| A1  Group 02  Abdalrahman Afifi  Hunter Lavender  Travis Mueller  Zachary Wildasin |
| --- |

# **Table of Contents**

[**Table of Contents**](#_igwbbtspgrqd) **1**

[**Vision**](#_612a8e46lhbd) **3**

[**Executive Summary**](#_l269g9gcls78) **3**

[**Extended Problem Statement**](#_6jbbxaxd0y5d) **3**

[**Requirements Specification**](#_s7rg3nuk30r1) **4**

[**Functional Requirements**](#_e1sww41fcvx1) **4**

[**Marketing Requirements**](#_8khiyhpewvys) **4**

[**Engineering Requirements**](#_p2efz9xncd7l) **5**

[**Marketing and Engineering Requirements Map**](#_aejiignkmu4b) **5**

[**Marketing or Engineering Requirements trade off chart**](#_q8nbfdvympox) **6**

[**Trade off chart between Engineering Requirements**](#_ov1qpq3z8t7p) **7**

[**Competitive Benchmarks**](#_17abzwrapuqb) **8**

[**Constraints and Standards**](#_4flvy1cgnrzm) **10**

[**Security**](#_ey69q4mcknd7) **10**

[**Reliability**](#_cdltbj3rw7h) **11**

[**Sustainability**](#_2hvaatdm7gz) **11**

[**Background Research**](#_cd4pxljejr3d) **11**

[Frontend](#_w4dewac1nzvp) 11

[Backend](#_2sslft27t9dl) 12

[**References**](#_3jg75anh6wo) **12**

[**Member Contribution Table**](#_th8slglhz8tr) **13**

# 

# **Vision**

For this capstone project our group is going to continue working on the The Platform for Intelligent Computer Algorithm Research and Development (PICARD) project that the Group 11 from the previous semester already started. PICARD sets out to enable research, development, and performance testing of distributed semi-supervised machine learning algorithms for classification in big data sets with severely imbalanced data and high labeling costs.

A requirement for the PICARD is to create the Fullstack. We also have to implement a method to efficiently clone the system on a different server. Then we have to make documentation to provide baseline security measures to be used across WVU networks.

# **Executive Summary**

This document defines the overall goals and requirements for the PICARD project. The requirements for the project are broken down into three fundamental categories: functional, marketing, and engineering requirements.

The functional requirements state what the user should be able to do with ease for the project to be considered operational. For example, our project must allow the user to setup, destroy, and modify nodes in an environment via user interface. The nodes in an environment is a system with defined resources. The nodes must be able to perform these basic tasks to be considered functional.

Marketing requirements include not only the concepts to sell the product, but the experience we can advertise to the users. Such as, the user experience and how the PICARD application compares to its competitors. The price point is another important aspect for the marketing requirements, and being able to label PICARD as open source.

The final requirements for PICARD are the engineering requirements. An example of an engineering requirement is being able to run the algorithms on a node, and being able to aggregate the results from the node.

# **Extended Problem Statement**

Dr. Tom Devine of West Virginia University is at the head of the PICARD’s development, a research platform built around his own personal server. PICARD aims to advance the testing and improvement of cutting-edge semi-supervised, distributed machine learning algorithms. To accomplish this, it involves establishing a virtual cluster of machines or containers that will simultaneously run the AI algorithm, dividing tasks among them for efficient and dependable processing of large data sets. Furthermore, we’re going to make sure that our framework is as secure as possible from basic vulnerabilities since it’s going to be free open source for users, we don't want them to be scared or think that it’s not secure enough.

The PICARD system is set to have a new frontend graphical user interface, most likely web-based. This allows for easy, secure, remote control and monitoring during operation. The interface will also allow for adjustments to the number of nodes running the algorithm and how they will divide the server’s resources. The visualizations created aim to provide the user with the ability to keep track of the hardware usage statistics of each node, significant findings, data analysis after completion, and real-time updates on the data and algorithm status.

Security is the backbone of our project. PICARD will be utilized to scan transaction data for cybersecurity breaches, allowing clients to identify and patch up their system's vulnerabilities. It is crucial to preserve the confidentiality and security of client data, as it can be used to exploit the weaknesses their system may have and trigger future cyber attacks. Hence, every aspect of the system, including the server, front-end interface, cluster manager, and each node must be securely configured and kept secure at all times, which will be done through the use of automated tools. To emphasize the importance of security, we will create a university lab server IT policy, aiming to standardize and enhance computer security across university labs while adhering to WVU ITS standards.

To meet these requirements, we are exploring both commercial and free and open source (FOSS) options to find the optimal balance between cost, functionality, user-friendliness, and performance. Our possible solutions will be evaluated based on the trade-off between requirements and comparative benchmarks, leading to a final decision. Afterwards, we will proceed to implement and test the chosen solution on the actual hardware and with diverse data sets.

# **Requirements Specification**

## **Functional Requirements**

1. Create the Fullstack for the PICARD.
2. Have user authentication using two factor authentication.
3. Make sure the website is using the HTTPS protocol for security
4. Be able to create nodes
5. Be able to delete nodes
6. Be able to create jobs through a UI instead of command line

## **Marketing Requirements**

1. Fullstack: Make an intuitive and accessible graphical user interface so that users can submit programs, and datasets, as well as start, cancel, and monitor Jobs rather than making them use the command line interface which isn’t user friendly to do all the aforementioned tasks. Create a backend to process the data on the server itself and connect the front and backend.
2. Extra Test Cases: Develop web application test case scenarios such as SQL injection buffer overflow, etc making sure that the website isn’t easy to hack and save from the day to day vulnerabilities.
3. Affordable: The Software should be minimal to no cost as to allow the cloning of this project easily, to be used by others.
4. Dependable: The software should be dependable and be capable of staying up most of the time.
5. Visualization: The software must be equipped with the ability to display data visualization and provide information on the hardware components of the machines where the algorithm operates.

## **Engineering Requirements**

1. Set up: Set up a distributed system with user defined nodes and resources.
2. Distribute: Distribute test data to the nodes.
3. Select Algorithm: Select algorithm to test with the user defined parameters from the node.
4. Run: the distributed algorithm on the simulated distributed system.
5. Aggregate: the results
6. Visualization: of the results
7. Delete: Be able to destroy the distributed system and either spin up a new one(or repeat previous steps)
8. Compare: Be able to compare the results from multiple nodes.
9. Efficient: The software should be fast and efficient, capable of executing quickly and with minimal extra resources.
10. OPENSCAP: The software must meet West Virginia University's and OPENSCAP's security standards to ensure secure operation within the university's environment.
11. Secure: Every node created must be secure to maintain the confidentiality of client data and prevent exploitation by malicious entities.
12. Maintainable: The software must be easily maintainable by a limited team after initial development, to ensure its continued efficiency and security throughout its life cycle.
13. Organized: The process of creating and utilizing the project should be thoroughly documented to allow future students to update, utilize, and expand upon it.

# 

# **Marketing and Engineering Requirements Map**

| Engineering Requirement | Marketing Requirement | Justification |
| --- | --- | --- |
| 1,9 | 3 | Our project must balance a solution that maintains open source software that is as affordable as possible and at the same time runs fast and efficiently. |
| 1 | 1 | The user should have the ability to execute the algorithm on their own data and observe the outcome through a graphical user interface, without the need for command line interaction. |
| 10,11,12,13 | 4 | To ensure the project's reliability, it is crucial to document the steps for its operation and use a medium with sustained support for running the nodes. Additionally, the project's security must be a top priority to avoid any potential rejection or failure. Hence, it is essential to secure both the nodes and the server. |

# **Marketing or Engineering Requirements trade off chart**

### 

|  | | Engineering Requirements | | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Marketing Reqs. | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |

| Table color key | |
| --- | --- |
|  | Positive correlation (requirements are mutually beneficial or fulfilling one naturally fulfills the other) |
|  | Negative correlation (requirements restrict each other or fulfilling one naturally makes it more difficult to fulfill the other) |
|  | Mixed correlation (requirements strongly influence each other but neither in a strictly positive nor strictly negative way; the way in which one requirement is fulfilled may require special consideration for the other; different fulfillment strategies for one will likely either benefit or hinder the other) |
|  | No relation between requirements |

# 

# **Trade off chart between Engineering Requirements**

|  | Setup | Distribute | Select Algorithm | Run | Aggregate | Visualization | Delete | Compare | Efficient | OPENSCAP | Secure | Maintainable | Organized |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Setup |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Distribute |  | |  |  |  |  |  |  |  |  |  |  |  |
| Select Algorithm |  | | |  |  |  |  |  |  |  |  |  |  |
| Run |  | | | |  |  |  |  |  |  |  |  |  |
| Aggregate |  | | | | |  |  |  |  |  |  |  |  |
| Visualization |  | | | | | |  |  |  |  |  |  |  |
| Delete |  | | | | | | |  |  |  |  |  |  |
| Compare |  | | | | | | | |  |  |  |  |  |
| Efficient |  | | | | | | | | |  |  |  |  |
| OPENSCAP |  | | | | | | | | | |  |  |  |
| Secure |  | | | | | | | | | | |  |  |
| Maintainable |  | | | | | | | | | | | |  |
| Organized |  | | | | | | | | | | | | |

# 

| Table color key | |
| --- | --- |
|  | Positive correlation (requirements are mutually beneficial or fulfilling one naturally fulfills the other) |
|  | Negative correlation (requirements restrict each other or fulfilling one naturally makes it more difficult to fulfill the other) |
|  | Mixed correlation (requirements strongly influence each other but neither in a strictly positive nor strictly negative way; the way in which one requirement is fulfilled may require special consideration for the other; different fulfillment strategies for one will likely either benefit or hinder the other) |
|  | No relation between requirements |

# 

# **Competitive Benchmarks**

In this section we’ll compare some products that are similar to the one we’re working on to see what went right and wrong with their product so we can not repeat their mistakes, make a better product and try to see their designs so we have an idea on how to build ours.

**CentOS 7**

CentOS 7 is a free open source operating system that's provided by Redhat Linux which can be used by beginners easily and safely.

**Pros:**

* Applications don't need to be updated often.
* Reliable and scalable.
* User friendly.
* Secure.
* Reliable.

**Cons:**

* It should be made ready for bigger enterprises.
* It should be faster for booting.
* It should be viable for daily use.
* Provide already tested updates to users.
* Don't use the users as guinea pigs for testing.

**Microsoft OneDrive**

Onedrive is one of the popular cloud storage and file sharing services for any windows user that’s been out there for so long.

**Pros:**

* Excellent interface**.**
* Clients for Android, iOS, Mac, and Windows.
* Well integrated with Windows and Microsoft 365.
* Strong online photo presentation and management.
* Powerful file-sharing and collaborative editing.

**Cons:**

* Less free storage than some competitors.
* Doesn't back up all folders on drive.

**IDrive**

IDrive is another cloud storage and file sharing service that has different benefits from OneDrive making sure it protects your data through regular backups.

**Pros:**

* Easy setup.
* Unlimited devices per account.
* Free local backup.
* Fully encrypted.
* Fast upload speeds.
* Excellent value.

**Cons:**

* Storage isn't unlimited.
* Limited Linux support.
* Complete disk image backup only for Windows.

**Google Drive**

Google Drive is the most known in cloud storage due to the fact that most of the people have a gmail account so therefore they get access to Google Drive right away.

**Pros:**

* Generous free storage space.
* Excellent productivity-suite collaboration.
* Includes desktop-to-desktop file syncing.
* Many third-party integrations.
* Cross-platform apps.

**Cons:**

* No password-protection for shared files.
* Mobile apps could do more; multiple apps required for all related functions.
* Some privacy concerns.

**SpiderOak One Backup**

SpiderOak One Backup is another competitive cloud storage but much different then all the other products that was mentioned since it focuses more on privacy and security.

**Pros:**

* Strong privacy features.
* Supports an unlimited number of computers per account.
* Excellent versioning capabilities.
* Includes file-sharing and folder-syncing options.
* Well-designed, full-featured desktop application.

**Cons:**

* Lacks multi-factor authentication option for web logins.
* No longer offers mobile apps.

# **Constraints and Standards**

This section outlines the constraints and standards that must be followed by our project.

# **Security**

The most important and most challenging requirement to meet is that our project must be completely secure. In order to be allowed to run inside West Virginia University’s network, both the server and all of the nodes need to be completely secure. After we add new test cases, notify WVU IT Department to do extra tests to make sure it is secure and follows the rules under the WVU IT Department standards. The team will also make the server and each node compliant to OpenSCAP security standards.

# **Reliability**

This software will run with large data sets, and thus require a great deal of time to run so we will need to make sure the the server is able to stay up and run the majority of time so we will need to secure the server from DDOS attacks and have it check for the numbers of users and establish a queue for waiting users.

# **Sustainability**

The system we use to host and secure our nodes should hold up over time as new updates and security vulnerabilities arise. The software must be well documented and built with a modular approach so if a new, more efficient solution comes out, or if anything is no longer supported, something new can be swapped in with as little headaches as possible. We can also try to build a connection from our software with CVEs so that security is always up to date.

# **Background Research**

## **Frontend**

For the project we must pick a framework that will provide a framework for the GUI. There are quite a few competitors in the market and there are pros and cons to all of them. When comparing Javascript to AngularJS the choice isn’t hard to make. The article from *GeeksforGeeks* highlights a lot of topics and how they have evolved with each other over time. AngularJS for the time being is going to be the easiest/better one to use out of the two. AngularJS is a great way to provide a model view controller design, and is founded by Google so the support using AngularJS is a huge pro.

AngularJS is not just one language, there are quite a few different languages for the GUI and understanding the differences between everything is important. Hyper Text Markup Language (HTML) is very basic and can provide the basic structure of the website. Javascript being implemented is what makes the GUI more advanced and will be able to implement dynamic pages.

AngularJS is not the only framework with Angular. The other option is just called Angular; however, despite the similar names they are significantly different from each other. The biggest difference is for Angular it uses type script instead of javascript. Typescript is a product via microsoft, and utilizing Angular allows you to have a component-based design. Being created by another very popular company the support/documentation is going to be good as well. The biggest pro with Angular is that it is not just supported for desktop, but also you can design for mobile devices as well.

Another framework for creating user interfaces is ReactJS. ReactJS utilizes NodeJS for the backend. ReactJS is backed by another prominent company Facebook, and is considered a Free and Open Source Software. Using ReactJS is great for UI, but also has very complex architecture that can become hard to manage. ReactJS also has a really hard time implementing microservices and is not the most scalable platform. ReactJS is great for creating single page applications, and can have an interactive user interface. ReactJS allows for the pages to be reusable and you can implement logic into them as well.

## **Backend**

The application is full-stack, so we must pick a backend for the project as well. According to *GeeksforGeeks*, one of the most popular backend frameworks is Django. This backend framework is open-source and is considered to be one of the most scalable and customizable frameworks. Django gets praised for the code reusability and being used on the DRY code principle. The DRY code principle focuses on reusability, efficiency, and maintainability. Django is used by several popular applications such as Google, Spotify, DropBox, and Mozilla. One of the many reasons for large web applications utilizing Django is being able to perform CRUD (Create, Read, Update, Delete) operations at a high speed. Django uses Python thus allowing easy integration with third party applications. Another feature is being able to use its very own built in user authentication.

Another open-source framework is Laravel, which follows the MVC(Model-View-Architecture) using PHP. Laravel consists of extensive libraries and API support; however, it is mainly used for e-commerce websites and news. This complete server side framework is solely responsible for data manipulation.

Next, we have another open-source web application framework called Express. Which is a NodeJS based framework, and supports compatibility with REST-API applications. The popularity of Express is due to its flexibility. Express allows for you to integrate with NoSQL, MongoDB, and NodeJS to handle requests.

Similarly, Flask is another open-source, lightweight framework using Python for web applications. It is a framework that is considered “beginner-friendly” with no dependencies. Due to Flask having no Dependencies, it can be referred to as a micro framework. Flask is still used by many due to its support for object-relational mapping, uploading, and validating. It can be used for large-scale applications to provide rapid development.

# **References**

“Difference between Angular and Angularjs.” *GeeksforGeeks*, GeeksforGeeks, 30 June 2022, https://www.geeksforgeeks.org/difference-between-angular-and-angularjs/.

“Difference between JavaScript and Angularjs.” *GeeksforGeeks*, GeeksforGeeks, 17 Jan. 2023, https://www.geeksforgeeks.org/difference-between-javascript-and-angularjs/.

“Difference between JavaScript and HTML.” *GeeksforGeeks*, GeeksforGeeks, 25 May 2022, https://www.geeksforgeeks.org/difference-between-javascript-and-html/.

“Difference between Node.js and React.js.” *GeeksforGeeks*, GeeksforGeeks, 31 May 2022, https://www.geeksforgeeks.org/difference-between-node-js-and-react-js/.

Muchmore, Michael, and Jill Duffy. “The Best Cloud Storage and File-Sharing Services for 2023.” *PCMAG*, PCMag, 3 Feb. 2023, https://www.pcmag.com/picks/the-best-cloud-storage-and-file-sharing-services?test\_uuid=05n7gTzbSo0Sh5pVEDljnCi&test\_variant=a.

“Top 7 Frameworks for Backend Development.” *GeeksforGeeks*, GeeksforGeeks, 10 June 2022, https://www.geeksforgeeks.org/top-7-frameworks-for-backend-development/.

# **Member Contribution Table**

Class: CSEE480 C01

Group Number: 02

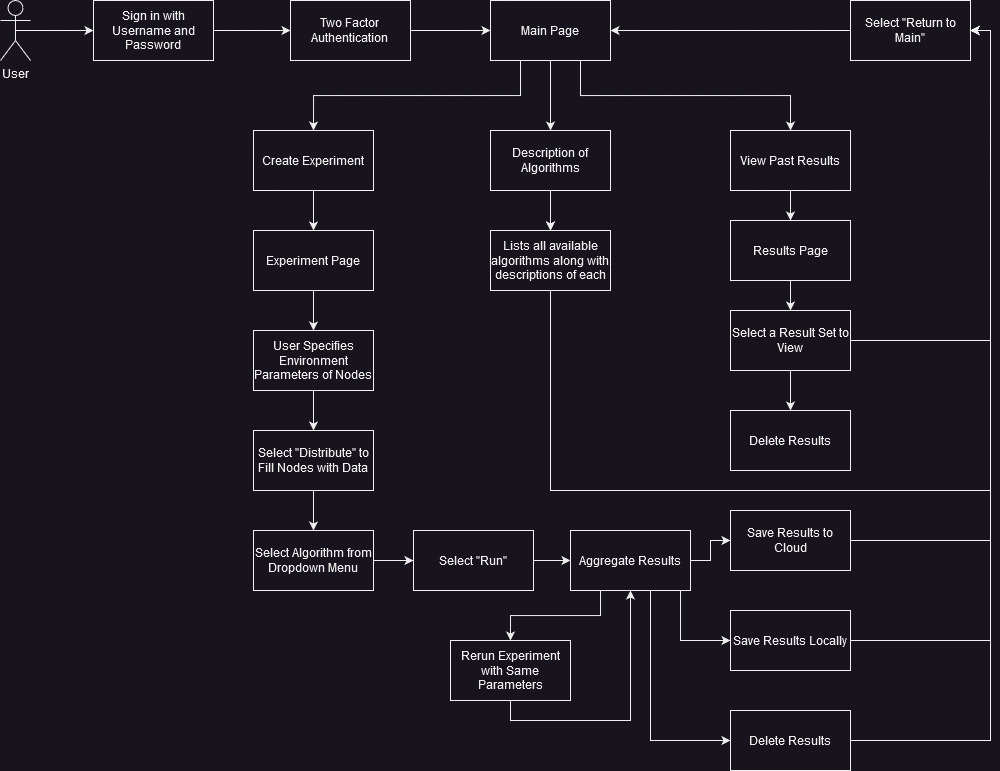
Assignment Name: A1

| Member Name | Meetings attended | Meetings missed | Contribution | Stuff done by member |
| --- | --- | --- | --- | --- |
| Abdalrahman Afifi | 3 | 0 | 100 | Vision, editing,marketing requirements, engineering requirements, competitive benchmarks, trade off charts |
| Hunter Lavender | 3 | 0 | 100 | Functional Requirements,  Engineering Requirements,Executive Summary  ,Background Research, References |
| Travis Mueller | 3 | 0 | 100 | Vision,  Marketing requirements, security, sustainability, marketing and engineering requirements map |
| Zachary Wildasin | 3 | 0 | 100 | Executive Summary, Extended Problem Statement  Marketing/Engineering Requirements Trade Off Chart |

# 

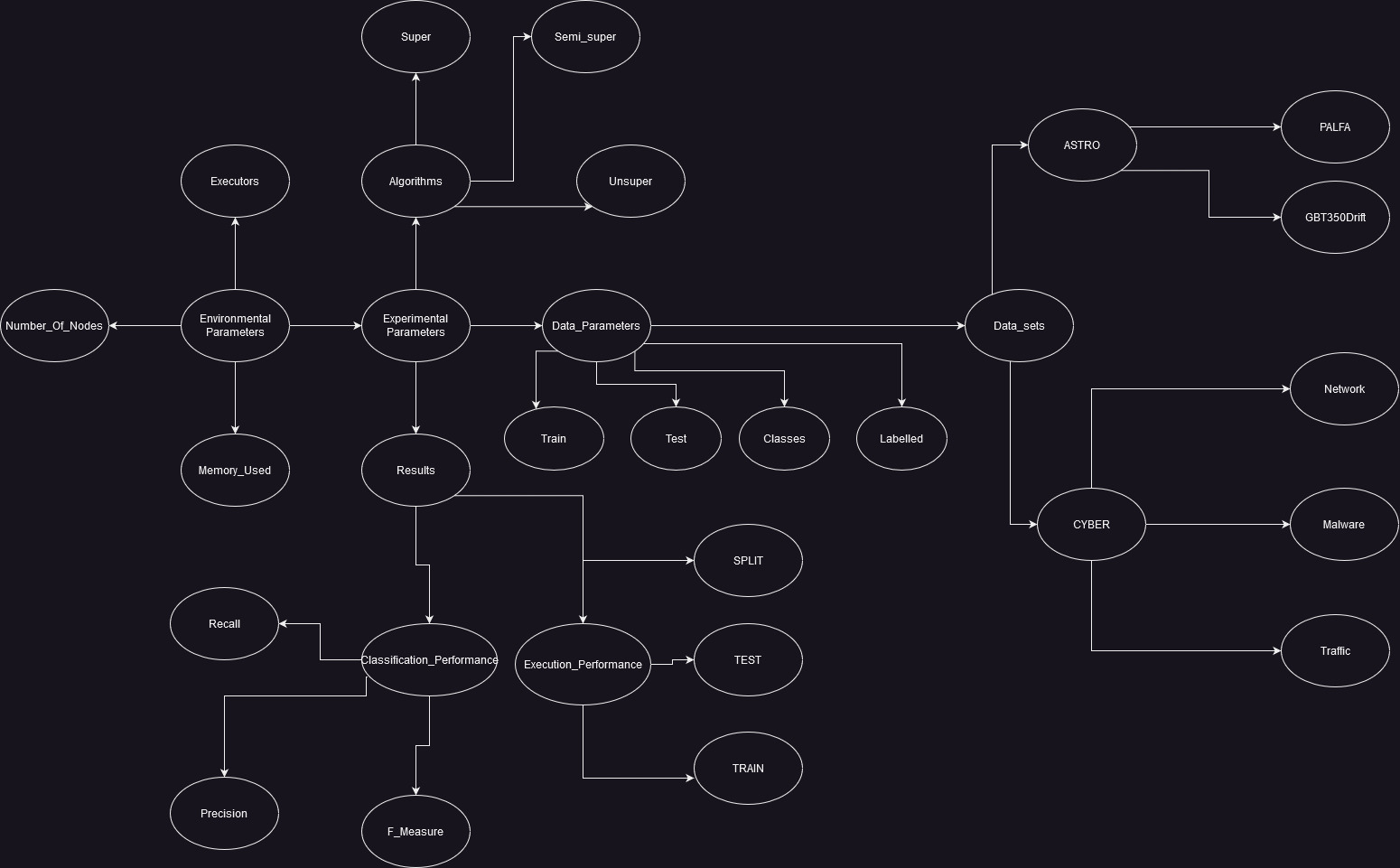
| A2  Group 02  Abdalrahman Afifi  Hunter Lavender  Travis Mueller  Zachary Wildasin |
| --- |

# **GUI Diagram**



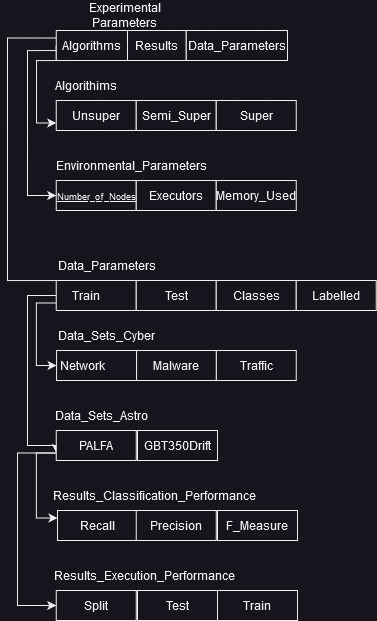
This is the frontend diagram of the PICARD and how it will operate.

# **ER Diagram**



This is a diagram explaining the backend in the PICARD

# **RM Schema**



This the RM schema for PICARD which shows how the database will be built

# 

# 

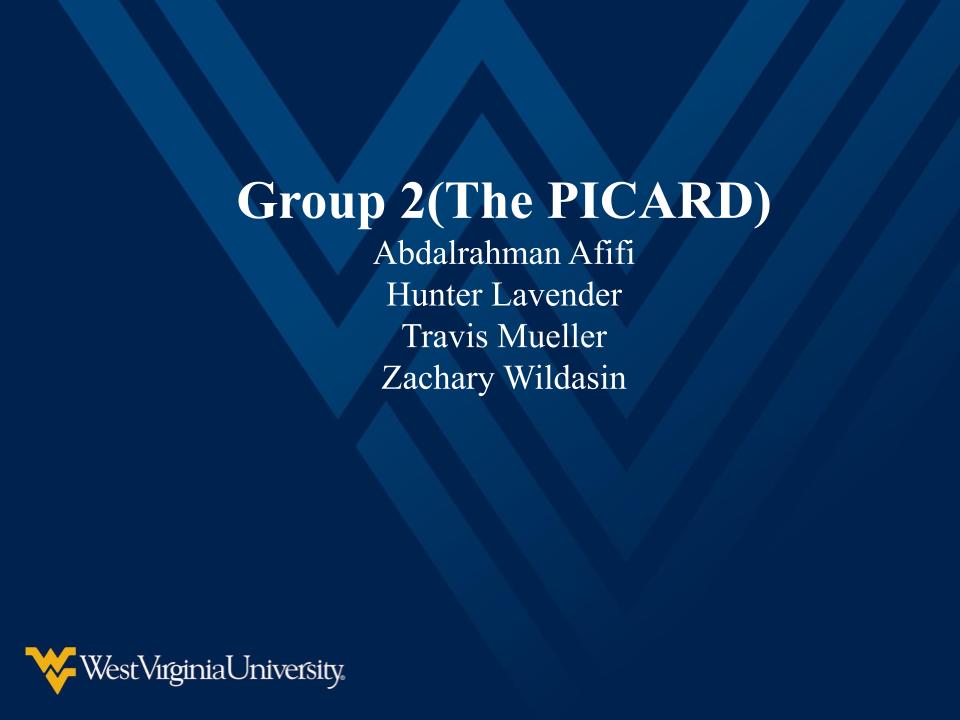
# **Member Contribution Table**

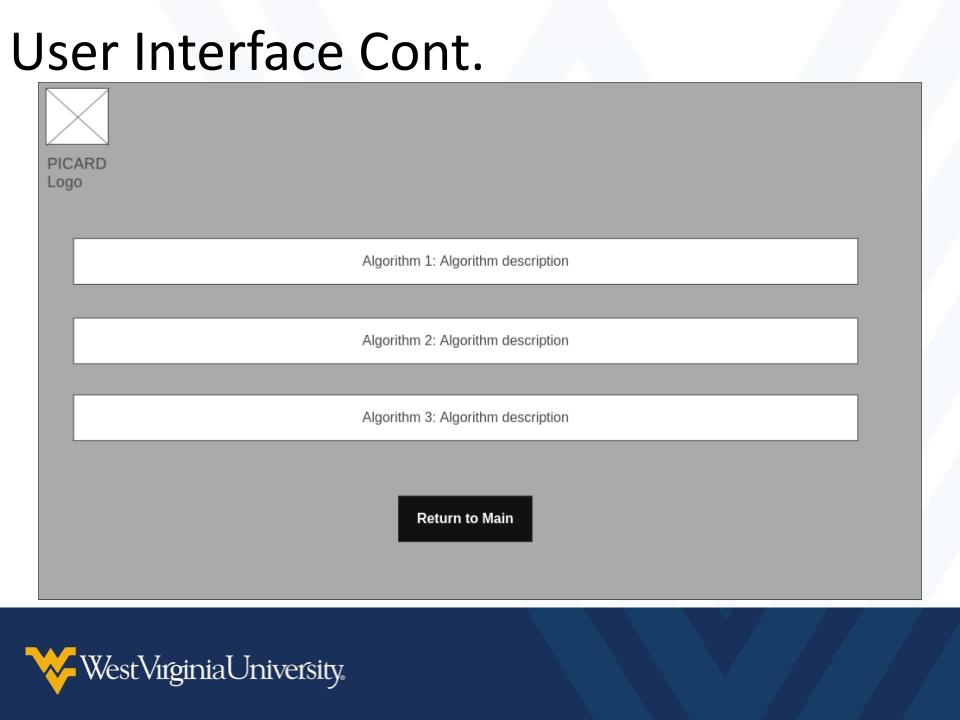
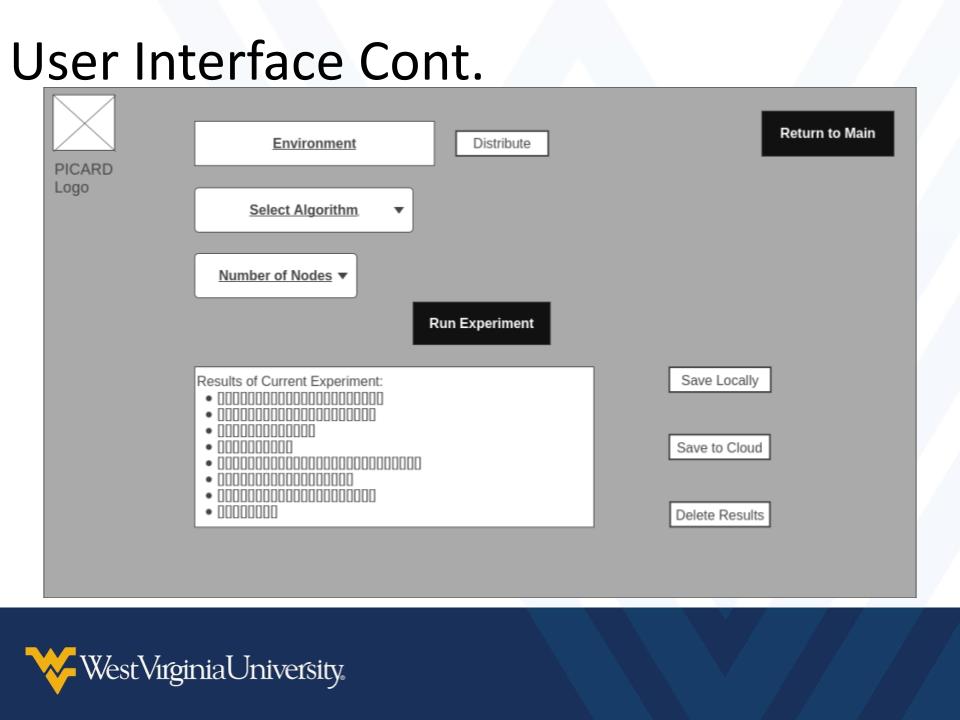
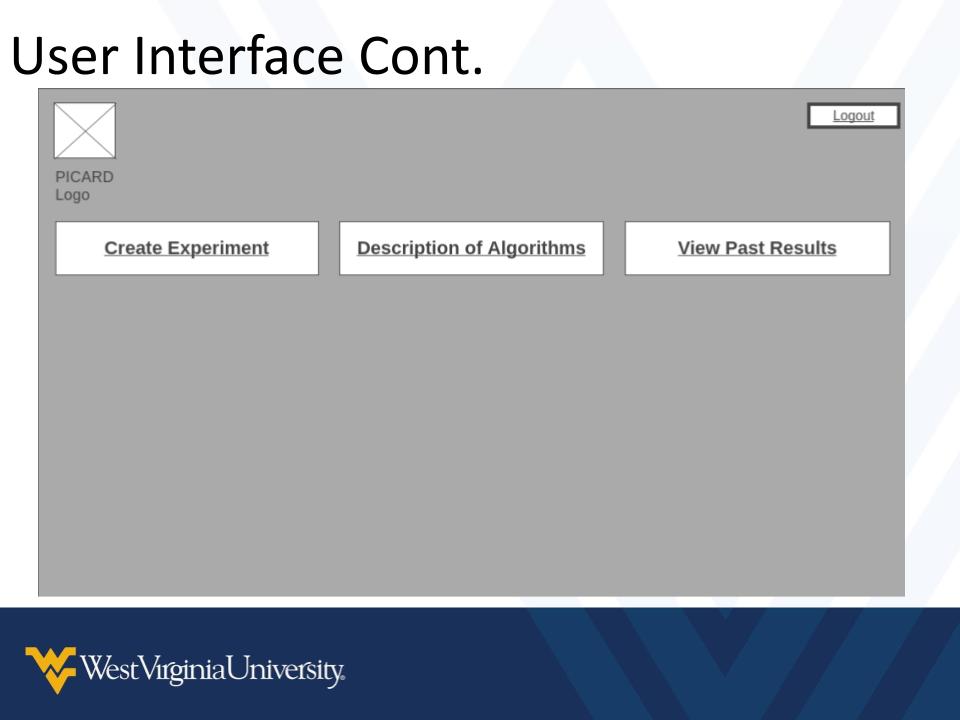
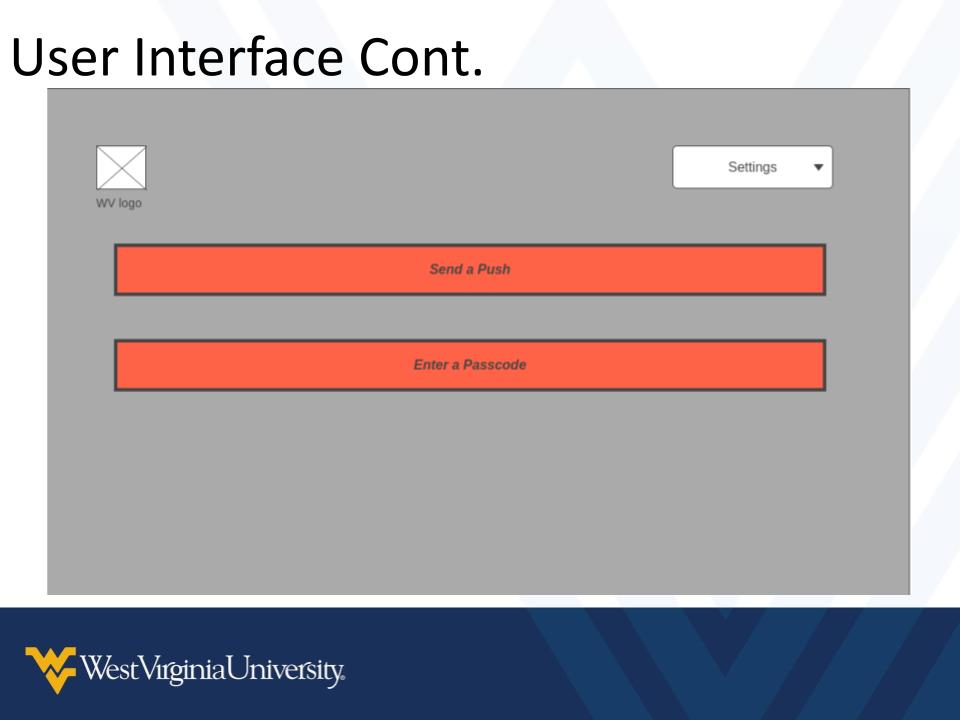
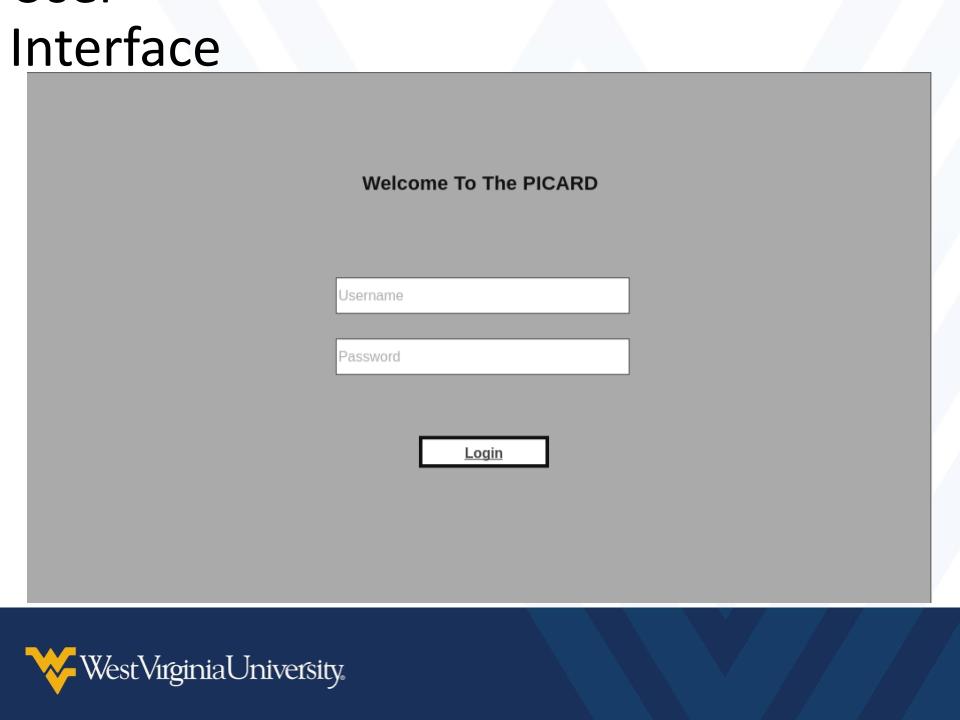
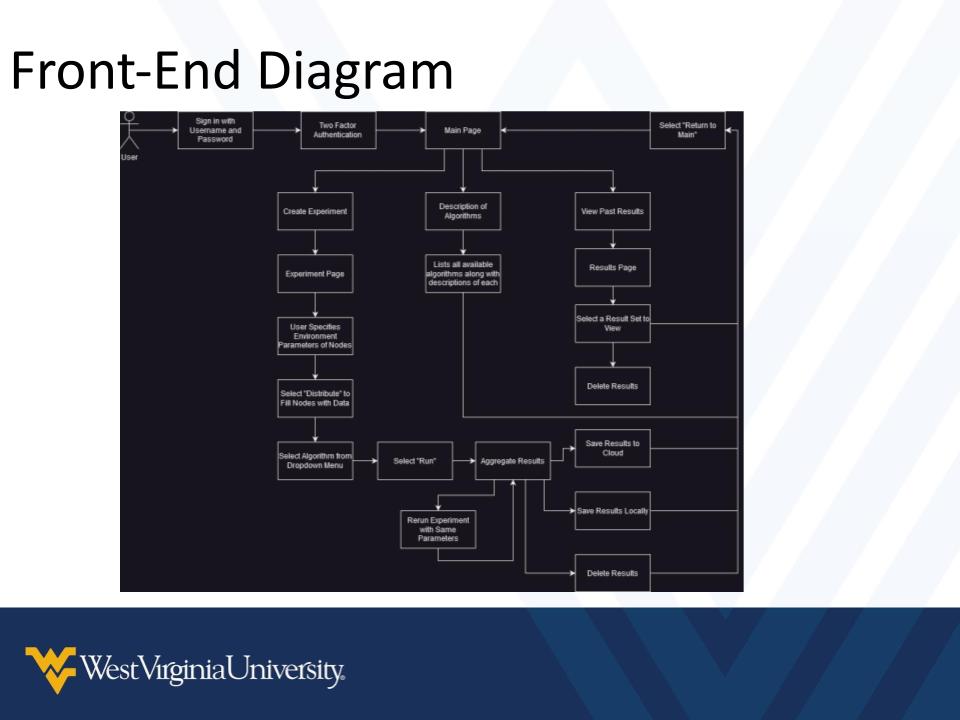
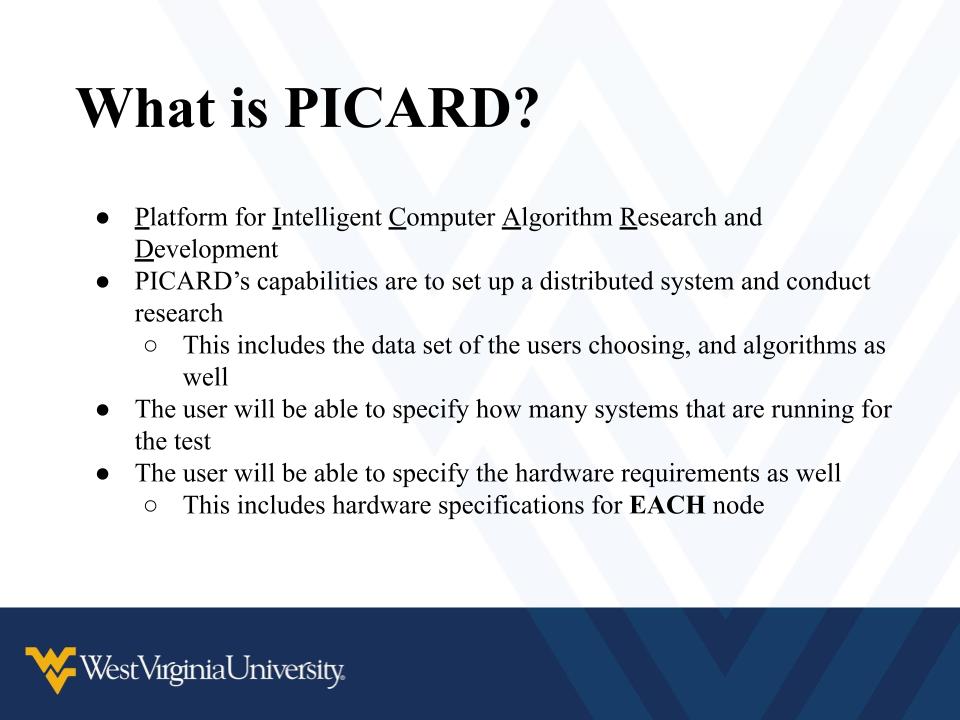
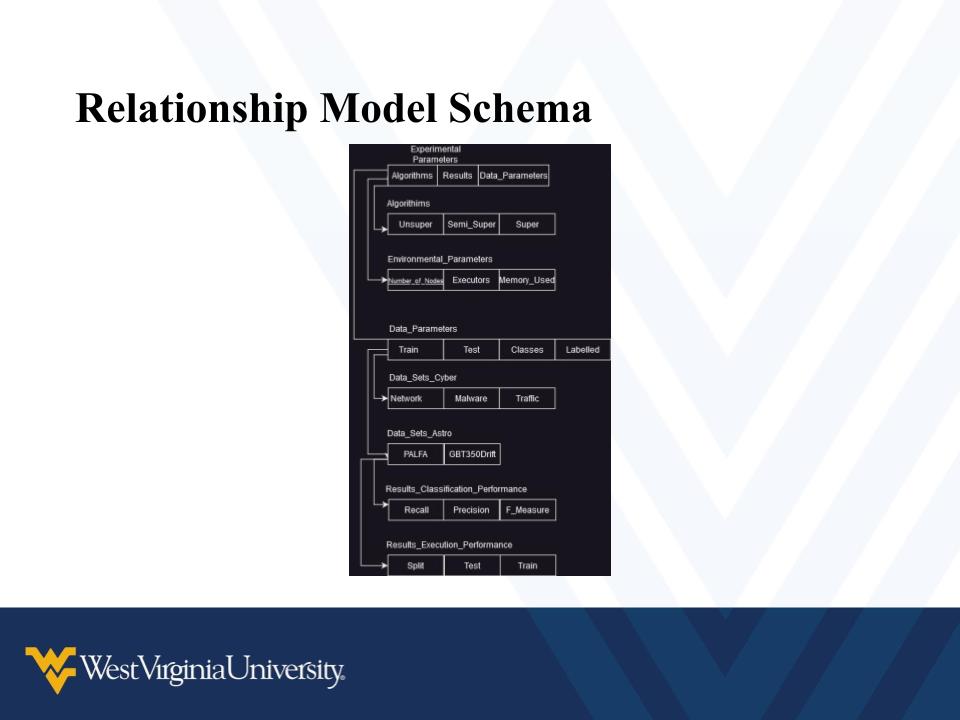
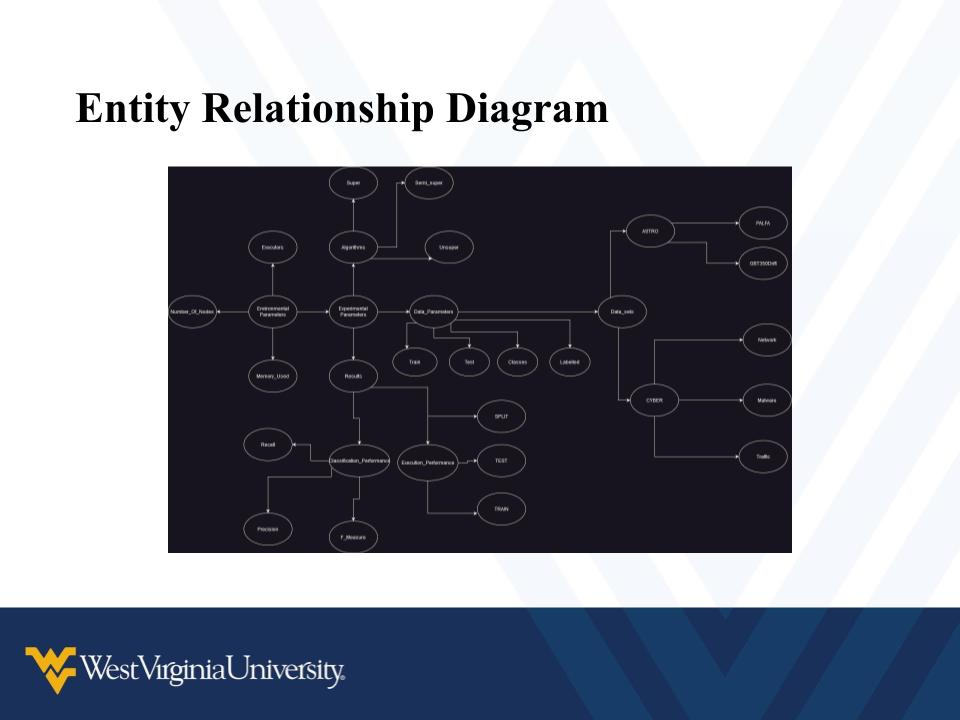
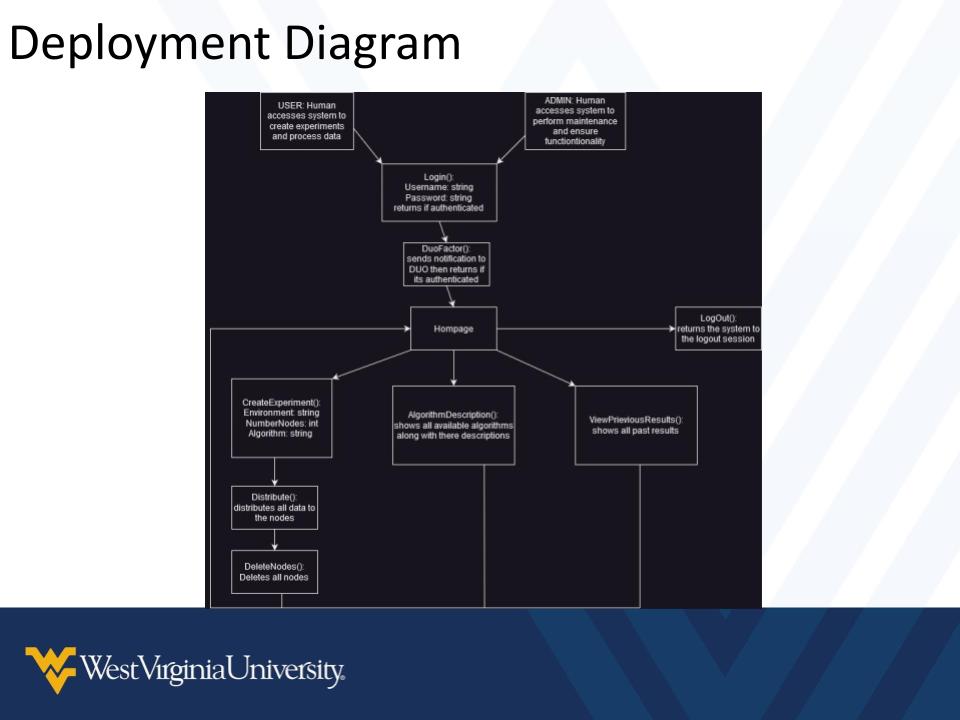
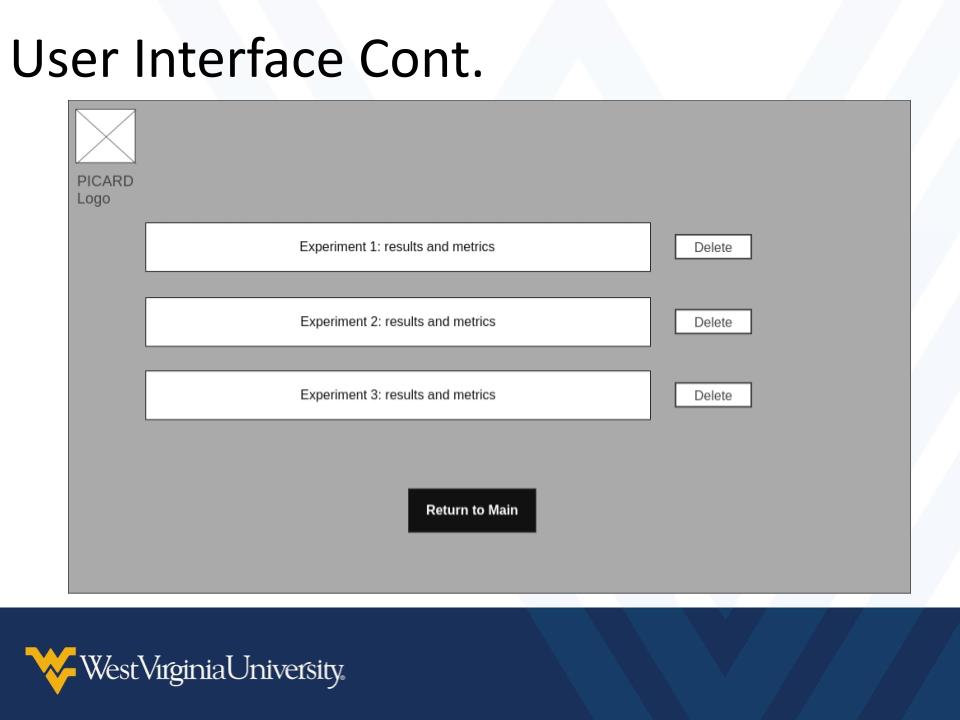
Class: CSEE480 C01

Group Number: 02

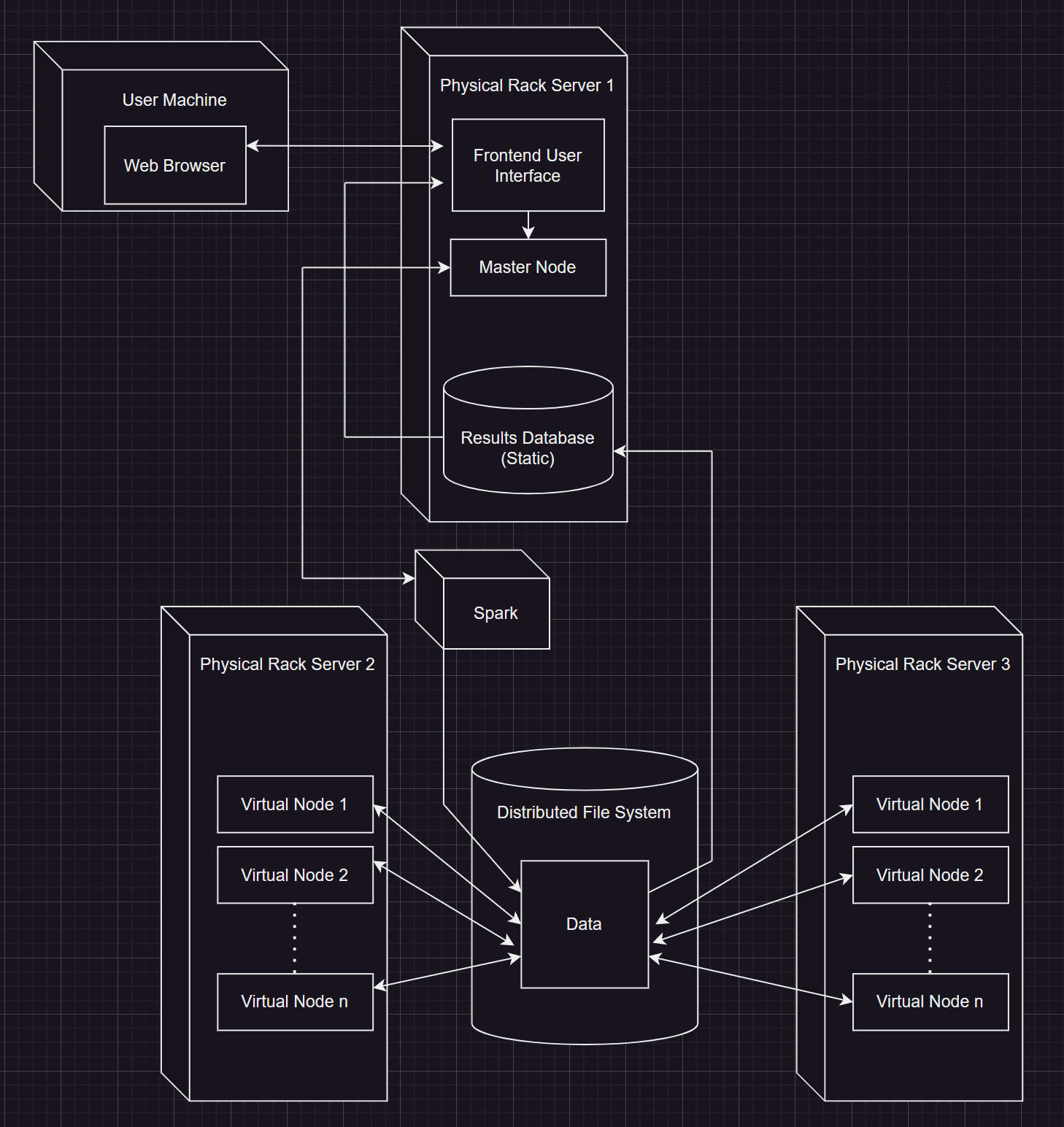
Assignment Name: A2

| Member Name | Meetings attended | Meetings missed | Contribution | Stuff done by member |
| --- | --- | --- | --- | --- |
| Abdalrahman Afifi | 2 | 0 | 100 | ER Diagram, RM Schema, edits |
| Hunter Lavender | 2 | 0 | 100 | ER Diagram, RM Schema |
| Travis Mueller | 2 | 0 | 100 | Front end Diagram |
| Zachary Wildasin | 2 | 0 | 100 | Front end Diagram |

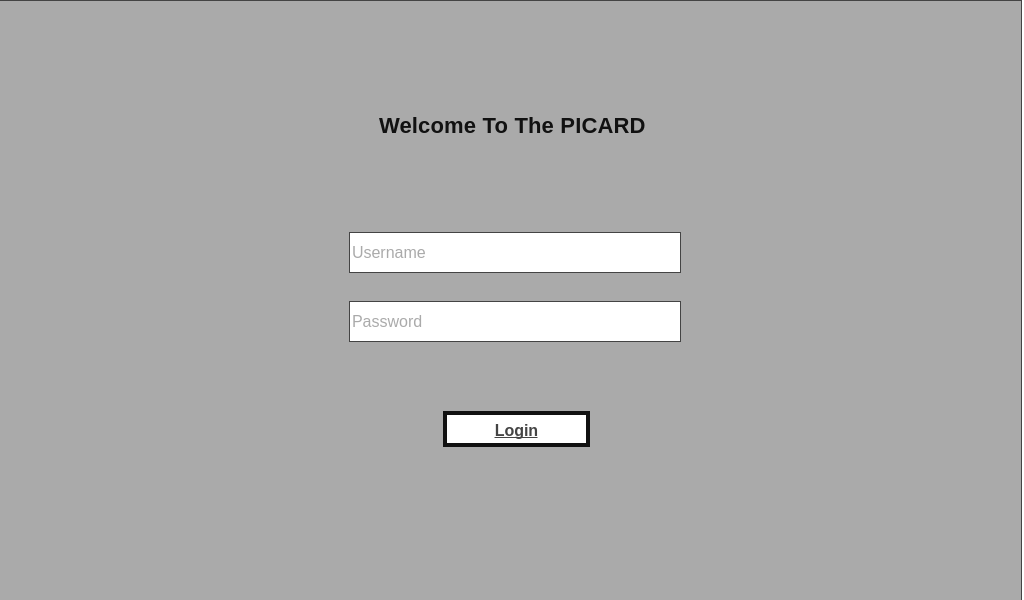
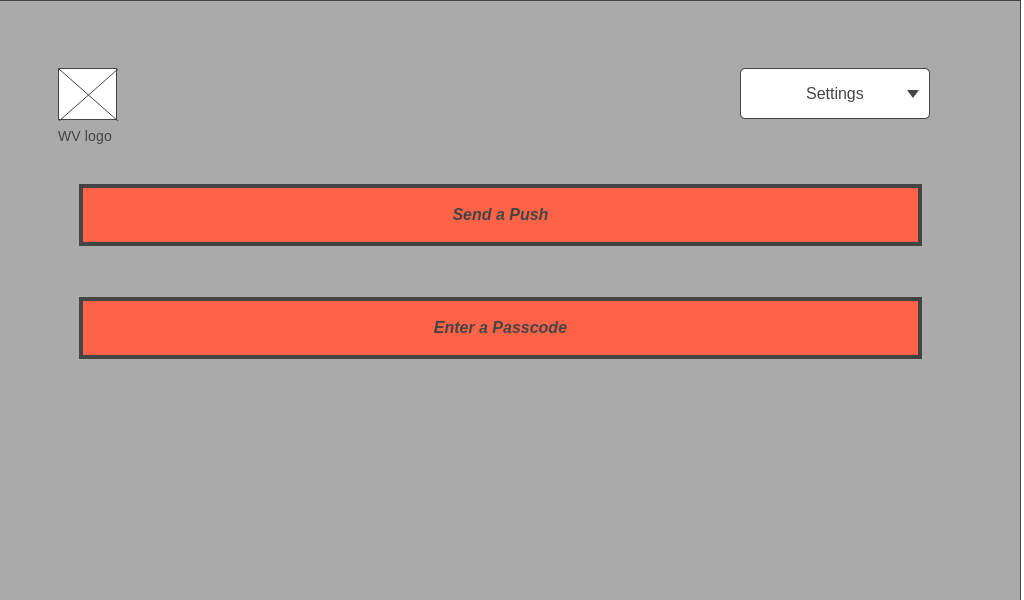


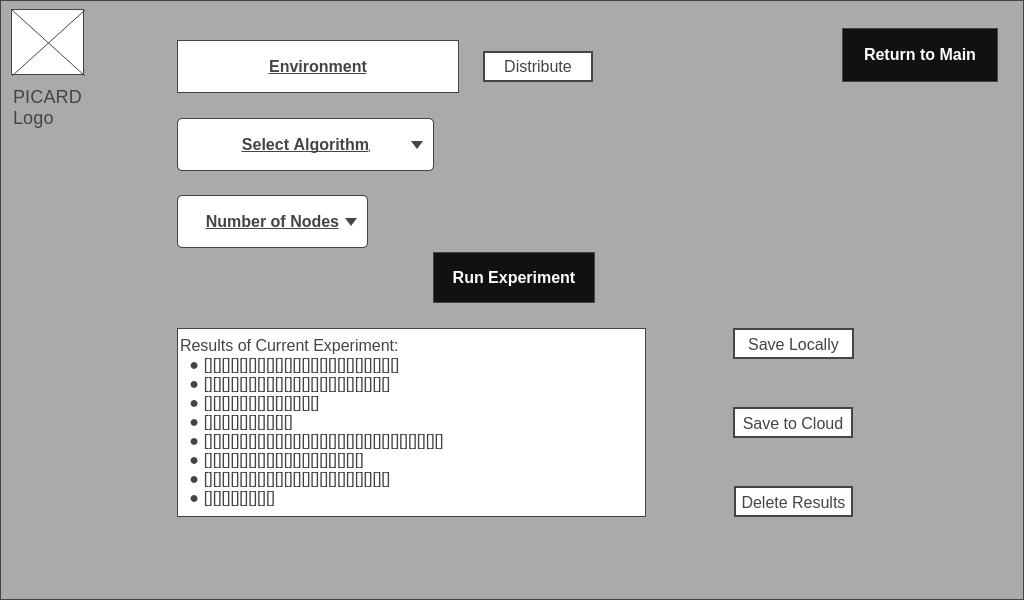
**G**

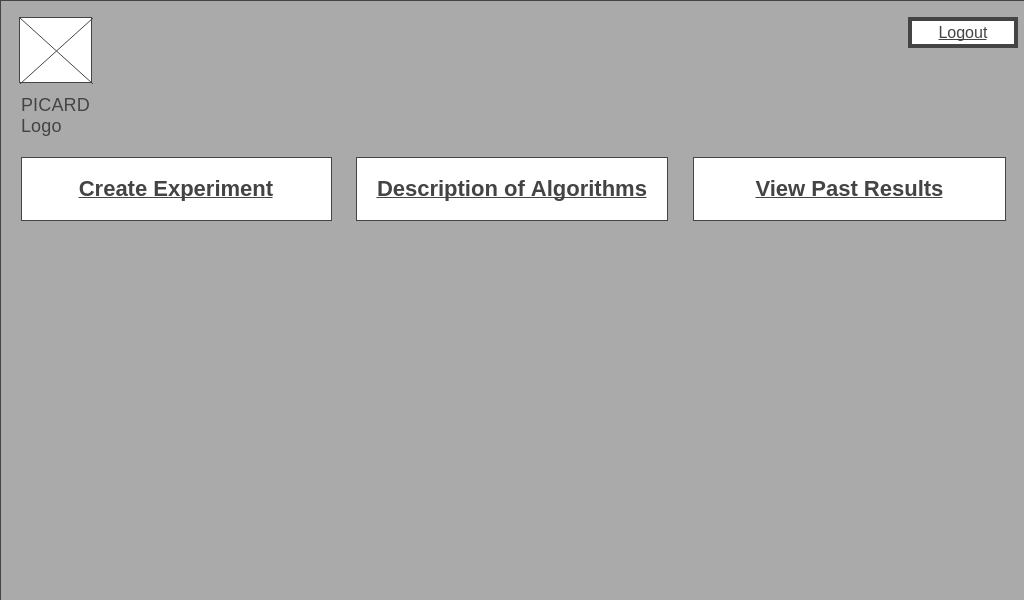
**Deployment Diagram:**



**User Interfaces:**







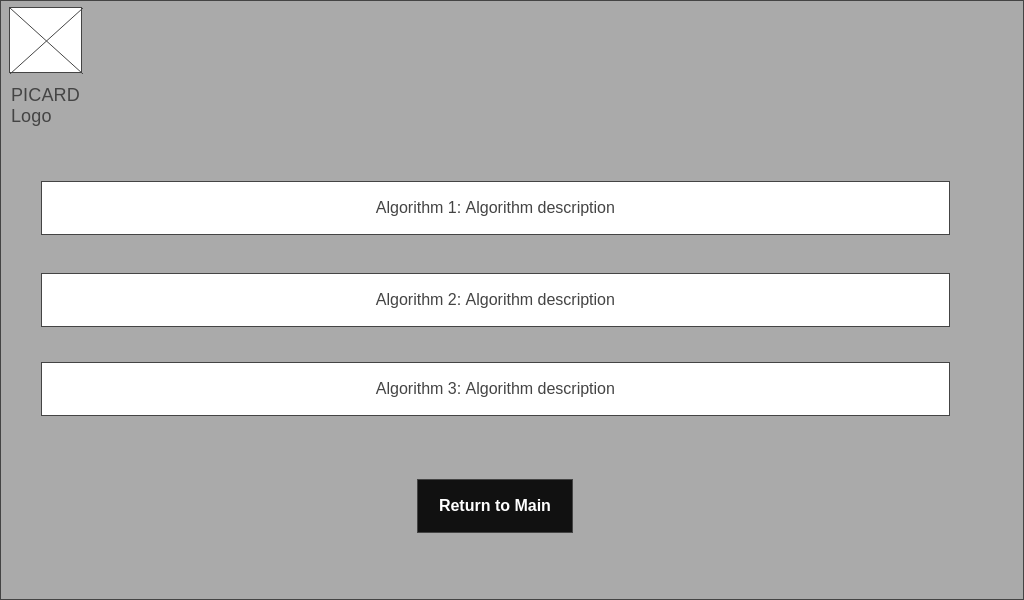
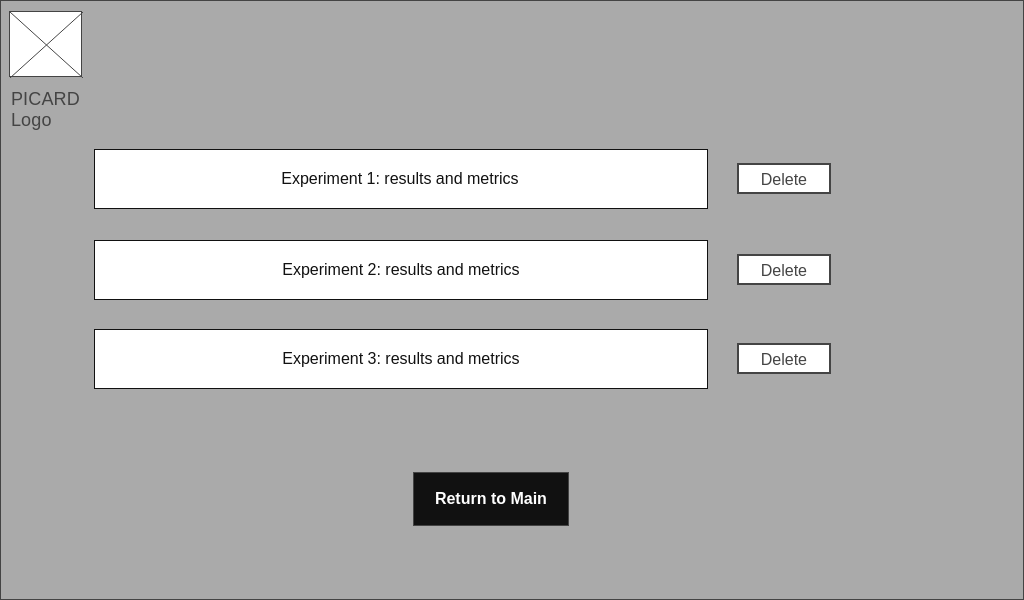


Figure 2:User Interfaces for The PICARD

**Function/Method Definitions and Database Design:**

* Login(string username, string password): boolean
  + Description: Login page for a user to start/resume session
  + Parameters:
    - Username: String value of the user in the system
    - Password: String value of the password in the system
  + Returns: Boolean Value, true on success, false on requirements for user or password being incorrect.
* Upon successful Login:
  + Description: Duo Authentication, send a push/enter a passcode
  + Parameters: Correct User Name & Password
  + Returns: True/False
* Once Logged in: Prompt the user with three options
  + Create Experiment
    - Enter Environment Parameters
      * Select Algorithm: List
        + SupervisedMLRF
        + SelfLearningMLRF
        + CoDRIFT
        + Collective
      * Number of Nodes: Integer Value
    - Run Experiment:
      * Description: Window showing the experiment running
      * Parameters: Must be logged into an account
      * Returns: Integer Value, 0 for no error, 1 for error.
    - Save to Results Database
      * Description: Button to save the data to the Results Database
      * Parameters: Must be logged into an account, Run experiment value. (Since you cannot save an experiment if the test is incomplete.
      * Returns: Boolean, True if downloaded, False if otherwise.
  + Description of Algorithms
    - Description: Prompt to new window showing algorithm’s description, value being a string, Each index in the list should have a description corresponding to list of algorithms
    - Parameters: N/A
    - Returns: String of specified index.
  + View Past Results
    - Description: Prompt to new window showing past results/metrics, values being integers
    - Parameters: Option for each Experiment to delete these results

**MySQL Database**

1. Tables:
   1. Experiment Parameters
      1. Algorithms: String Listing all algorithms
      2. Results: List of Key Value Pairs
      3. Data\_Parameters
   2. Algorithms:
      1. Name: String
      2. Type: String
   3. Enviromental\_Parameters
      1. Number\_of\_Nodes: Integer
      2. UID: Integer
      3. Executors: Integer
      4. Memory\_Used: Integer
   4. Data\_Parameters
      1. PercentTrain: Integer
      2. PercentTest: Integer
      3. NumClasses: Integer
      4. PercentLabeled: Integer
   5. Data\_Sets\_Cyber
      1. Network: Long
      2. Malware: Long
      3. Traffic: Long
   6. Data\_Sets\_Astro
      1. PALFA: Float
      2. GBT350Drift: Float
   7. Results\_Classification\_Perfomance
      1. Recall: Float
      2. Precision: Float
      3. F\_Measure: Float
   8. Results\_Execution\_Performance
      1. SplitTime: Long
      2. TestTime: Long
      3. TrainTime: Long

**Test Plans:**

**Security:**

For the test plans, we’re going to make sure that our website is secured from the most known vulnerabilities such as DDoS attacks, Cross-Site Scripting, SQL Injection attacks, Broken Authentication, Password-based Attacks, man in the middle attacks and buffer overflows.

Furthermore, to be able to do this tests we should either do our own case scenarios by applying it manually and we’ll also use softwares like:

* SonarQube.
* Nessus.
* Semgrep.

**Performance:**

The other type of testing we’ll do is to test the performance of the PICARD which will be done after the interface is done to make sure that everything is working as it’s supposed too such as:

* Each button does what it’s supposed to do and routes to the correct page.
* User friendly.
* The speed of the process is as it should be.
* Assurance of expected outcome with test input for node algorithms.
* Use the dataset that was used by the group before us to rerun the tests again using it.
* Reliability of each solution making sure that the overall up and down time of each node and the whole system.

**Member Contribution Table:**

| Member Name | Meetings attended | Meetings missed | Contribution | Stuff done by member |
| --- | --- | --- | --- | --- |
| Abdalrahman Afifi | 3 | 0 | 100 | User interface, test plans, Final presentation |
| Hunter Lavender | 3 | 0 | 100 | Database design, final presentation |
| Travis Mueller | 3 | 0 | 100 | Deployment Diagram |
| Zachary Wildasin | 3 | 0 | 100 | Deployment Diagram, final presentation, user interface |

**Project Contingency Plan**

Our team is considering three main frameworks for the development of the frontend which are: AngularJS, Angular, and ReactJS. Research has already been done and the pros and cons of each have been established. We plan to further test each framework for performance and usability, after which we will make our final decision.

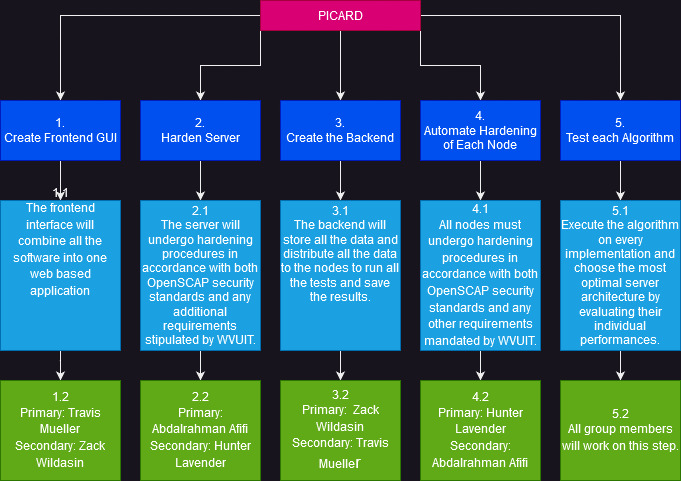
Currently, we prefer the frameworks in this order:

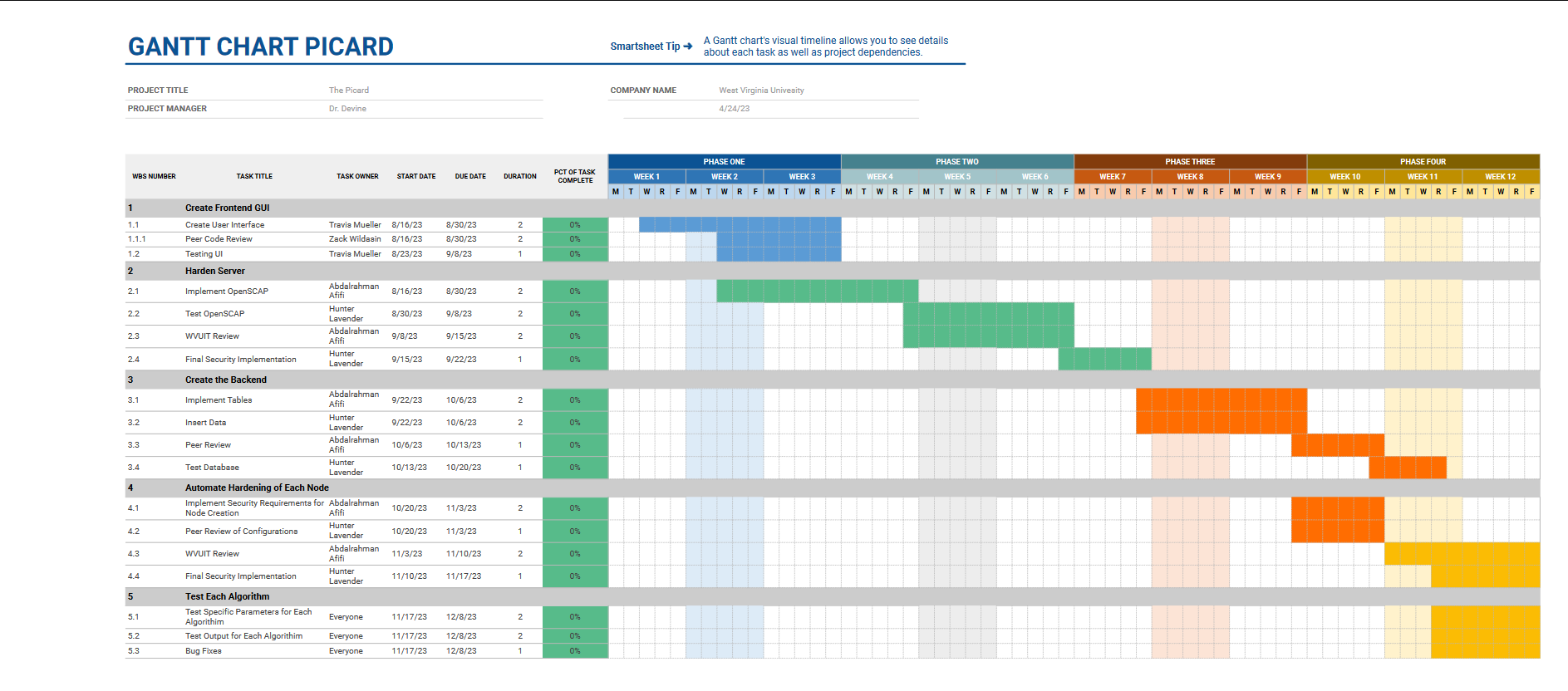
1. AngularJS
2. Angular
3. ReactJS

As for the backend of our application, there are four frameworks that we are considering right now: Django, Laravel, Express, and Flask. Our team has already gathered a list of pros and cons related to each for the scope of our application. Based on these pros and cons, and after the completion of availability, performance, and usability testing, we will decide which of the four frameworks we will be moving forward with.

As of now our preferences, in order, are:

1. Django
2. Flask
3. Express
4. Laravel





| Member Name | Meetings attended | Meetings missed | Contribution | Stuff done by member |
| --- | --- | --- | --- | --- |
| Abdalrahman Afifi | 2 | 0 | 100 | Gantt chart |
| Hunter Lavender | 2 | 0 | 100 | Gantt chart |
| Travis Mueller | 2 | 0 | 100 | Detailed work breakdown |
| Zachary Wildasin | 2 | 0 | 100 | Detailed work breakdown |

**A6 - Summary Report of Group Participation and Procurement List**

**Total Meetings:**

| Member Name | Meetings attended | Meetings missed |
| --- | --- | --- |
| Abdalrahman Afifi | 11 | 0 |
| Hunter Lavender | 11 | 0 |
| Travis Mueller | 11 | 0 |
| Zachary Wildasin | 10 | 1 |

**Procurement list:**

For our project we don’t need to buy anything since the software that we’re using is free and open source.

