

Final Report

Senior Desing II

Project Title	Interactive evaluation of shortest path methods
Client & Advisor	Goce Trajcevski
Team	sddec23-14
Team Members	Alex Blomquist, Samuel Caldwell, Selma Saric, Yadiel Johnson

Table of Contents

1	Original Project Stipulations	2
1.1	Functional Requirements	2
1.2	Non-Functional Requirements	2
1.3	Relevant Standards.....	2
1.4	Engineering Constraints.....	3
1.5	Security Concerns and Countermeasures	3
2	Revised Project Design	5
2.1	Frontend	5
2.2	Backend.....	6
2.2.1	Design Adherence.....	6
2.2.2	Design Changes.....	7
2.3	Implementation Details	9
2.3.1	Frontend.....	9
2.3.2	Backend	10
2.4	Testing Process and Testing Results	12
2.4.1	Frontend.....	12
2.4.2	Backend	12
2.5	Our Work in the Context of Related Products	3
3	Bibliography.....	14

1 Original Project Stipulations

What was the context by which the project changed?

Below is a list of the critical aspects of the project's requirements as discussed with our client during the first semester of Senior Design.

1.1 Functional Requirements

To fulfill the purpose of the project, the product must:

- Enable seamless integration of multiple algorithms and their execution of various datasets.
- To provide correct measurements of the runtime of different algorithms on individual datasets the system must do multiple algorithm runs and use statistical analysis to reduce the possibility of variance given external factors (e.g., context switches, multithreading, and hardware performance).
- Allow users to select datasets and algorithms to test.
- Provide informative visualizations of algorithm outcomes and compare these against each other.
- Generate a report of algorithm efficiency and related data.
- Provide the user with the format to create datasets for executing algorithms, such that the product can use them.

1.2 Non-Functional Requirements

The implementation must:

- Use resources optimally per algorithm run.
- Support parallel, independent workloads.
- Store reports and records of algorithm-dataset runs.

In addition, there are qualitative & aesthetics requirements that should be met.

- Present information neatly for easy understanding.
- Provide detailed information as an option for reports and visualizations.

1.3 Relevant Standards

The following is a listing of the various engineering standards that the team followed and adjusted, with the goal of ensuring that both the design and development process is consistent and high quality. They have been meticulously selected to align the project's development lifecycle to the industry's best practices, and by following their guidelines, the team agrees that they will strengthen both aspects of this proof-of-concept.

- ISO/IEC/IEEE 26514-2021- Systems and Software Engineering Design and Development of Information for Users [1]

- To provide the user with the tools to make the most informed decision when picking an algorithm and educating them on the benefits, standardized documentation is crucial for informing the team about algorithm choices, ensuring consistency in creating and managing documents across the project, benefiting stakeholders involved in designing the software system's various elements. The standard within the framework of our project, assisted us in the design documentation of the graphical user interface, API, and many areas throughout the software system in a manner that provides structure and consistent formatting of information for all project stakeholders.
- ISO/IEC/IEEE 29119-1-2021 - Software and Systems Engineering Software Testing -- Part 1: General Concepts [2]
 - This document provides guidance on creating a comprehensive and consistent approach to software testing. Additionally, it provides insights to reduce risks associated with software failures with the aim of improving the quality of a software product. In the context of our project, this will guide us in designing the respective unit, integration, and acceptance testing scenarios, as well as in defining meaningful regression tests. During the project some of these, such as regression tests became infeasible given the time constraints of the project.
- IEEE/ISO/IEC 42010-2022 - International Standard for Software Systems and Enterprise Architecture Description [3]
 - The document emphasizes focusing on stakeholder perspectives while creating the architecture and provides guidance for documenting the design process. Because the project is aimed at assisting educators and students, we coordinated and met with our advisor frequently during the various stages of the architecture's design.

1.4 Engineering Constraints

- The system must provide all functionality as a full-stack solution.
- The overall implementation should be within a “reasonable” budget (e.g., no more than \$200).

1.5 Security Concerns and Countermeasures

Given our project is a proof of concept, security is not a major concern.

1.6 Our Work in the Context of Related Products

Below is a list of existing products on the web that are similar in nature to our current project. These products all provide a visualization for many common single source shortest path algorithms, as well as letting users implement their own data sets to be tested on, albeit in a limited manner.

- “Pathfinding Visualizer” by Clément Mihailescu
- “Single-Source Shortest Paths” by VisuAlgo

- “Find Shortest Path” by Graph Online

Below is a list addressing the advantages and shortcomings of our project in relation to these previously existing products.

- Advantages
 - Can evaluate more complex data sets with unique properties (i.e., varying edge weights, large data sets)
 - Provides empirical data for the user based on the algorithms’ performances.
 - Allows for the direct comparison of multiple algorithms.
- Shortcomings
 - Less meticulous visualization of algorithms at work.
 - Not designed for users with no experience in single source shortest path algorithms
 - Limited number of unique algorithms to work with

2 Revised Project Design

How did our design evolve throughout development?

Throughout the development of our project this semester, we have made certain changes compared to our original design. Detailed below are the changes we have had to make for the core areas of our project, the frontend and the backend.

2.1 Frontend

The main changes for the design of the frontend involved simple aesthetic changes, the addition of components, and the rearranging of different web pages. The mentioned changes are explained in more detail below:

- **Improved the Overall Color Scheme**
 - The original wireframe we created had a grey, black, and white color scheme for all the website pages. Ultimately, the team decided that this color scheme was a little too simple and that we should add more contrast and color to the web pages. Now the different web pages like the home page, tutorial page, and the results page have a black, white, and green color scheme, with black being the main color and green being an accent color on components like the buttons.
- **Addition of Tutorial Page**
 - Initially, we did not plan on creating a tutorial page for our project as the team felt the website would be simple enough for a user to understand even if they had not used it before. However, we also initially planned for users to be able to enter multiple dataset types of different formats. As we were developing our project this semester, we quickly realized that allowing for multiple dataset formats would make the project far too complex and decided against it. Now, our website only takes datasets of the 9th DIMACS Challenge graph format, so we created a tutorial page to explain to users how their dataset files need to be formatted for the website to work properly. The tutorial page also explains to new users how they can use our website properly.
- **Addition of Source/Destination Point Selection**
 - While planning out our project last semester, we assumed that users would enter the source and destination points they wanted to use within their uploaded dataset file. With this assumption in mind, we did not see a need to allow users to choose source and/or destination points on the website. Once we started development using the DIMACS file format, however, we learned it does not require the user to designate a source and destination point, so we decided to implement dropdowns that allow the user to select these points instead.
- **Addition of Directed/Undirected Checkbox**
 - Like the source/destination point selection situation mentioned above, we also assumed that users would somehow be able to mark whether they wanted to use a directed or undirected graph within their dataset file. The DIMACS file format, however, also does not accept or require this so we decided to implement a checkbox that a user can check if they would like to use a directed graph.

- **Removal of User Accounts/Login & Register Pages**
 - Last semester, we initially planned to implement user accounts so that users could store their datasets and algorithm reports to their account for future use or viewing. Throughout development this semester, we realized that we would not have enough time to implement user accounts and ultimately our main goal was to have successful algorithm execution and comparison. This was a proof-of-concept implementation that ultimately did not need user accounts to fulfill its functional requirements and purpose.
- **Usage of Sigma Instead of Graphology**
 - As discussed in the initial design document, Graphology was chosen as one of the visualization tools for displaying the graph. However, after further research and testing, it was discovered the visualizer required the use of Sigma, as Graphology only provided a data storage system. In turn, Graphology was dropped from the project. The project was redesigned to implement its own graph storage system for the frontend that could be used for Sigma. This decision was made to reduce the overall complexity of the project, as well as allow for the new graph storage to function with the Mapbox visualizer as well.
- **MapBox Visualization, Sigma Visualization, and Algorithm Run Results Now on Separate Pages**
 - In our initial wireframe for our project, we planned to put both our visualizations and the algorithm run results on the same page. During development, the team realized putting all three components together on the same page would be difficult to implement and would be too much on one page for the user to process and could potentially confuse them. Ultimately, we decided to put all three components on three separate pages, so the user could view them individually and understand the purpose of the three different components much easier.

2.2 Backend

The scope of the design was primarily guided by the robust documentation the team had written in the first semester of Senior Design. With that as a starting point, there were some notable changes that were done to improve the project's outcome. The design aspects that were adhered to, and those that were deviated from, can be found here.

2.2.1 Design Adherence

All major decisions from the original design document have been followed to the extent possible. This includes:

- A RESTful web application with an associated API path.
- A small yet representative set of *Single-Source Shortest Path* and *All-Pairs Shortest Path* algorithms.
- A modular approach that allows separating the web server from the algorithm execution logic.

Regarding the implementation of the design.

- “The system should support multiple algorithms and datasets. This necessitates a scalable format for adding algorithms and datasets, as well as requiring data persistence.”
 - The implementation of the server-side code features a design for the Algorithm Execution Driver (AED) module that allows for extensibility. The way this is done is by using polymorphism and abstractions to reduce the immediate dependencies some aspects of the code have on others.
 - For example, the AED only concerns itself with the existence of things that extend the abstract classes for *Single-Source Shortest Path* and *All-Pairs Shortest Path* algorithms. This allows for custom implementations of these classes that could, in practice, be extended with custom algorithms.
 - Datasets are split into two possible representations: a *Graph* object, and its serialized content stored inside the persistence-friendly *Dataset* object. The former is a fully featured standardized graph format that can be transformed into other, algorithm-specific graphs, and the latter serves as its wrapper for persistence operations handled by the Spring Framework.
- “Algorithm execution must be measured, and metrics must be gathered regarding their runtime and space complexity for the purposes of comparison. The system must therefore use statistical analysis to gather and present this information to users in a usable manner.”
 - Metrics gathered reflect what can be extracted from an algorithm’s execution. This includes statistical analysis based on their runtimes across a certain number of runs, but not memory usage (see Design Changes below).
- “Parallel execution of algorithms to support multiple user workloads.”
 - Utilizing Spring Web and its underlying Tomcat server, it is possible to create a servlet application that utilizes a preset thread pool to handle multiple requests concurrently.

2.2.2 Design Changes

The original design for this project expected a level of separation between the Algorithm Execution Driver (AED) and the algorithms it would be executing. The idea being that it would be a multi-language tool, capable of executing shortest path algorithm code and interpreting the results. However, it was found that this manner of implementing it would bring in additional complexities that encumbered the rest of the design. Here are some of the associated issues:

1. Language compatibility and overhead
 - a. Supporting a theoretically limitless number of languages, data I/O formats, runtime environments, libraries, and their associated nuances would extend change the scope of the project to be primarily about interoperability rather than algorithm execution and comparison. Even reducing it to one other language, such as C, would require developing ways of achieving concurrency-enabled inter-process communication, data structure conversions, operating system resource management, cross-language error handling, and more.
 - b. There are exceptions that would simplify this process: the Java Native Access Library (JNA) and Java Native Interface (JNI) would allow for execution of native C code within the Java Virtual Machine (JVM). This approach introduces some

problems: JNA has overhead that is not directly related to the algorithm implementation itself, and they violate the current security model as seen below.

2. Security model and stability requirements
 - a. The above complications would result in a reevaluation of the security aspects of this design. The execution of arbitrary code in other languages that would be enabled by such a design goes against the relatively minor concerns associated with security that one would associate with a utility like the one proposed.
 - b. Additionally, if instead the design incorporated JNA or JNI, there is a concern that at a certain scale, the JVM would be burdened with non-collectable unmanaged memory that would cause it to inadvertently crash. Because Spring Web utilizes multiple threads, it is possible to have multiple requests on large graphs with high storage requirements resulting in an application crash that is significantly harder to debug and correct.

Because of this, the decision was made to restrict the program to Java, focusing on optimizing and refining the execution of algorithms within the Java ecosystem rather than incorporating other languages. This has other implications that will be discussed below.

Regarding space complexity and memory utilization metrics

A significant change was made to the original design regarding how the AED would gather information about the resources allocated to each algorithm.

From the initial Design Document:

“Algorithm execution must be measured, and metrics must be gathered regarding their runtime *and space complexity* for the purposes of comparison. The system must therefore use statistical analysis to gather and present this information to users in a usable manner.”

The JVM does not support either manual memory allocation nor per-method memory analysis; beyond using tools such as a profiler that runs parallel to the JVM, there is no way to accurately gauge the memory usage of a singular method at runtime and also have it fed to the same application. This is entirely a language limitation which could be circumvented by using another one, however it would require using libraries other than Spring Framework and its dependencies which the team was already well acquainted with. As a result, live memory usage metrics, including gathering and comparison, in practical measurements such as bytes, has been omitted.

Regarding persistence and structure

Details on how persistence and modularization deviate from the initial design can be found in the Implementation Details section below.

Regarding the pipeline and Docker integration

To speed up continuous integration/deployment as well as running the server the team decided to utilize Docker. This would interface with the Gitlab-runner and containerize the web application from the GitLab Repository and host it on the VM as a Docker Image accessible to anyone on the ISU network.

Regarding the MySQL & data storage

To store the runtime logs and datasets on the backend, we used MySQL to store them on it, primarily for its compatibility with Docker and Spring Boot. This was primarily done to provide data

persistence regarding datasets that may be especially large and contain many nodes with respective edge weights.

2.3 Implementation Details

2.3.1 Frontend

Sigma Implementation

- When a user uploads a graph from the homepage, the details of the graph are saved to the local storage. Once the user is directed to the Sigma visualization page, the graph is built from the local storage, which is then fed into the Sigma visualizer. The visualizer is then rendered onto an HTML canvas, which will show the graph as a series of colored nodes and edges.

MapBox Implementation

- Since Mapbox requires a GeoJSON format, the preconfigured graphs used for the Mapbox implementation are first translated into a GeoJSON object, which is then fed into Mapbox. Mapbox will then render the converted graph as a map, with the edges being represented as roads, and the nodes being represented as the road intersections.

Goal of Each Website Page

- Home Page
 - The main goal of the home page is to allow the user to select which algorithm(s) they would like to run, and upload a new dataset or choose which preconfigured dataset they want to run said algorithm(s) on. The user has the option of choosing one or two single source shortest path or all pair shortest path algorithms. Once they upload or choose a preconfigured dataset, the user can also choose which source and destination points they would like to use. Lastly, there is a checkbox that a user can check if they would like to have a directed graph, and can leave unchecked if they want it to be undirected.
- Tutorial Page
 - The main goal of the tutorial page is to explain to users how to use the website and to explain the file format that they need to use for the dataset they want to upload. The tutorial details the specific steps the user has to take to get the website to function properly, and has a detailed explanation on how their dataset file needs to be formatted in order for our backend to be able to process it correctly.
- Sigma Visualization Page
 - The main purpose of this page is to display the Sigma visualization that is created from the dataset the user uploaded. The visualization includes the nodes and edges/edge weights that were written in the user uploaded dataset file.
- MapBox Visualization Page
 - The purpose of this page is to display the Mapbox visualization that is created from one of the preconfigured datasets saved in the project. The edges and nodes are visualized here as roads and intersections, respectively. This visualization will provide users with a different way to view a graph dataset in a way that is more familiar to them.

- Results Page
 - The main purpose of the results page is to display metrics about a user’s algorithm run, such as runtime.

Frontend to Backend Communication

- Algorithm Dropdown Population
 - To populate the single source shortest path and all pair shortest path algorithm dropdowns, a GET request is made to the backend for the list of algorithms and it dynamically populates the dropdown using this list.
- Algorithm Execution and Obtaining Run Results
 - Once the algorithm(s), dataset, source and destination points, and whether the graph is directed or not has been selected on the fields within the home page, the user will click the “Start Now” button. This button sends two POST requests to the backend.
 - The request body of the first POST request contains the dataset file in a string format that the AED in the backend will read for the algorithm execution. There is also a Boolean value associated with the directed/undirected checkbox, that is set to true if the user selected the checkbox and wants a directed graph. The third value in the request body is the name for the dataset, that gets a random unique identifier assigned to it. Lastly, an id value of ‘0’ is sent, which will be overridden with a different id number by the backend code in the response. The response sent in sends this new ID number, which will then be used in the next POST request.
 - In the payload of the second POST request (which happens immediately after the first one), the ID from the response of the last POST request is sent as the ID for the dataset. The source/starting point is also sent in the payload of this request. The response that is received from this POST request contains the different algorithm results/metrics from the algorithm run, including metrics like runtime, and is displayed on the algorithm results page.

2.3.2 Backend

Because the AED is assumed to only operate under a singular language, it is possible to refine the design such that it capitalizes on that language’s features. The initial design featured a block diagram, seen below, wherein the *Backend* would access different storage mediums for logs, datasets, and algorithms. This followed the assumption that the AED would have to execute commands to run certain algorithms and would therefore have to access them on some operating system directory. Additionally, it assumed that datasets would be entirely transposed by the AED into specific formats that would be used for both algorithms and web server operations and rendering.

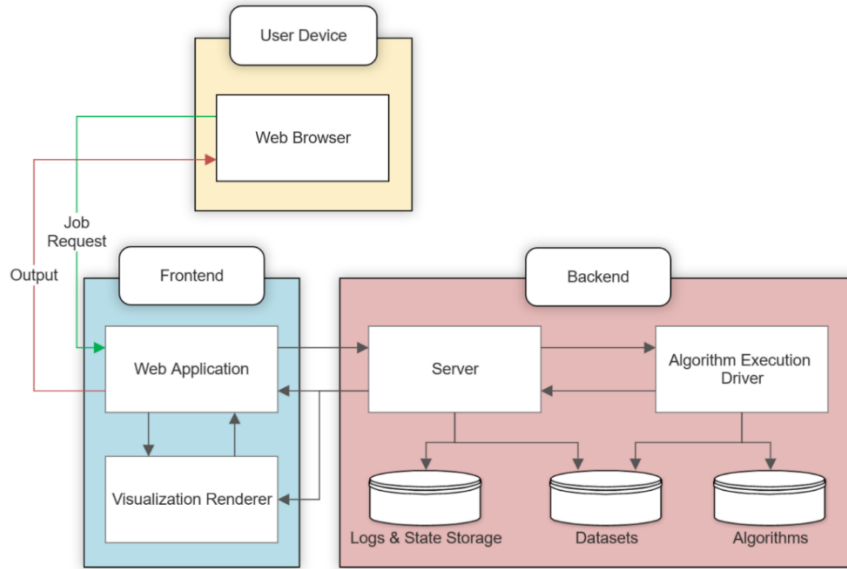


Figure 1 - Original Block Diagram

In the updated design for the backend, the structure changes significantly. Algorithms are provided as pure Java solutions which may have their own graph representations, execution format, etc. The algorithms must therefore be wrapped to some standard format that would allow for them to be programmatically executed, alongside providing ways to create individual, specific graphs from a standard graph format. Additionally, an *Algorithm* Executor would take the necessary operations and ensure that they are performed as per its specification, taking into account factors such as minimum repetitions. Note that the AED need not receive the algorithm definition from within itself; it is possible to create a custom implementation that gets provided to the AED upon instantiation.

Below is a diagram that attempts to illustrate the updated backend design, with a focus on the AED. Note that names, colors and item shapes do not follow the same pattern as the previous block diagram.

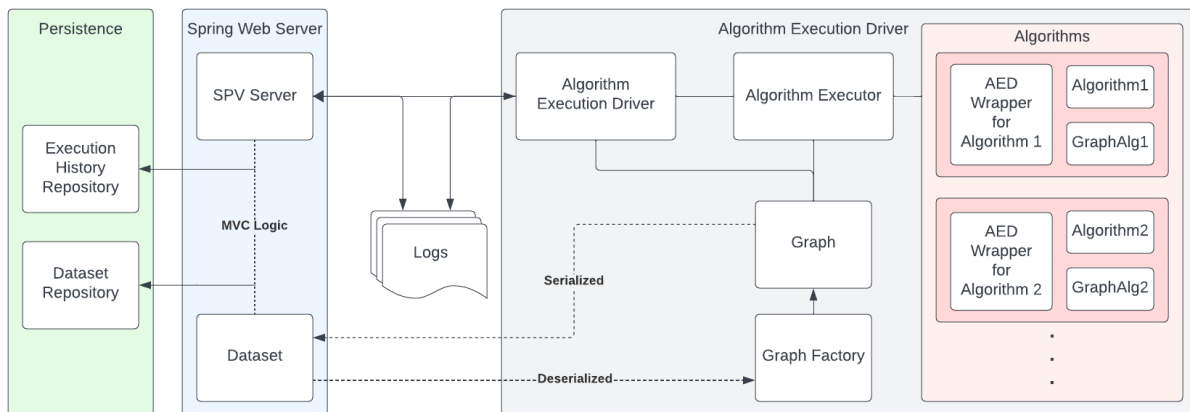


Figure 2 - Revised Backend Block Diagram

2.4 Testing Process and Testing Results

2.4.1 Frontend

Algorithm Selection & Dataset Upload Tests

The home page of the website includes a dropdown menu that allows the user to select algorithm(s) to execute. There is also a button that will open the user's file explorer so they can select a dataset file and have it uploaded to the website. Both of these input selections were tested using Jest to ensure that everything was functioning properly so that inputs could eventually be passed to the backend.

Visualization Tests

Visualization tests were performed through manual test cases and scenarios. These included testing the visualizer navigation functionality, ensuring the user's uploaded datasets were being represented correctly, and testing the overall appearance of the datasets to ensure the visualizers could present the datasets in an easy-to-read fashion.

Testing Tools

- Jest

2.4.2 Backend

The design document outlines the general goals for testing sections of the project. In the backend, these tests covered the following topics:

- “The accuracy of the algorithm executions must be prioritized in testing; incorrect or misrepresented results undermine the main utility of this project.”
 - The test suite validates the minimal interference caused by the introduction of wrappers and auxiliary data structures by providing a test that runs the original code against the wrapped alternative and compares their runtimes. This value should be +/- 5% of the original code.
- “The reliability of the system under abnormal situations must also be tested. The processing of various algorithms and datasets, some of which are user submittable, requires that a malicious input does not render the system unstable.”
 - Extensive testing over validation of datasets provided by users, including datasets that have a larger than reasonable node or edge count, and over a large number of requests, are part of the testing suite.
- “Testing the validity of the datasets programmatically against their specified format using static analysis tools is required.”
 - This was achieved by using a custom version of the 9th DIMACS Challenge graph format, which can be processed and converted to a standard AED-compatible graph. This process has several tests associated with it, including when converting to algorithm-specific graph formats.

Integration test automation for API endpoints took precedence over the other tests, as they provided the most important insight when working through the entire application. Integration tests are bundled together to create the end-to-end test suite. From there, tests for the AED were designed

to cover two important areas: ensure expected functionality from certain subclasses (namely algorithms) and achieve a reasonably high code coverage percentage. For the web server, the tests focused on the expected HTTP response codes for each endpoint given a specific scenario.

Testing tools

The testing suite that was proposed originally has been updated. Now, a combination of the following noteworthy tools and libraries are actively used in the backend tests:

- Spring Boot Test
 - Mockito
 - Jackson
 - JUnit
 - MockMVC
- IntelliJ IDEA
 - Profiler
 - Code Coverage Analysis
 - HTTP Client

The toolset provided by IntelliJ has streamlined development and debugging for the backend. The others, provided under the Spring Boot Test dependency, allow for very particular tests over certain components (the controllers and endpoints in particular) that would otherwise necessitate different tools.

3 Bibliography

- IEEE. (2022). IEEE/ISO/IEC International Standard for Software, systems and enterprise--Architecture description. *ISO/IEC/IEEE 42010:2022(E)*. doi:10.1109/IEEESTD.2022.9938446
- IEEE. (2022). ISO/IEC/IEEE International Standard - Software and systems engineering --Software testing --Part 1:General concepts. *ISO/IEC/IEEE 29119-1:2022(E)*. doi:10.1109/IEEESTD.2022.9698145
- IEEE. (2022). ISO/IEC/IEEE International Standard - Systems and software engineering -- Design and development of information for users. *ISO/IEC/IEEE 26514:2022(E)*, 1-76. doi:10.1109/IEEESTD.2022.9690115
- La Sapienza. (2006). *9th DIMACS Implementation Challenge - Shortest Paths*. Retrieved from Sapienza, Universita Di Roma: <http://www.diag.uniroma1.it/~challenge9/format.shtml#graph>
- Pruehs, N. (2009). *Implementation of Thorup's Linear Time Algorithm for Undirected Single-Source Shortest Paths with Positive Integer Weights*.
- Tharmarajasingam, T. (2023). *Algorithms*. Retrieved from GitHub: <https://github.com/thuva4/Algorithms>
- VMware, Inc. (n.d.). *Spring Framework Documentation, 6.1.1*. Retrieved 2023, from Spring: <https://docs.spring.io/spring-framework/reference/index.html>
- Wetherell, J. (2020). *Java : Algorithms and Data Structure*. Retrieved from GitHub: <https://github.com/phishman3579/java-algorithms-implementation>

Appendix I

Setup:

- To setup the project, first start by downloading the source code from Gitlab
- If using an IDE:
 - Open the project in your preferred IDE
 - Run the Application.java file in the IDE, located under src/main/java/dev/yjohnson
 - Once the server application has started running, you may begin using the application by opening the Homepage.html file in a browser, which is located under src/main/resource/public
- If Using the CLI:
 - Navigate to the project directory
 - Run the following commands:
 - ./mvnw.cmd compile
 - ./mvnw.cmd package
 - ./mvnw.cmd install
 - java -jar -Dspring.profiles.active=test target/spv-server-0.0.2-SNAPSHOT.jar
 - Once the server has started, you may now open the Homepage.html file in your browser, which is located under src/main/resource/public

Demo:

- Starting from the homepage, you must select which type of algorithms you wish to use, this can be found on the left side of the screen. Simply press either the APSP button for all pairs shortest paths or the SSSP button for single source shortest path.
- Once one of the two buttons is pressed, a drop-down menu will appear, allowing you to select which specific algorithms you would like to use depending on which category you selected. Select one or two algorithms from the list, which should appear below the drop-down menu. An example of this is shown in Figure A.1.1.1

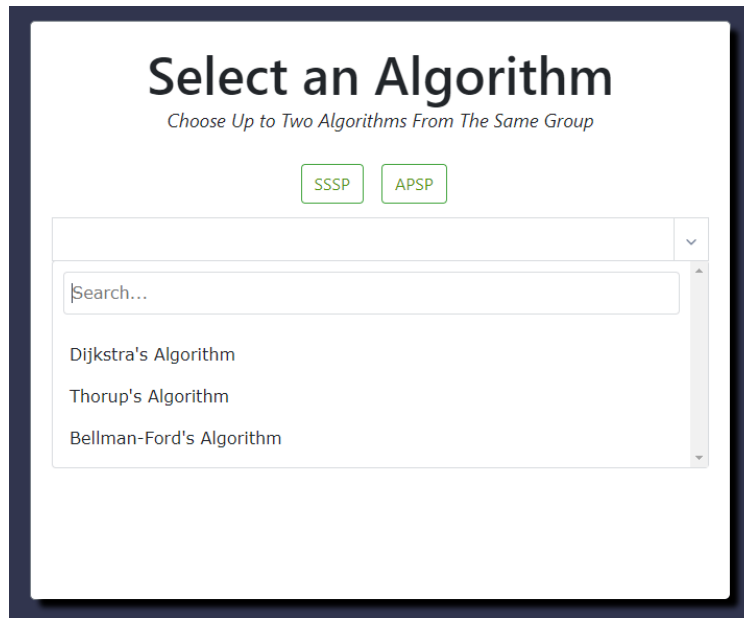


Figure A.1.1.1 – Algorithm Selection

- Once one or two algorithms have been selected, you may choose which dataset you would like to use. You may either select one of the preconfigured graphs by clicking on one of the buttons that says, “Long Beach” or “Escalon”.
- If you wish to upload your own dataset instead, press the choose file button, which will allow you to upload your own text file describing the dataset. Once you have selected your dataset, press the upload button to confirm.
 - Note: The text file describing the graph must follow a certain format. See Figure A.1.1.2 to see an example of the file format. the tutorial page will also show information on how to format the text file.

```

c 9th DIMACS Implementation Challenge: Shortest Paths
c http://www.dis.uniroma1.it/~challenge9
c Sample graph file
c
p sp 6 8
c graph contains 6 nodes and 8 arcs
c node ids are numbers in 0..5
c
a 0 1 17
c arc from node 0 to node 1 of weight 17
c
a 0 2 10
a 1 3 2
a 2 4 1
a 3 2 1
a 3 5 3
a 4 1 1
a 4 5 20

```

Figure A.1.1.2 - File Format

- After selecting a dataset or uploading your own, you may choose to preview it in a new tab by clicking the preview button
 - Note: If you choose to use one of the preconfigured datasets, the visualizer will be set to run as a MapBox visual, showing either a map of Long Beach or Escalon, depending on which dataset you chose. If you choose to upload your own dataset, the visualizer will run as a sigma visual, showing the dataset as a series of colored nodes and edges. An example of both visualizers can be seen in Figure A.1.1.3

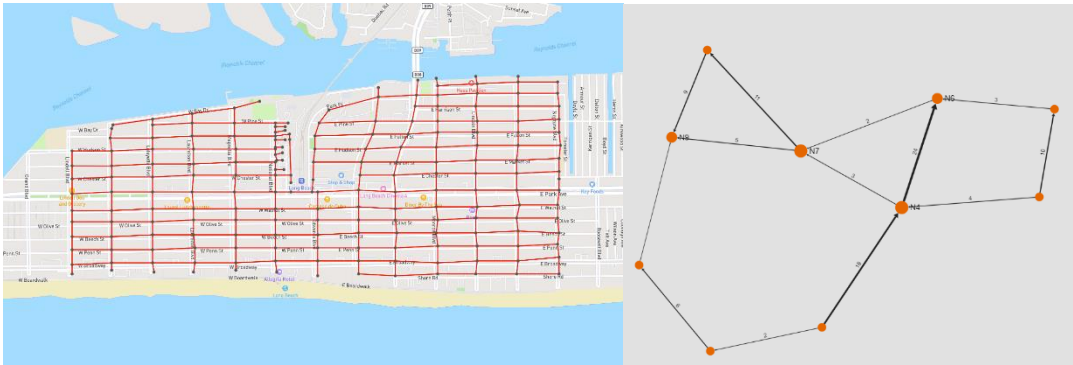


Figure A.1.1.3 - Mapbox Visualizer (Left) and Sigma Visualizer (Right)

- You will now be able to enter a starting and ending point for the dataset within the range of 0 to the number of nodes in the dataset. Enter the number for the starting point in the textbox labeled “source point” to set that node as the source point. Do the same thing again for the ending point to set that node as the destination point.
- Finally, after the algorithms have been selected, the dataset has been chosen, and the starting and ending points have been specified, you will now be able to press the “Start Now” button to begin the algorithms execution on the datasets. This will take you to the results page to display the results of the algorithm executions.
- In the results page, you will be able to view the output of the algorithms chosen on the home page. The results will display information regarding the size of the graph they executed on, the average time it took per execution, the total time for all the executions, and the standard deviation of all the executions.
- You may also press the “View Path” button to be taken back to the visualizer page. This page will show you the shortest path in the dataset, from the starting node to the ending node that you selected on the homepage.
- After viewing the data from the results page, you may press the “Restart” button to be redirected back to the homepage, where the process will begin again.

Appendix II

The below Figures A.2.1.1-3 detail removed features from our project design such as Figure A.1 showcasing our original User login, info, and registration pages with the reasoning for its removal found in section 1.1.

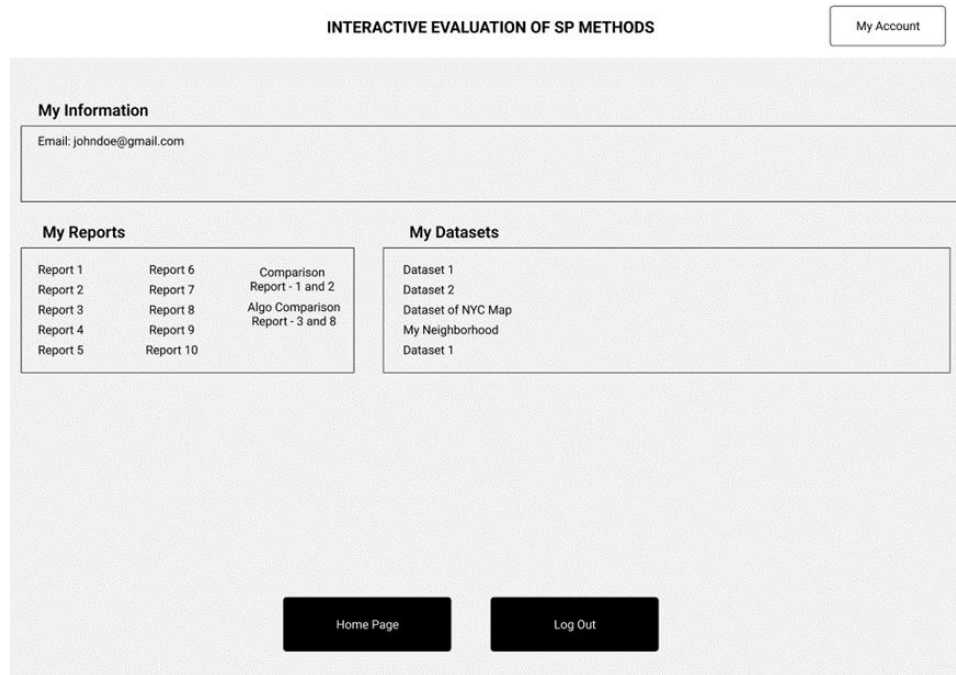
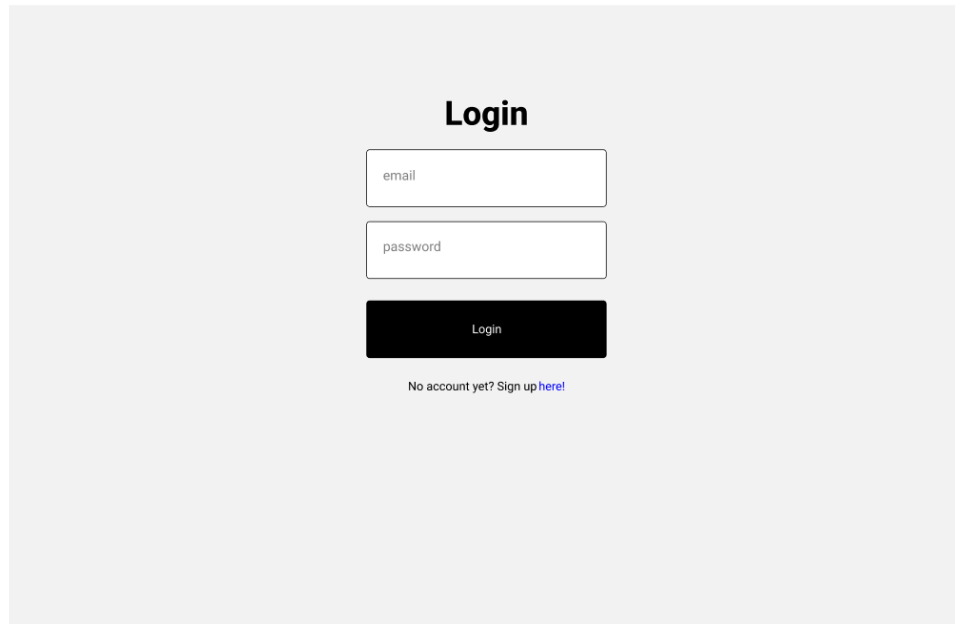
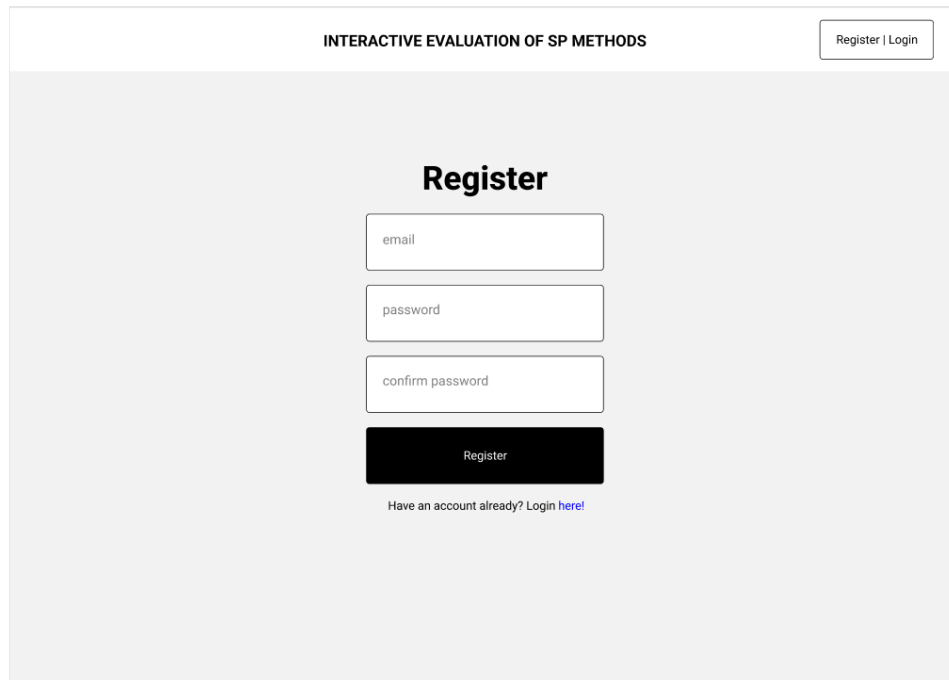


Figure A.2.1.1 – User Information Page



The login page features a central heading "Login" in bold black text. Below it are two white input fields with black borders, labeled "email" and "password". A black button with white text "Login" is positioned below the password field. At the bottom, there is a link: "No account yet? Sign up [here!](#)".

Figure A.2.1.2 – User Login Page



The registration page features a central heading "Register" in bold black text. Below it are three white input fields with black borders, labeled "email", "password", and "confirm password". A black button with white text "Register" is positioned below the "confirm password" field. At the bottom, there is a link: "Have an account already? [Login here!](#)".

Figure A.2.1.3 – User Registration Page

The below Figure A.2.2 depicts the original system architecture of our original design before the implementation of our continuous integration tools like docker as well as other changes on the Frontend and the Server/Algorithm Execution Driver.

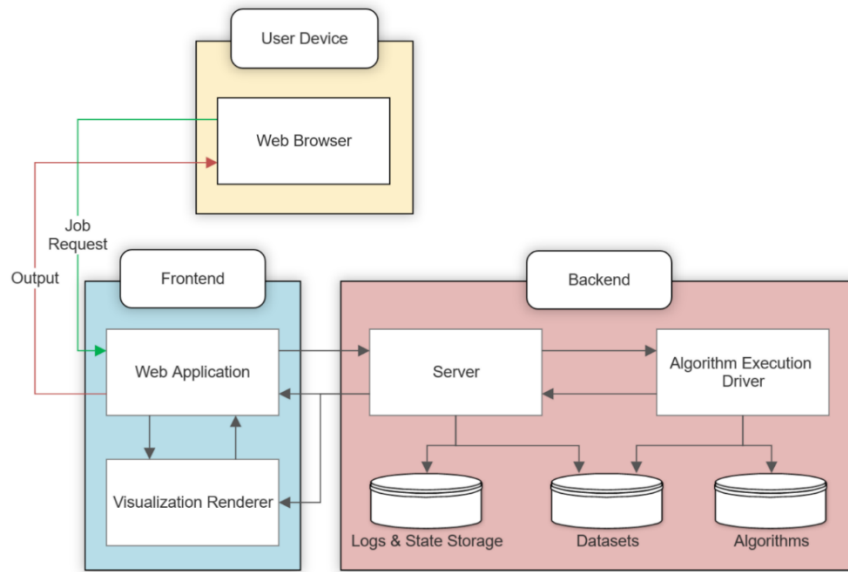


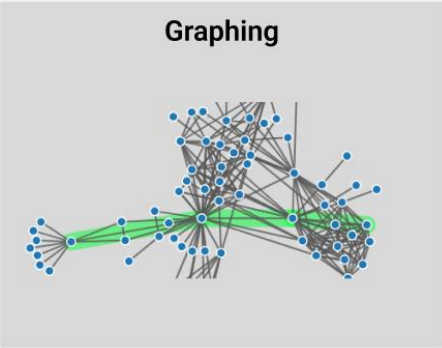
Figure A.2.2 – Original System Architecture

The Figure A.23.3 below displays our original algorithm results page before the visualization and metrics report were moved to separate pages.


INTERACTIVE EVALUATION OF SP METHODSMy Account

Your Algorithm Visualization

Graphing



Mapping



Metrics Report for Your Algorithm Execution

Runtime: ...
Other Metrics Here

Runtime: ...
Other Metrics Here

Save My Report

Figure A.24.3 – Original Algorithm Visualization Page

Appendix III

Applicable Courses from Iowa State University Curriculum

- COM S 228 – Introduction to Data Structures
- COM S 309 – Software Development Practices
- COM S 311 – Introduction to the Design and Analysis of Algorithms
- COM S 319 – Construction of User Interfaces
- COM S 329 – Software Project Management
- COM S 363 – Introduction to Database Management Systems
- CPR E 416 – Software Evolution and Maintenance
- S E 317 – Introduction to Software Testing
- S E 339 – Software Architecture and Design