intuitive data models, type safety, and autocompletion, reducing the amount of programming for CrimLog developers.

MongoDB 6.0:

Purpose – Store data to track students within a queue

Chosen – Chosen because it is a non-relational database. Additionally, it is simple to use while being efficient and effective.

MongoDB Atlas:

Purpose – Host the MongoDB Database in the cloud.

Chosen – Chosen because it is easy to implement with MongoDB database. Additionally, it has a simple installation and does not have a high cost.

Vercel:

Purpose – Host the Svelte Web application.

Chosen – Vercel was chosen because it is a simple and effective way to host a web application in the cloud with continuous integration and deployment while developing.

GitHub:

Purpose – Allows developers on the team to make changes to the code at the same time. Allows code to be stored in the cloud to prevent data from being lost.

Chosen – GitHub is an industry standard for Git. Additionally, developers at CrimLog have experience with GitHub and it integrates with Vercel.

Docker:

Purpose – Containerizes the application into an image, which allows CrimLog to leverage Amazon Web Services (AWS) for cloud hosting.

Chosen – Docker was chosen because it integrates with the AWS cloud provider and is an industry standard. Additionally, developers at CrimLog have experience with Docker.

AWS Elastic Container Registry (ECR):

Purpose – Stores the Docker image for the CrimLog API and integrates easily with other AWS projects selected for the completion of the project.

Chosen – Industry standard for storing Docker Images to be utilized with other AWS services.

AWS Elastic Container Service (ECS):

Purpose – Allows CrimLog to host the CrimLog API within AWS and easily integrate with ECR, which provides ease of access for pulling the Docker Image.

Chosen – Industry standard for hosting applications in AWS. Provides more customization than AWS Elastic Beanstalk.

AWS Fargate:

Purpose – Allows CrimLog to host the CrimLog API on the AWS cloud without having to manage servers.

Chosen – Industry standard for hosting applications and integrates well with ECS and ECR.

AWS Route53:

Purpose – Creates a domain for CrimLog to utilize on the cloud, allowing users to access the site on the Internet through a domain rather than connecting directly to an application load balancer.

Chosen – Industry standard for creating a domain within AWS. Additionally, this will allow CrimLog to easily implement a HealthCheck for the application.

AWS CloudWatch:

Purpose – Monitor the CrimLog application hosting in AWS.

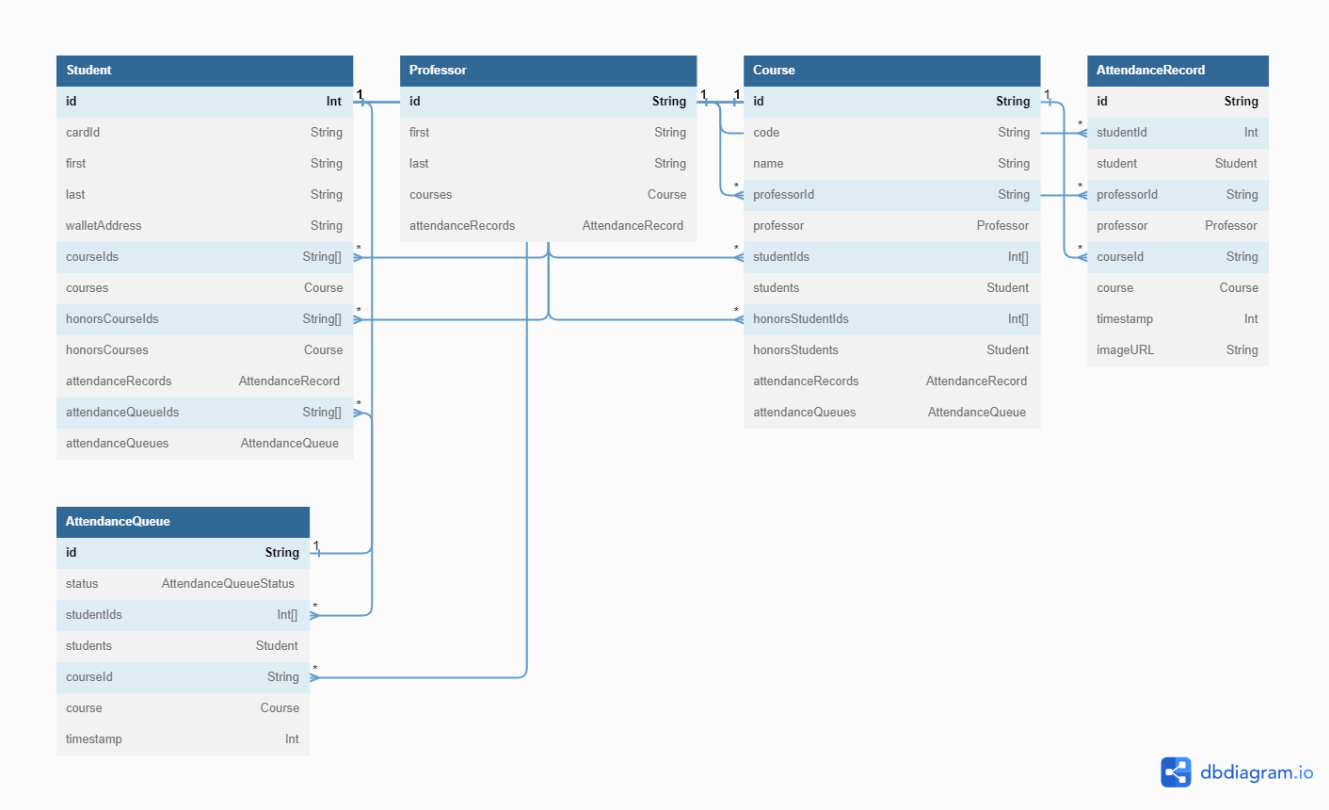
Chosen – Industry standard for monitoring application within AWS. Additionally, it is automatically integrated upon the creation of a Fargate service.

Proof of Concept

The proof of concept will be leveraging a Raspberry Pi, NFC card reader, NFC card, and NodeJS script to show CrimLog has proficiency utilizing NFC and connecting hardware and software components. The CrimLog proof of concept will take an NFC card input against the NFC card scanner, connected to the Raspberry Pi via GPIO pins. The Raspberry Pi will run a basic NodeJS script that will process the card input and display information based on the unique identifier (uid) on an NFC Card. This proof of concept demonstrates the connection between the hardware and software components, showing that an NFC card can be read and processed by a JavaScript script running on the Raspberry Pi. The information displayed based on the unique identifier on the NFC Card will be the unique identifier and the name of the cardholder.

Result: Currently, CrimLog can successfully read NFC card inputs and process the input, displaying the NFC card unique identifier. To complete the proof of concept, CrimLog will continually work on developing the output of displayed information based on the unique NFC Card identifier to display the name of the cardholder when an NFC card is tapped against the reader.

Database Schema Diagram:



For a larger view of the diagram, please visit: https://bit.ly/crimlog_mongodb_schema

CrimLog utilizes MongoDB for storing information. There will be 5 collections for CrimLog: AttendanceQueue, AttendanceRecord, Course, Professor, and Student. Documents in the AttendanceQueue will store information including the status of the queue (default is ACTIVE), the studentIds as an array to represent each student in the queue, the courseId of the queue's corresponding course, and a timestamp of when the queue was created. This will be used to store the attendance queue for each class, which is shown on the web application. The AttendanceRecord documents will store information including the studentId, the professorId, courseId, timestamp, and an imageURL. This will be a representation of the immutable records of all documented attendances. The Course documents will store information including the class code, class name, professorId, array of studentIds, and an array of honorsStudentIds. This will be utilized to store a record of information about the course and who should be attending the course, as well as whether those students are attending as honors. The Professor documents will store information about the professor, such as their first and last name. This will be used to show which professor is teaching which course and allows for professor information to be reused

across classes. The Student documents will store information including their NFC cardId, first name, last name, walletAddress, an array of courseIds, an array of honorsCourseIds, and an array of attendanceQueueIds. This will allow student information to include their name, their NFC card, their NFT wallet, all the courses they should be attending, which of those courses are honors, and a list of all the attendance queues they have ever been in.

Database DDL Scripts:

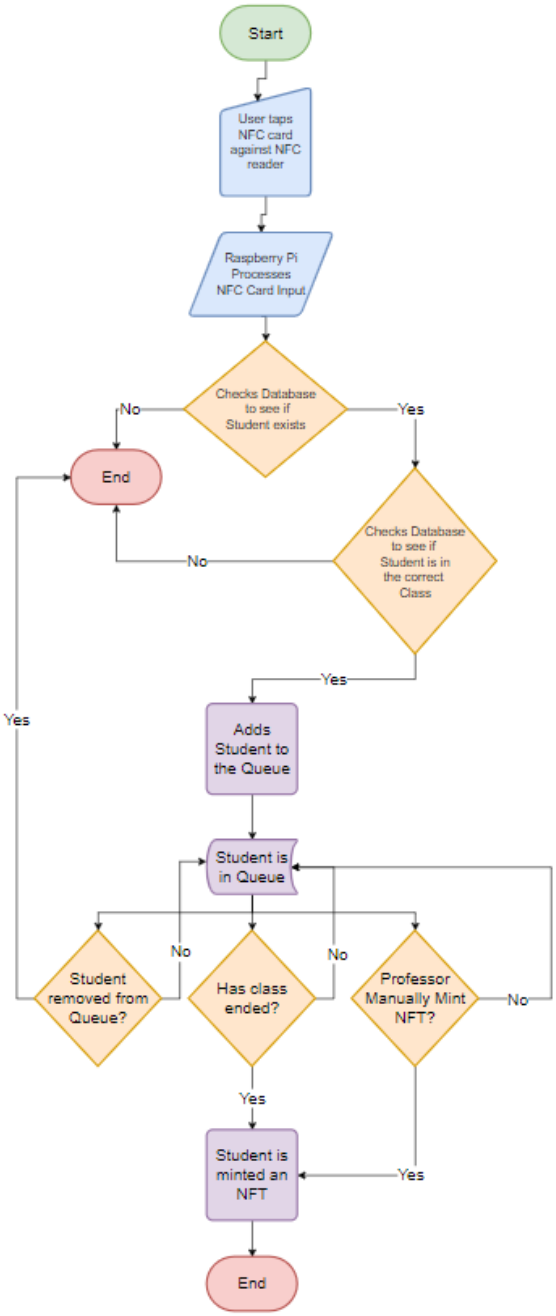
MongoDB does not have DDL scripts; therefore, this section is not applicable.

Database Prisma Schema:

The Prisma Schema is used to create the entire MongoDB database structure.
Link: https://github.com/crimlog/api/blob/dev/prisma/schema.prisma

Flow Charts/Process Flows:

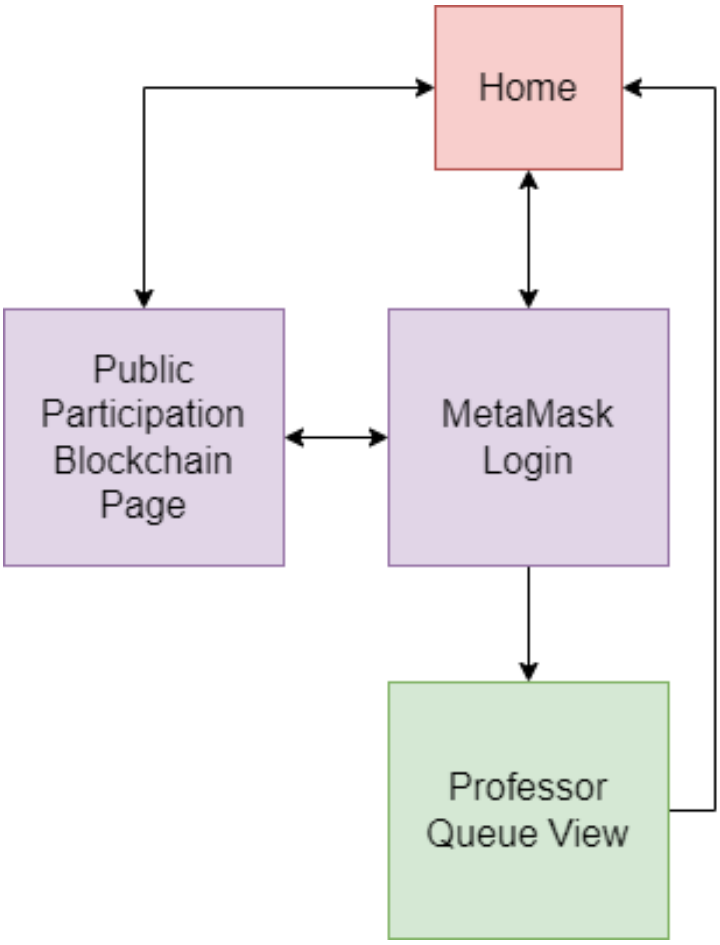
This flowchart provides a high-level overview of the flow of the CrimLog process for students from tapping the NFC card against the reader to receiving an NFT for participation.



For a larger view of the diagram, please visit: https://bit.ly/crimloft_flowchart_nfc-nft

Sitemap Diagram:

This sitemap outlines the pages on the Web Application.



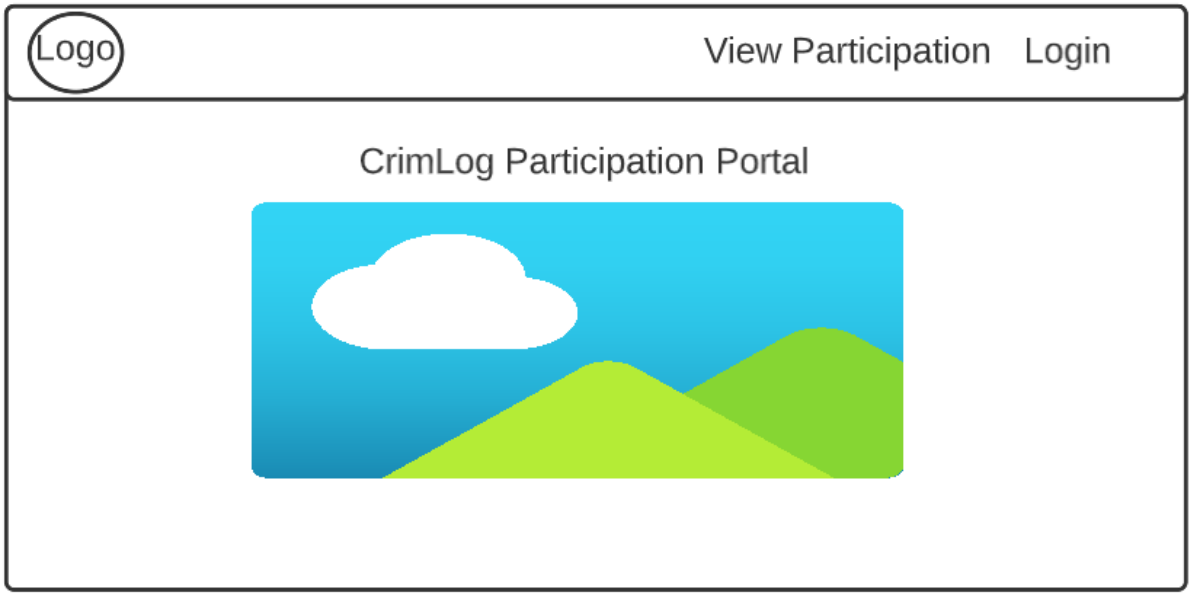
The Sitemap shows the different pages on the Web application, which will be made utilizing Svelte and Svelte Kit. This site will include a home page, which acts as the landing page for CrimLog. Additionally, there will be a public participation page, which displays all minted NFTs. This will show all public participation found on the block chain. Additionally, there will be a login aspect, which will be handled by an external authentication integration through the MetaMask browser extension. After logging in to the application through MetaMask, a professor may view the queue of students. This queue view will be a table that will show the students within a class. At this point, a professor may remove a student from the queue or mint a student an NFT to provide proof of attendance.

MetaMask as an authentication provider allows CrimLog to leverage decentralization, which is the foundational design fundamental that drives web3. This principle goes beyond cryptocurrency and blockchain technology as it relates directly to how regular users create, manage, and sign into their website accounts. Several of the flaws and inconveniences associated with traditional username/password management were resolved with the arrival of web2 and OAuth through third-party providers. One popular service could provide a single point-of-entry for all a user's accounts. Creating and trying to remember username/password credentials for dozens of sites was no longer a concern if they had a "Login with Google" button. However, as the internet continued to grow, so did several of the big technology companies. If Google is down, their API cannot be used as a third-party authentication provider, rendering the user attempting to login with OAuth locked out. In extreme cases, if Google were to terminate a particular user's account, every website that user had previously used their Google account to log into would be effectively inaccessible. The root cause of this problem is centralization.

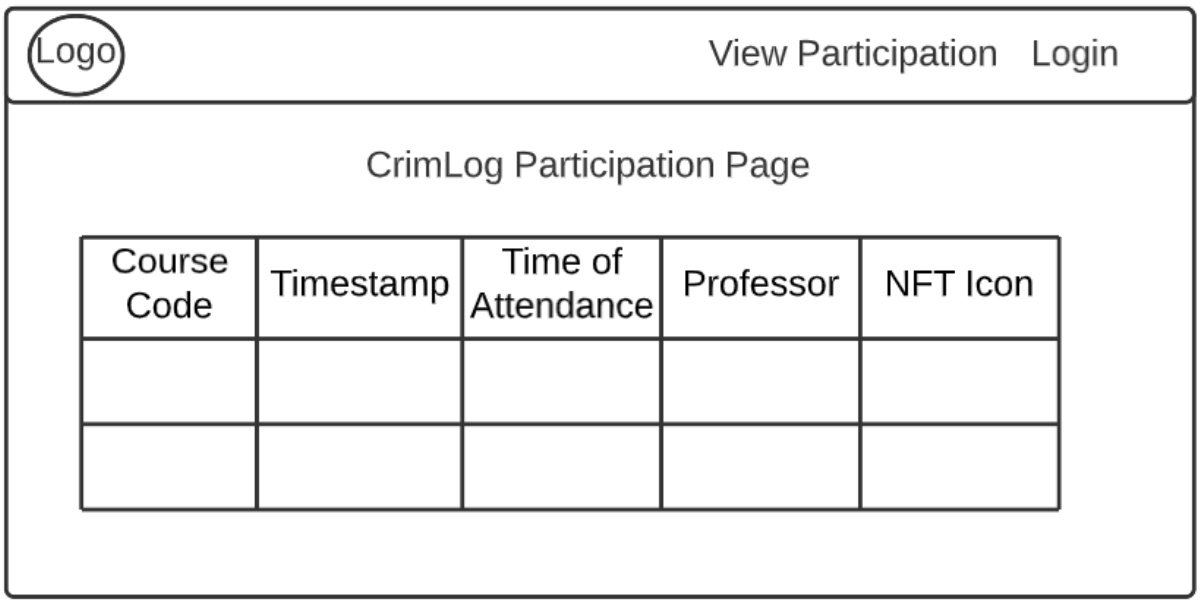
Web3 attempts to tackle this issue by restoring the user control of their credentials, like in web1, while maintaining the convenience of a single point-of-entry that web2 popularized. MetaMask is one such service that implements this solution in their software, but they are not the only ones that do this. MetaMask happens to be the most popular among web3 users, especially due to its low barrier of entry. CrimLog will be relying on MetaMask to provide a signed, cryptographic transaction from the client on which the MetaMask browser extension is installed. At a technical level, MetaMask uses asymmetric cryptography with digital signing to provide user-controlled, reliable authentication in the web3 space.

User Interface Diagrams:

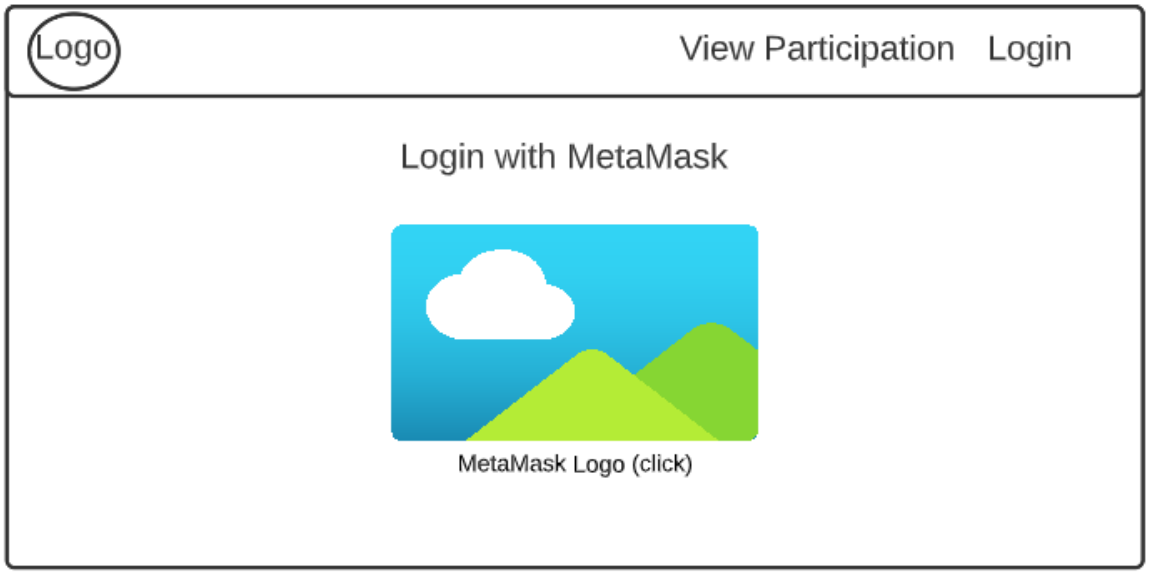
These wireframes provide a basic outline of the expected user interface of the Web Application, which will be made utilizing Svelte and Svelte Kit.



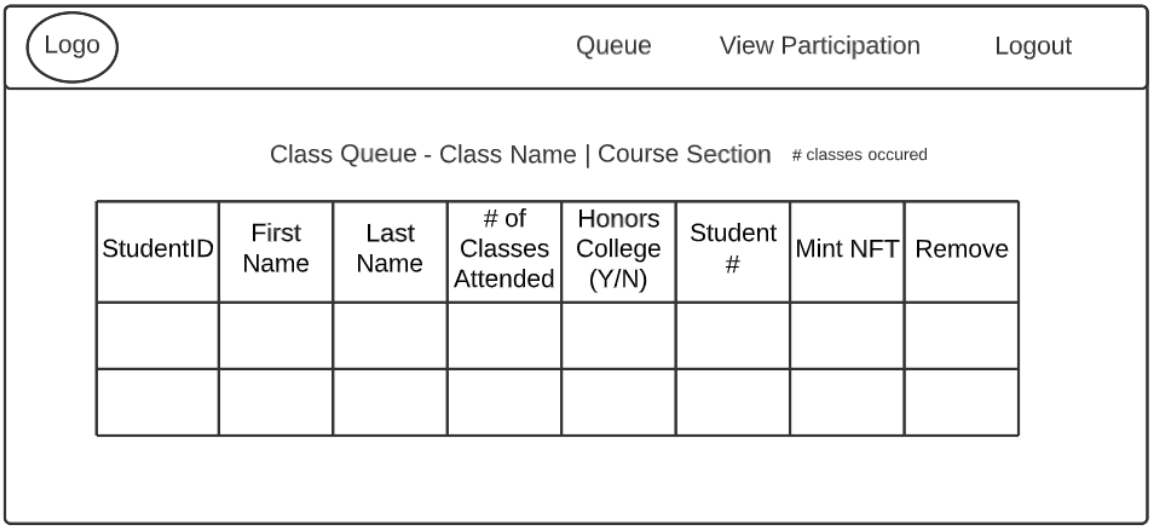
Home



Participation Blockchain Page



Login Page

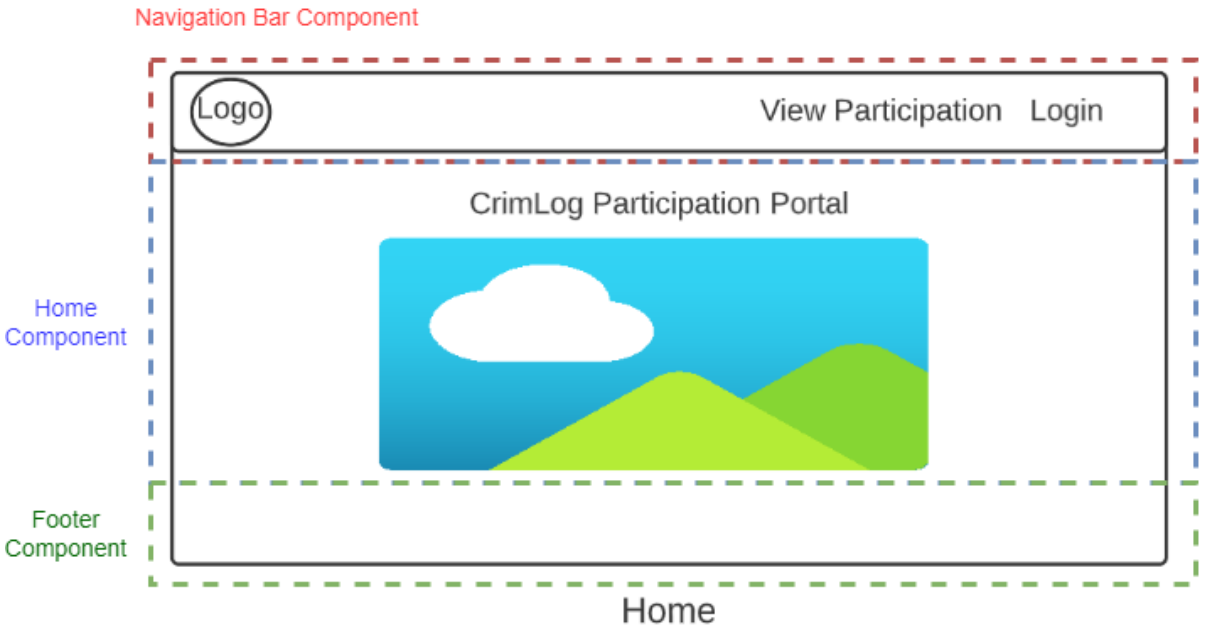


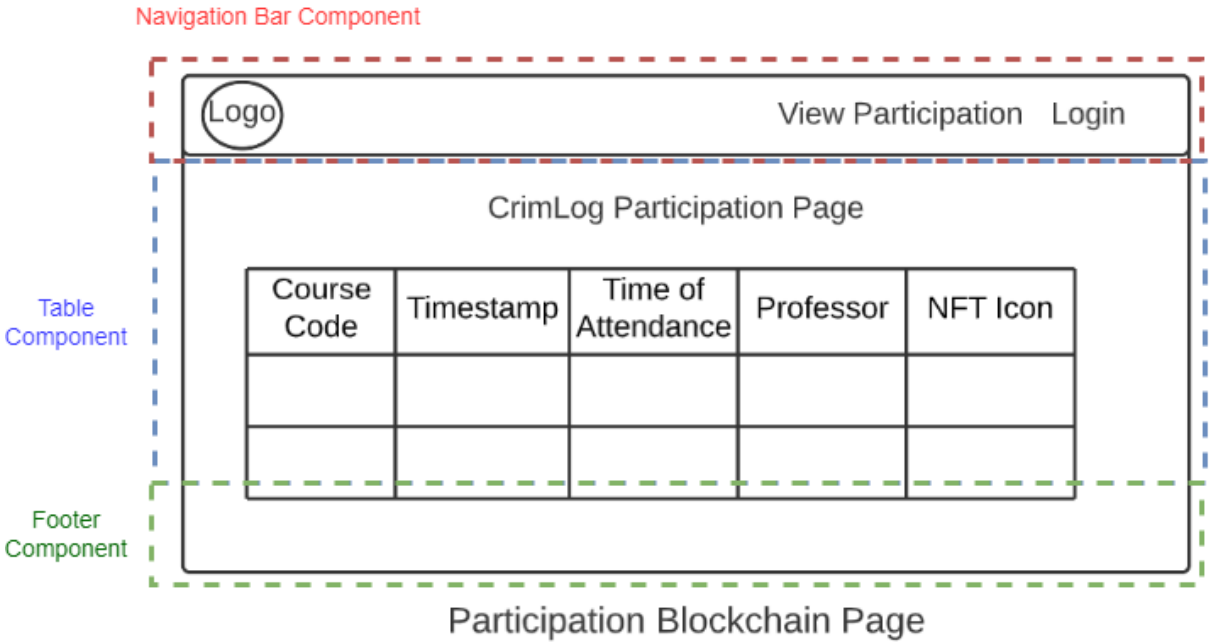
Professor Queue View (logged in)

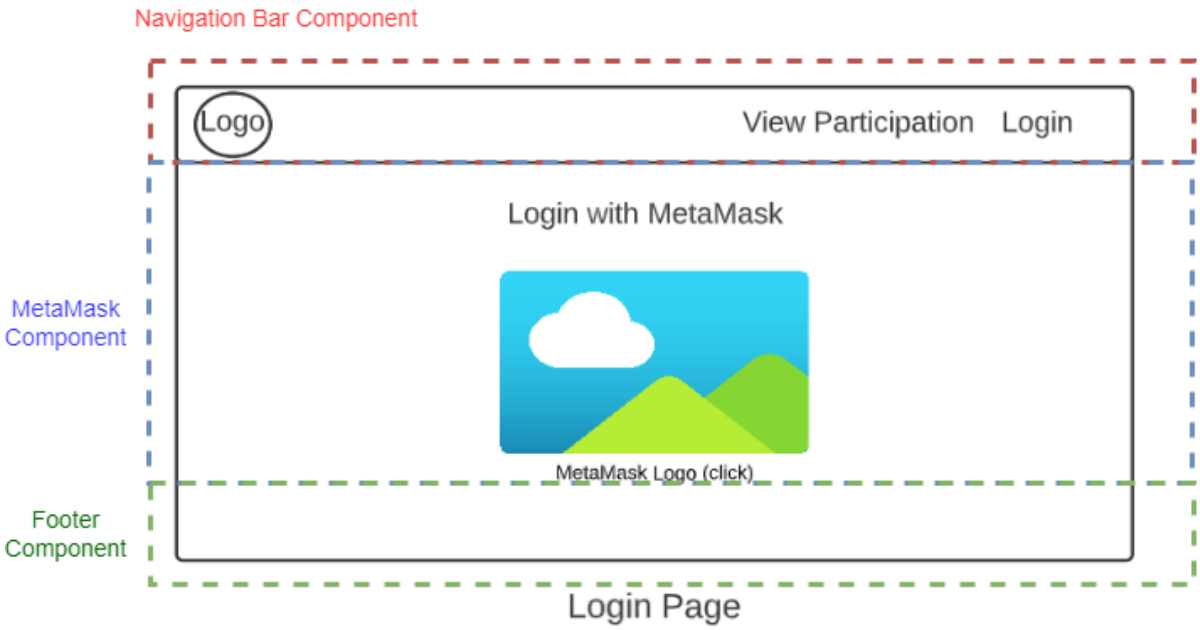
These pages give a basic display of the user interface for the web application for all the pages, which aligns with the sitemap. Every page listed in the sitemap has a respective wireframe, allowing developers to consult a frame of reference when implementing the Web application.

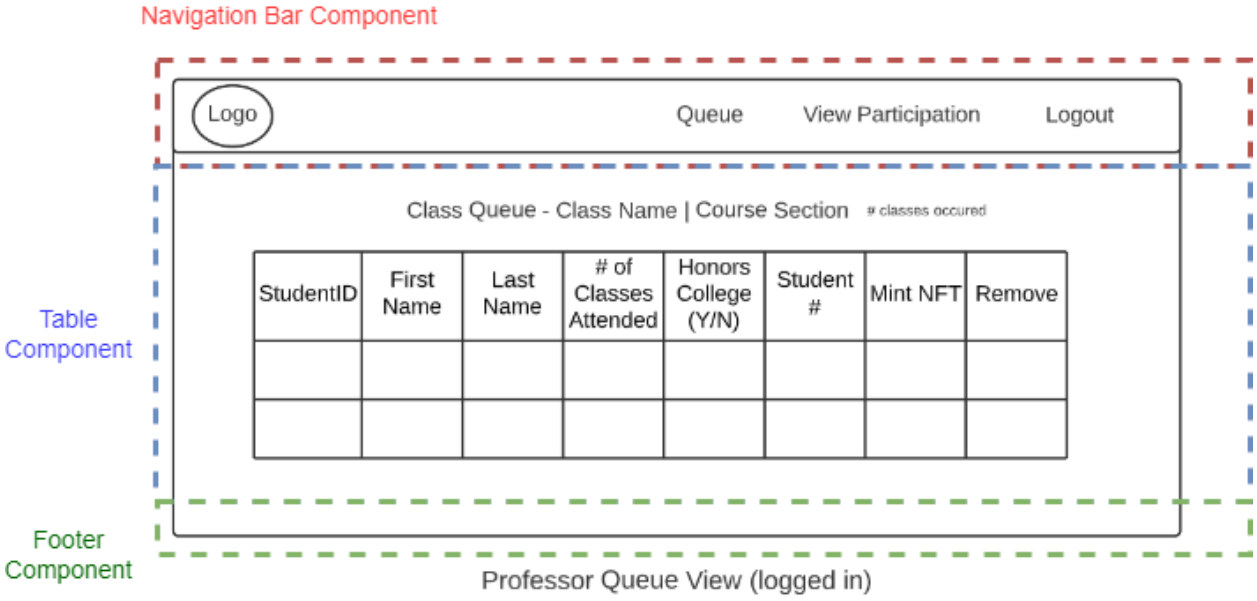
UML & Component Diagrams:

Below is a component diagram for the web application of CrimLog created in Svelte, breaking down the page to show individual components.





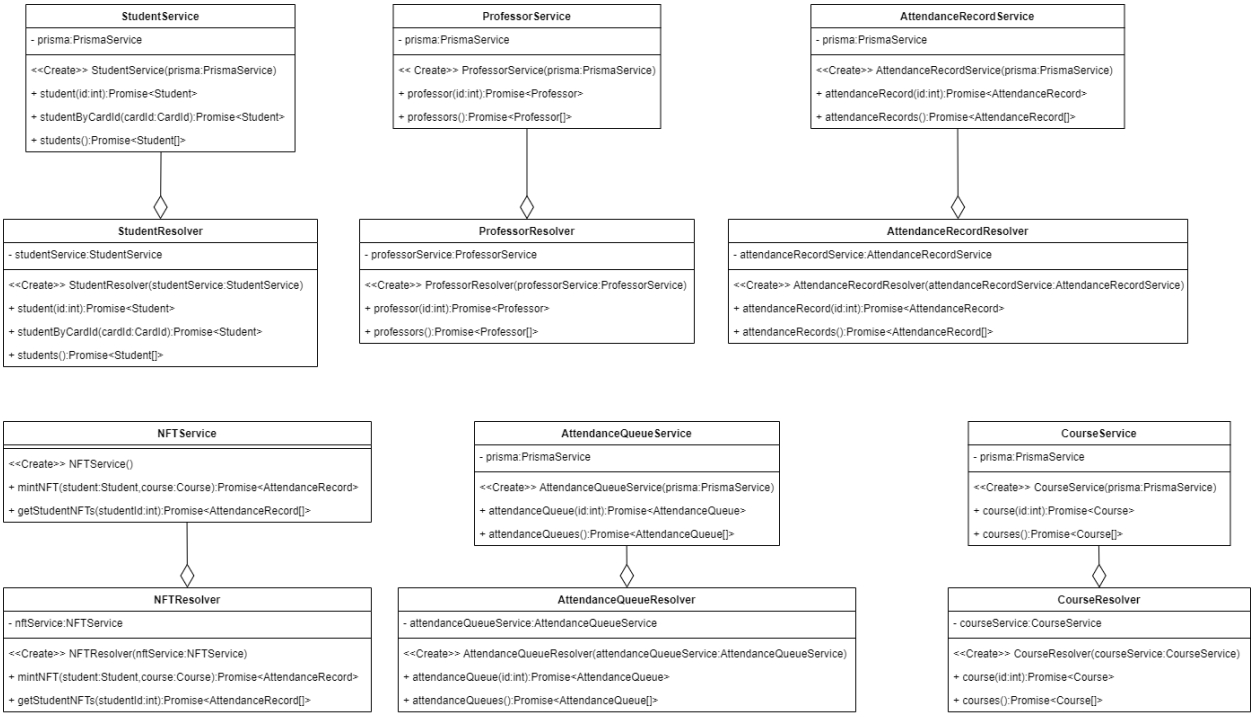




These diagrams highlight the different components utilized throughout the web application for CrimLog. These components include a navigation bar and footer component on each page. Additionally, there will be a home component, a MetaMask login component, and a table

component. The table component will be re-used for both the queue and participation page, allowing for re-usability.

Below is a UML diagram for the Nest.JS API, encompassing the business layer service objects for CrimLog.



For a larger view of the diagram, please visit: https://bit.ly/crimlog_uml

The CrimLog API leverages GraphQL, Nest.JS, and Prisma. This API allows for the retrieval and modification within the MongoDB database. This API includes models, services, and data access objects. Prisma is utilized to create the MongoDB database, as well as establish the connection between the database and the data access objects. Therefore, this UML outlines the 6 service classes created in Nest.JS, CourseService, ProfessorService, AttendanceRecordService, AttendanceQueueService, StudentService, and NFTService. Additionally, it outlines the 6 resolver classes, which include an instance of its respective service. Resolver classes contain resolver functions, which resolve a value for a type or field within a schema utilizing GraphQL. The 5 services connecting directly to the database, which are all services except the NFTService, utilize an injected instance of the PrismaService singleton to connect to the MongoDB database

and perform data retrieval and manipulation. These methods are also outlined below within the CrimLog Service API Design documentation and are the exposed endpoints of the API.

Service API Design:

The internal Service API design for CrimLog is built upon the GraphQL API Specification. Full documentation for the internal codebase is found here: https://bit.ly/crimlog_apidesign_graphql

Additionally, this is linked as an external document in the Deliverable Acceptance Log chart.

NFR's (Security Design, etc.):

CrimLog is taking a revolutionary approach by turning the classic question of "how does the design support the non-functional requirements?" on its head, and instead asking "how do the non-functional requirements support the design?" This allows the project to model its design separately from the NFRs. After the design has been completed, the non-functional requirements will be constructed around the already established design, in a way the best supports said design. This design-first principle is what drives the CrimLog project development lifecycle and yields concrete, robust results.

CrimLog will feature a non-functional requirement of documentation for the Public GraphQL API. This documentation will be extensive and detailed, sharing insight to assist other companies in implementing the service. This ties in with CrimLog's out of scope features of allowing other colleges to utilize the participation software solution and linking the minted NFT to learning management software to automatically provide points for participation. Beyond the GraphQL API design, CrimLog will implement JSDoc documentation standards throughout the internal codebase. Additionally, instructions within the open source CrimLog repository will provide steps for implementation within a markdown file.

Operational Support Design:

Monitoring will be set up exclusively through the AWS Management Console for ease of use. CrimLog will leverage an in-house custom logging library to input logging information, warning, and errors into CloudWatch. Each endpoint will log its processes, allowing CrimLog developers to trace process flow and detect errors and bottlenecks as they occur. Additionally, the process flow between the layers from data access to the business service layer API will log information, allowing developers to detect errors and provide insight when making bug fixes.

For logging, CrimLog will leverage AWS CloudWatch. All logs will be logged into CloudWatch, including error output for troubleshooting. Additionally, CloudWatch will also act as CrimLog's monitoring system. A HealthCheck endpoint will be monitored by CloudWatch, linked to the Fargate instance. When the HealthCheck fails, AWS Simple Notification Service (SNS) will alert CrimLog of the outage, allowing CrimLog to monitor uptime. Therefore, AWS CloudWatch will be leveraged for monitoring and logging within the CrimLog design.

Other Documentation:

N/A

No further support documentation is required, as CrimLog was able to successfully document prevalent data relating to the development of the product within the resources and information presented throughout the entirety of the Design Report.

Meeting & Brainstorm Session – November 3, 2022

CrimLog held a team meeting on November 3rd. During this meeting, the team discussed the logical implementation of the software and hardware architecture. To ensure program efficiency, CrimLog ensured the team was following KICKit principles. Additionally, the team discussed and began testing the POC and plans for the capstone showcase.

Meeting & Brainstorm Session – November 15, 2022

CrimLog communicated asynchronously to discuss diagrams and other aspects of the Design Report. Additionally, information and preparation for the peer review presentation were discussed and reviewed.

Appendices

Appendix A – Technical Issue and Risk Log

Issues and Risk Log								
Issue or Risk	**Description**	**Project Impact**	**Action Plan/Resolution**	**Owner**	**Importance**	**Date Entered**	**Date to Review**	**Date Resolved**
R	Creating auto generated custom NFT icons	Without auto generated custom NFT icons, CrimLog will be unable to mint easily visually distinguishable NFTs for participation	To mitigate this issue, CrimLog will ensure the table has distinguishable information that makes the custom NFT icons less necessary for distinguishing NFTs	Melanie	High	2022-11-16	2022-01-25	
R	Connecting Monitoring and Logging for the Web Application and API into CloudWatch	Without proper monitoring and logging, it will take longer for developers at CrimLog to detect errors and find solutions. This could cause the project to fall behind schedule.	To mitigate this issue, CrimLog will research connecting Svelte Application logging into CloudWatch. Additionally, CrimLog will research other logging providers such as Splunk.	Elijah	High	2022-11-16	2022-01- 11	

Appendix B – References

This section is not applicable because no external references were utilized throughout the creation of this Design Report.

Appendix C – External Resources

GIT URL:	https://github.com/crimlog/api
Hosting URL:	Not applicable

