CS 520
Development: Elevation-based Navigation System: ELeNA

Done By: Abhinav Reddy Yadatha (33476300) - github username : abhinavreddy2499
Divija Palleti (33612743)- github username : divijapalleti
Neha Jhurani (33972962) - github username : nehajhurani
Jaswanth Reddy (33618888) - github username : jaswanth0206

1. Introduction

Conventional navigation systems optimize for the shortest or fastest route between two given locations. However, they do not consider elevation gain. We are developing ELeNA: Elevation-based navigation system with the objective to create a live web/mobile map that makes it easy for users to locate routes with the required elevation, i.e., they can see the routes that have least elevation and the one that has the highest. Individuals, especially bicyclists and individuals with disabilities, can benefit with this application since they can find routes with least elevation and avoid exerting themselves when climbing. Likewise, if a user wants to have a high intensity workout for a constricted amount of time, he may choose to maximize the elevation gain. The user will also have an option to enter the value of x, which denotes the percentage of shortest route under which a suitable route is returned.

2. Software Requirement Specification

- Overview

Proposed Idea - To implement ELeNA, an elevation-based navigation system, we will be using the Dijkstra an A* algorithm to find the shortest route between the source and destination address entered by the system. We will be factoring in the elevation within the algorithm. We will be using MVC architecture. The model will be the algorithm we implement. The view will be the third-party Map like interface (Google Maps/Apple Maps) that we will integrate in our model. The controller will be the server to which we will send our inputs and receive our outputs. We will evaluate the performance of both the algorithms and the navigation system as a whole.

Stakeholders – Professor and Teaching Assistants for coordinating and managing the system development. Students (group members - developers) to

research and develop the navigation system. Users (other group students, other professors, or any other user) operate the navigation system.

- List of Features

The project will have all the features that a conventional navigation system contains, for example – automatic rerouting, Bluetooth Hands Free Calling, Turn-by-Turn voice directions, integrated traffic receivers, etc. Along with it, it will have a feature to get the desired elevation and return the most suitable path within x% of the shortest distance between two points.

- Functional Requirements

They outline the fundamental system behavior under every possible condition. It helps consumers accomplish their objectives; functional requirements are product features that developers must implement. User stories are frequently used to define them. The following functional needs were taken into consideration when developing ELeNA.

User Story - The user needs to choose the source and destination, along with the elevation and a number x which will give us the percentage of shortest distance that needs to be considered.

- ➔ The user should be able to see maps like-interface (for example – Apple Maps, Google Maps).
- ➔ The user should be able to enter the desired source and destination.
- ➔ The user should be able to enter the desired elevation gain which will allow him to either maximize or minimize the elevation.
- ➔ The user should be able to enter the value of x.
- ➔ The result will be shown to the user in the Maps type interface.

- Non Functional Requirements:

This describes the non functional requirements from a stakeholders perspective.

Version Control - We will use Git for version control to track multiple iterations of the code, identify the contributors of specific changes, and record the time of those changes. This method also enables collaborative work on different versions and allows for easy restoration of previous versions in case of any failed attempts. This will help with the **maintainability** of the system.

Usability - The user interface needs to be highly user-friendly and effortless to navigate. Consistency in the color scheme is essential, as it provides the user with a seamless experience throughout the process, from selecting the source and destination to viewing the routes.

Efficiency - To operate efficiently in real-time scenarios with minimal delays, the application must be designed accordingly. The user interface needs to be

optimized to function smoothly and load quickly on web browsers. Quick and real-time responses to user requests are crucial. Employing cache to store frequently searched locations can enhance the system's efficiency.

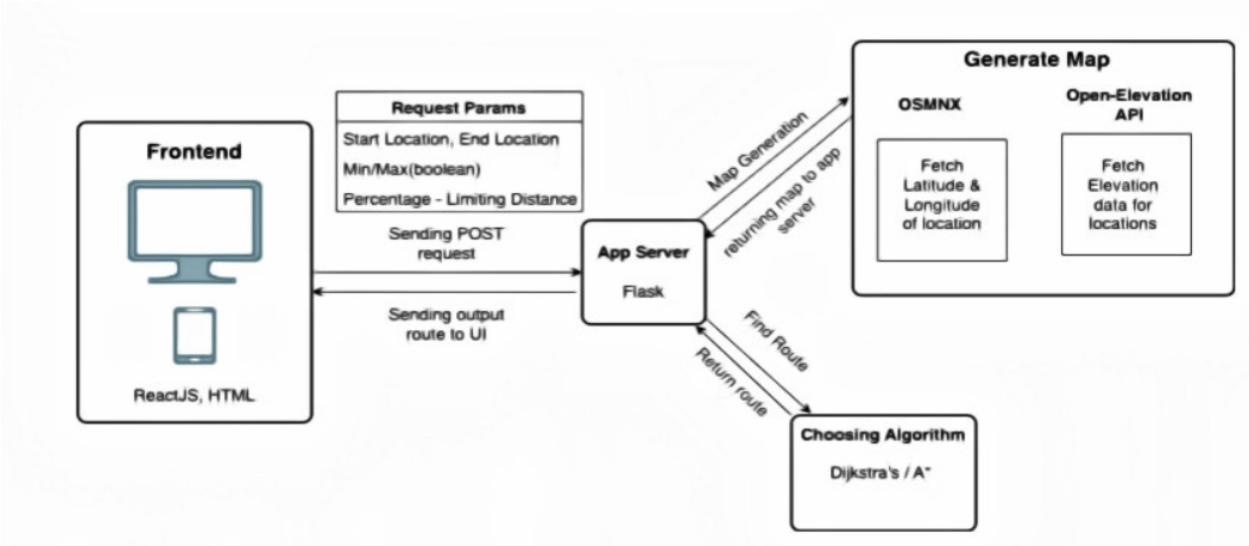
Correctness - Accuracy is among the most vital non-functional requirements.. The route provided by the application should be optimal within the given constraints, while also fulfilling the user's ultimate objective, whether it's to minimize or maximize elevation or to find the shortest path with a wide range of elevation thresholds.

Testability - Adequate unit test, integration and system coverage (including structural code coverage and mutation analysis) is essential to thoroughly evaluate the product, particularly in handling various corner cases. The tests should focus on ensuring the accuracy of the algorithms employed in the application, with regard to the user's objectives of minimizing or maximizing elevation gain.

Reusability - The components in the code should be designed modularly which will ensure that they can be reused and will adhere to the DRY principle, which means avoiding repetition of code. All functionalities in the code should be clearly represented in the system, and any other components that need to use these functionalities should do so through software abstractions to ensure clarity and avoid confusion.

3. Design

- Architecture



We decided to use Model View Controller (MVC) architecture for our project. We wanted to separate the models into different parts so that all of us can work on it simultaneously. We wanted different components to handle different features of our application. It made the maintenance, organization and testing of our application easier. We used python to implement the different APIs for the various features of our project. We also used heap to cache the application, because we have observed that a user inputs the same location a number of times, and storing it in a cache will make our system more effective. For our system (backend) to interact with User, we created a user-friendly UI using HTML, ReactJs. The UI has been developed keeping usability in mind, we used light color pallets and made sure each of the fields required by the user is illustrated explicitly and clearly. For better visualization, we used the openStreet API given by Google. To connect the frontend with the backend, we used Flask framework. This framework allowed us to send the required user input to the python APIs exposed and was able to receive the directions and statistics in return. The statistics included the distance in miles and the elevation of the path proposed by the algorithm.

- Tech stack with justification

We developed our project using different tech stack for the different components. To make the user interface, we used HTML and ReactJs. We used this because it offers us all the features needed to make the UI user-friendly. We did not want to complicate the UI so that it is easy and clear for anyone to use it. We used python for our backend. This is because, we were all familiar with the language and OOP principles could be incorporated into our code. We exposed various APIs that are needed for our code using python. We used MVC architecture so that we could further divide the backend into smaller components so that each of us can work on the system simultaneously. MVC architecture also helps with understandability, testability, and maintainability. To connect both of them, we used flask framework. Flask framework is a light weight and flexible web framework. It is well documented and works really well with python, so it made the perfect framework for us.

- User Interaction

  First, the user enters a source, destination and other constraints w.r.t elevation. Then Maps APIs fetch the latitude and longitude of the source and destination. Then the OSMNX APIs (NetworkX and OpenStreetMap integration) fetch the source, destination and intermediate locations as nodes in a graph.
  OSMNX does this by drawing a bounding box around the source and destination nodes and fetching all the intermediary nodes and constructing a graph based on distances.
  Next step is the routing algorithm : (Dijkstra). These algorithms find the shortest path for the specific user requirements. FInally, the elevation gain in addition to the shortest distance with the map visualization.

- UI design and data model

  The design consists of the following fields : Source, destination, elevation gain and x% of the shortest path. The ui design of the is as follows.

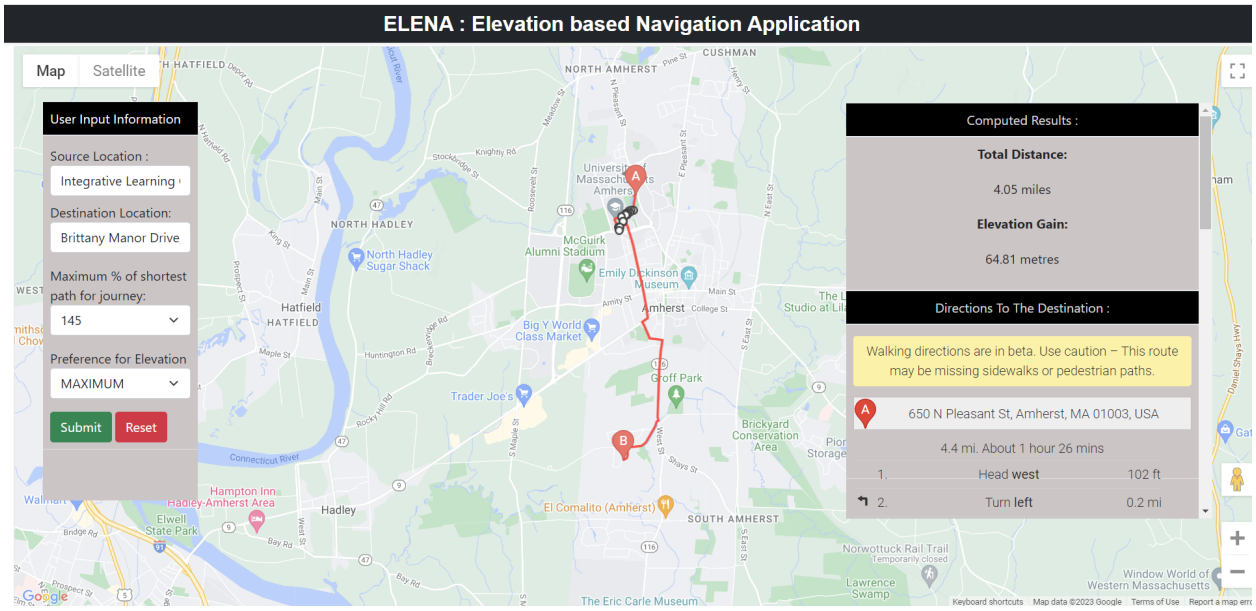


Figure : Screenshot of UI with maximum elevation

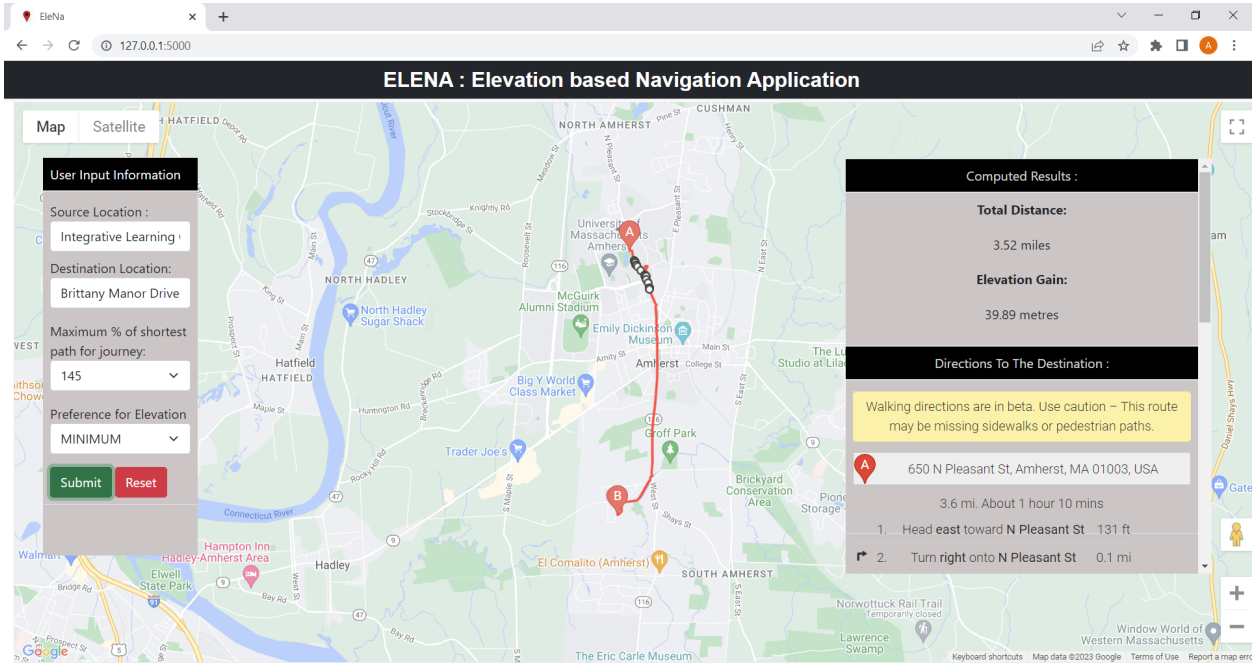


Figure : Screenshot of UI with minimum elevation

Data Model :

We are not using any database to store any of the customer data anywhere. So we are not using any data model and we are computing on-the-fly.

4. Evaluation Criteria

- Test Cases

  Unit Test Cases

  1. Unit tests cases for each component to scrutinize the code independently and detect any flaws in the system in the earlier stages are written.
  2. Code coverage is maintained to check all the functionality of the system.
  3. Some of the test cases include
     a. Testing routing algorithms
     b. Testing the elevation path

```
@Test("")
def test_get_graph(end):
   print("# Testing the get_graph method in MapGraphModel.py......")

   G = Graph().get_graph(end)
   assert isinstance(G, nx.classes.multidigraph.MultiDiGraph)

@Test("")
def test_get_shortest_path():

   print("# Testing get_shortest_path method in ShortestPathController......")

   startpt=(42.3732216,-72.5198537)
   endpt =(42.4663727,-72.5795115)
   G = Graph().get_graph(endpt)
   controller = ShortestPathController(G)
   shortest_path = controller.get_shortest_path(startpt,endpt)
   assert abs(shortest_path.get_distance()-11956.737999999996)<=100
```

```
@Test("")
def test_calculate_elevation_path():
   print("# Testing calculate_elevation_path method in AlgorithmsController....")
   startpt=(42.3732216,-72.5198537)
   endpt =(42.4663727,-72.5795115)
   G = Graph().get_graph(endpt)
   s_controller = ShortestPathController(G)
   shortest_path = s_controller.get_shortest_path(startpt,endpt)
   controller = AlgorithmController(G,shortest_path.get_distance(),float(150) /
100.0,"max",shortest_path.get_origin(),shortest_path.get_destination(),1,2)
   elevation_path=controller.calculate_elevation_path(G, shortest_path.get_origin(),
shortest_path.get_destination(), None,weight=lambda u, v, d:
                                                math.exp(1 * d[0]['length'] * (
                                                            d[0]['grade'] + d[0]['grade_abs'])
/ 2)
                                                + math.exp((1 / 100) * d[0]['length']))
   print("elev path",elevation_path[0])
   assert elevation_path[0]==66704169
   assert elevation_path[1]==6302552856
```

Integration Test Cases

1. We will combine individual components and test the application as a whole.
2. We have tested our application with the places in and around Amherst, this includes:
    a. Keeping a track of the latency.
    b. Keeping a track of the output route.
    c. To ensure the accuracy of the routing algorithms, they are tested using mock graph provider APIs.
    d. The responsiveness of the front-end is evaluated by executing all possible user scenarios this involves testing with both minimizing and maximizing elevation gain.

- Peer Review

The group mates have reviewed each other's code. We have improved the code quality, and identified the bugs that might have been missed during development, improve our code's readability, and help us document things better.
We have also enforced consistent coding style throughout the project through peer review. This helped us check non-functional requirements of the system.

- User Acceptance Testing

    - Final stage of the software development life cycle
    - Actual users are engaged to test the software's ability to perform the tasks it was designed to address in real-world situations.
    - involves the implementation of both Alpha and Beta testing approaches

      Alpha Testing - It is a form of acceptance testing that is conducted to identify any potential issues or bugs before the final product is released to end-users. We conducted individual alpha testing on each component and use that information to enhance our approach by addressing the identified issues.

      Beta Testing - It is a type of testing that involves end-users, as opposed to developers, who test the software application.
      This test was conducted in the inclass - a group of friends/students have tested our application and provides us feedback. A few improvements were made baswed on this feed back. This involves - making the UI more interactive and informational.

Application Backend Screenshot :

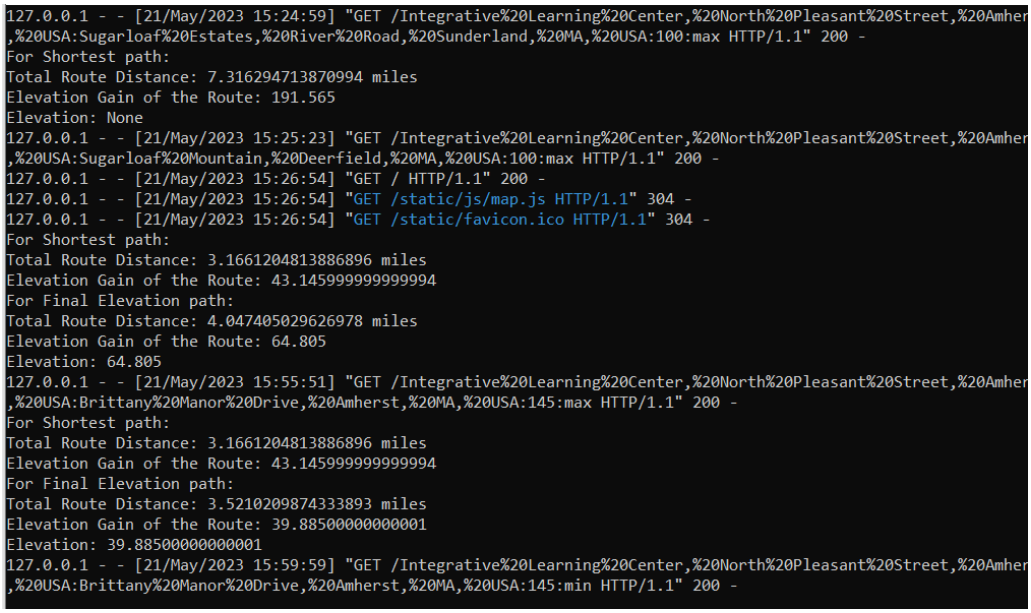
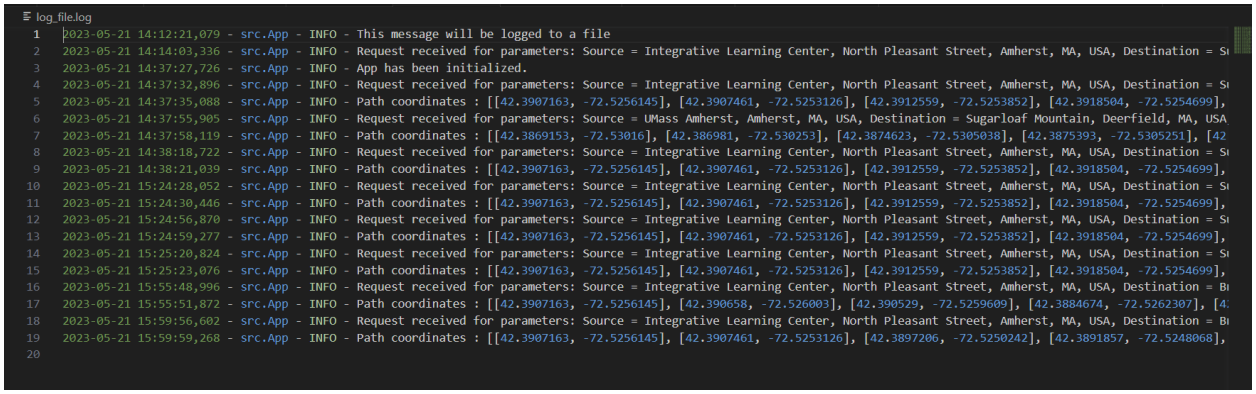


Figure : Backen interaction of the software

Logger statements screenshot :


log_file.log

```
2023-05-21 14:12:21,079 - src.App - INFO - This message will be logged to a file
2023-05-21 14:14:03,336 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 14:37:27,726 - src.App - INFO - App has been initialized.
2023-05-21 14:37:32,896 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 14:37:35,088 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3912559, -72.5253852], [42.3918504, -72.5254699],
2023-05-21 14:37:55,905 - src.App - INFO - Request received for parameters: Source = UMass Amherst, Amherst, MA, USA, Destination = Sugarloaf Mountain, Deerfield, MA, USA
2023-05-21 14:37:58,119 - src.App - INFO - Path coordinates : [[42.3869153, -72.53016], [42.386981, -72.530253], [42.3874623, -72.5305038], [42.3875393, -72.5305251], [42
2023-05-21 14:38:18,722 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 14:38:21,039 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3912559, -72.5253852], [42.3918504, -72.5254699],
2023-05-21 15:24:28,052 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 15:24:30,446 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3912559, -72.5253852], [42.3918504, -72.5254699],
2023-05-21 15:24:56,870 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 15:24:59,277 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3912559, -72.5253852], [42.3918504, -72.5254699],
2023-05-21 15:25:20,824 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = S
2023-05-21 15:25:23,076 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3912559, -72.5253852], [42.3918504, -72.5254699],
2023-05-21 15:55:48,996 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = B
2023-05-21 15:55:51,872 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.390658, -72.526003], [42.390529, -72.5259609], [42.3884674, -72.5262307], [4
2023-05-21 15:59:56,602 - src.App - INFO - Request received for parameters: Source = Integrative Learning Center, North Pleasant Street, Amherst, MA, USA, Destination = B
2023-05-21 15:59:59,268 - src.App - INFO - Path coordinates : [[42.3907163, -72.5256145], [42.3907461, -72.5253126], [42.3897206, -72.5250242], [42.3891857, -72.5248068],
```

Best practices followed in the project:

- We followed the MVC (Model-View-Controller) architecture in our project.

- To avoid using ambiguous or hard-coded strings, we utilized constant strings.

- To enhance code understandability, we added comments to classes and methods, along with a ReadMe file and a requirements file for easy code setup.

- For testability, we implemented unit test cases to validate the functionality of the tool. Additionally, we performed thorough manual testing.

- We adhered to appropriate naming conventions, making the code easier to comprehend.

- Before pushing changes, we conducted manual reviews with team members to ensure the correctness of the modifications.

- Weekly meetings were conducted to maintain team-wide awareness of the project's progress.

- We implemented the Observer design pattern, where the model serves as the observable and the view functions as the observer in our architecture.