Final Project Manual

by

Gleb Myshkin, Bowen Jiang, Matthew Smith

Stevens.edu

December 19, 2024

Final Project Manual

Gleb Myshkin, Bowen Jiang, Matthew Smith
Stevens.edu

Table 1: Document Update History

Date	**Updates**
11/07/2024	DDM: • Added Chapter 1 of MilestoneTwo • In Chapter 1, MilestoneTwo is added, created requirement table, architecture diagram, two activity diagrams for each use case, sequence diagram, and a story board.
12/04/2024	DDM: • In Chapter 1, added a class diagram, fixed the architecture diagram and sequence diagram.
12/07/2024	DDM: • Added Chapter 2 of MilestoneTwo • In Chapter 2, MilestoneTwo is added, created a table of takse broke from Use cases, add meeting notes, Include documentation of each iteration end. • Modified the Class Diagram in Chapter 1 • Added GitHub issue with the tasks being worked on and GitHub Kanban
12/12/2024	DDM: • Added Communication Diagram text • Added Interaction Diagram text • Added Object Diagram text • Updated Class Diagram text

Table 1: Document Update History

Date	Updates
12/17/2024	DDM: • Added Deployment Diagram with text • Added Package Diagram with text • Added Composite Structure Diagram for InvestmentManager with text • Updated Sequence Diagram and added text about it • Added Timing Diagram for BankInvestment Calculation with text • Added New Chapters: Agile Processes and Updataed/Added Chapter 3, Task and Progress Management • Updated Process Reflection and Meeting Notes • Added screenshots of messages between team members for documentation purposes • Updated Index
12/18/2024	DDM: • Updated Class diagram • Updated Activity and Use case diagrams • Added Object, Interaction, and Communication Diagrams with text

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Our group is a three-person group and all three of us are software engineering majors. Bowen and Matthew are in their 3rd year and Gelb is in his 4th year. In terms of work splitting, Matthew did the coding for the project and some diagrams, Bowen and Gelb did the overleaf upload and formatting, and they also did parts of the diagrams and the text. In terms of work splitting, every person gets around 33 percent of work equally with a few percent up and down.

Chapter 2

Final project, Investment Manager

Problem Statement: Trying to decide how to invest is very difficult for most. From your average citizen trying to maximize their savings, to large businesses needing to decide on what projects to go for. Not only are the numbers difficult to understand sometimes, but its also sometimes hard to compare investments that should seemingly be simple, since as all businesses will tell you, a dollar today is worth more than a dollar tomorrow. Our bot will be able to take the information one has, and do its best to try and both A. calculate investment information (how much interest do I need to turn x into y, how much is this project worth if I invest in x amount of money every month at y interest, etc). And B. Compare investments to see which one is more desirable given a factor(s) to compare.

The reason we are making this is to both A. Simplify the process to make it more digestible and easier for others to figure out how investments work, how to do it, and the options available. If you were to ask most people how much they need to invest to have what is worth now 200000 in about 20 years, they wouldn't be able to do it. Our bot makes it both easy and simple to figure that out. B. Make it easy to compare investments that would otherwise take lots of calculations. While in some cases it may seem obvious, there are so many factors in play that its hard to keep track of them all and calculate what you want to know. Even in the highest level of business, its often hard

to figure out what investments are worth the risk, and which ones to pick in a sea of choices. Our bot can easily compare multiple investments, and pick the best one to show off.

What we are doing and why we are doing it:

We are trying to make a bot that can do two things. Firstly, we want it to be able to fill in missing pieces whenever possible about an investment and try to show off all that information to those who might not be able calculate everything, making it easier for the average person to both digest the information and make informed decisions. Secondly, we want it to be able to compare two or more investments and select the best one, given some set of criteria (lowest initial investment to most profit, requires least amount of time to reach a certain amount, etc). This would make it easy for businesses to compare investments in a way that makes it easier to pick one in a potential sea of options. This will be done by using some tables of information, and using formulas and calculations, to fill in information or find information the user asks for. As for comparisons, the business will submit information about two projects, and the bot will return the result that most satisfies the criteria given. It is mostly an analytical bot, showing results given some information the user will most likely know.

Our project is a good solution for these problems because for many individuals, investments are seemingly complicated. With all the factors (inflation, future value of money, how interest compounds, etc), its often difficult for the average person to decide what is a good investment, or how to reach a specific goal. A bot that can tell people what they want/ need to know can make it more likely for someone to invest in the first place, thus potentially allowing people to set themselves up for better futures, or set goals to get them where they want to be. As for businesses comparing investments, this is something all businesses must do if they want to stay afloat. Is the shiny new thing really worth the steep price? Is this project saying it will make a return of 20,000 in 10

years actually worth the time and hassle? If I can only pick one of two projects, how do I know which one to do? These are all questions businesses make time and again, many failing because they didn't do the math or try and figure out the complicated mess, especially smaller ones that cant afford the mathematicians and business experts to do the math the for them. Our bot will be made easy and simple to use, and allow for those business owners to try and make the best, most informed decisions they can, easily comparing and returning the best project or investment to choose.

Our project:

Investments made easy

Table 2.1: Requirements Table

Requirement	**Use Case Needed**
Storage: Create and store investment and project objects (Should)	Use case 1 and Use case 2
User-Input: Take input from user to select and store information (Should)	Use case 1 and Use case 2
Calculations: Calculate missing information given other information (Must)	Use case 1
Comparisons: Take two investment objects and compare specific values (Must)	Use case 2

Use Case List:

Compute investment or project information

1. Preconditions
    - none
2. Main Flow
    - User will try and create an investment/project object by providing all known information[1]. Bot will calculate remaining values if possible [2]. Bot will display all information known [3]
3. Sub Flow
    1 User will select to create an investment or project object, and be given a list of variables one at a time they can either input or bypass using specific key (”Unkown” or ”NA”)
    2 Bot will select which formula to use that is stored in the program based on the missing information. If not possible, will return that there is not enough information, otherwise the object is created and stored
    3 If created, bot will display all information found, then all the information.

4. Alternative Flow

    1- Not enough information is given. Bot will ask for more information or if the request is canceled

Compare investments/projects

1. Preconditions

    - 2 or more project or investment objects exist

2. Main Flow

    - Bot asks for the criteria to base the comparison on, and for what type of object[1]. Bot will either take variable or calculate value for all objects and insert them into a list[2]. Bot will return ordered list of objects based on said criteria[3]

3. Sub Flow

    1 Bot will ask for which type, than show the user a list of criteria they can ask to compare based on the users input. The user will give a number representing the one they want

    2 If a variable, bot will add objects into a new list based on said variable, showing the name and value as a list, resulting in a 2d array. If a calculation is needed, the bot will go through the list of objects, calculate the variable desired, and store (as a list) the name of the object, and the value.

    3 The list will be sorted based on the desired critera in order, and displayed to the user

4. Alternative Flow

    1- Not enough objects, bot will return ”comparison not possible”

    2- Either a calculation or just a varibale is needed

Table 2.2: Use Case Table

UseCase	**Requirements fulfilled**
Use case 1 (Determine missing information)	Requirements fulfilled:User input, Calculations, Storage
Use case 2 (Compare investments on given criteria)	Requirements fulfilled:User input,Comparisons

Class and Profile diagram:

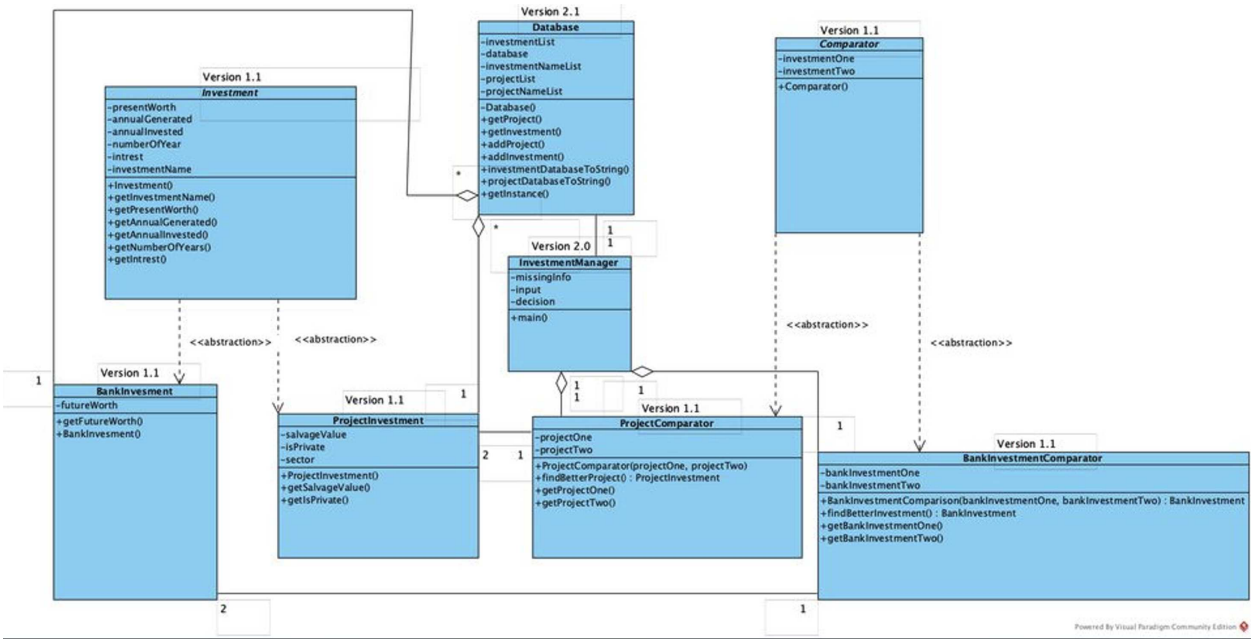

Figure 2.1: Class and Profile Diagram for Architecture Design

The reason we have combined the two is because they are incredibly similar, since many things included in a profile diagram are not a part of this project.

For this project, InvestmentManager is the main class, as it both runs main, and takes in a lot of information, so it does the heavy lifting, taking the users inputs and doing with it whatever is necessary. Investments is an abstract class,with concreate ProjectInvestment and BankInvestments classes that include extra information. Database is our singleton pattern, as it is only constructed once, and stores all the known projects and bank investments in arrayLists, as well as the names

individually so that they can be called later. Finally, we have the Comparator abstract class, with concreate BankInvestmentComparator and ProjectComparator, which takes in 2 investestments of its type, and calculates the benefit to cost ratios, and finally returning the investment with the higher ratio. The investments are a congregation of database, since multiple of them are stored and somewhat depend on Database. They are also linked to their comparator of choice since they are used as arguments when constructing the Comparator objects. Database and Investment Manager, and the Comparators are all linked since they either call eachother, or take information from eachother.

Design Pattern:

➢ **Singleton design pattern**

Database.java

```
1   import java.util.Scanner;
2   import java.util.List;
3   import java.util.ArrayList;
4   public class Database{
5       private ArrayList<ProjectInvestment> projectList = new ArrayList<ProjectInvestment>();
6       private ArrayList<String> projectNameList = new ArrayList<String>();
7       private ArrayList<BankInvestment> investmentList = new ArrayList<BankInvestment>();
8       private ArrayList<String> investmentNameList = new ArrayList<String>();
9
10      private String database;
11      private static Database instance;
12      private Database(String database){
13          this.database = database;
14      }
15
16      public static Database getInstance(){
17          if (instance == null){
18              instance = new Database("theDatabase");
19          }
20          return instance;
21      }
```

Figure 2.2: Singleton Design Pattern

For this project, we used a singleton design pattern with our Database class. We only every call it once, and it cannot ever be called again. The reason for this is that Database just stores everything in its own list, and as such, there is no programming need to call it twice, as it would only create more problems.

Architecture Diagram:

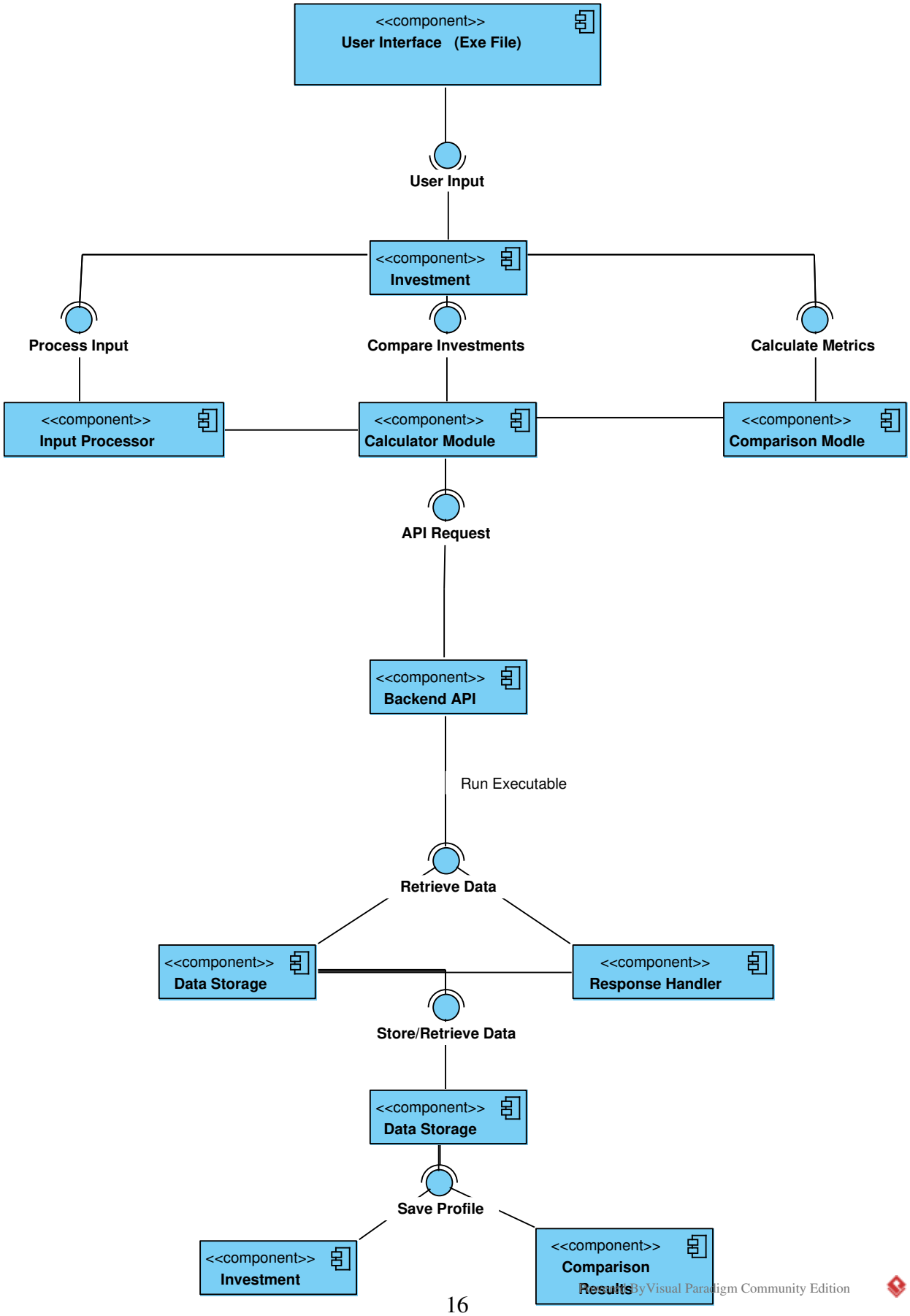

Figure 2.3: Component Diagram for Architecture Design

The architecture of the investment comparison bot is organized into several distinct layers, each fulfilling a specialized role within the system. This layered structure provides clarity, modularity, and a clear separation of responsibilities, ensuring that each part of the system can be developed, maintained, and scaled independently.

The User Interface Layer is the entry point of the system, designed to provide user-friendly experience for users to interact with the bot. The interface allows users to input investment data, and compare two or more different projects. It's responsible for collecting user input,displaying calculated outputs, and handling any feedback or prompts for additional information.

The core of the bot is in the Investment Bot Layer, which contains the primary modules responsible for processing user inputs, performing calculations, and compare two set of data. Within this layer, the Input Processor validates incoming data to ensure it is complete and accurate, if extra information is needed, the bot will also ask for that to determine if the information is enough to do the calculation/comparison, while the Calculator Module performs financial computations such as calculating future values and returns on investment. The Comparison Engine allows users to compare multiple investment options based on the criteria provided. Additionally, the Response Generator formats the results into user friendly responses, and the Error Handling Module manages issues such as missing information, guiding users to provide complete inputs. This layer acts as the main engine of the bot, transforming raw user data into actionable insights.

Supporting the Investment Bot is the API, which serves as an intermediary between the bot and the data storage system. The Request Handler in this layer manages incoming requests from the bot for data storage. By enforcing business rules and managing data operations, this layer ensures that the bot operates in a secure and efficient manner. This layer is crucial for ensuring data consistency and upholding security standards.

The Data Storage Layer provides persistent storage for all user and bot generated data. It contains repositories for user profile data, investment records, calculation history, and comparison

results. This layer enables the system to restore important information, allowing users to access past calculations or view stored comparison results. When users initiate new calculations or comparisons, the Data Storage layer supports efficient retrieval of relevant data, ensuring a consistent user experience. It interacts closely with the Business Logic layer to manage data requests and securely store user information.

An integral part of the architecture is the Response Handler, which ensures that the output from the Calculator and Comparison Modules is translated into user friendly responses. After computations are completed, the Response Handler formats the results, emphasizing clarity and readability. It prepares the data for presentation on the User Interface, organizing information in a way that highlights key insights and facilitates easy interpretation.

Overall, the architecture is designed with modularity in mind. Each component has a well defined role, and their interactions are managed through clear interfaces. This separation of parts allows for easier maintenance and moreover, the use of an executable file for the User Interface simplifies deployment and ensures that users can access the bot without complex installation procedures.

Deployment Diagram:

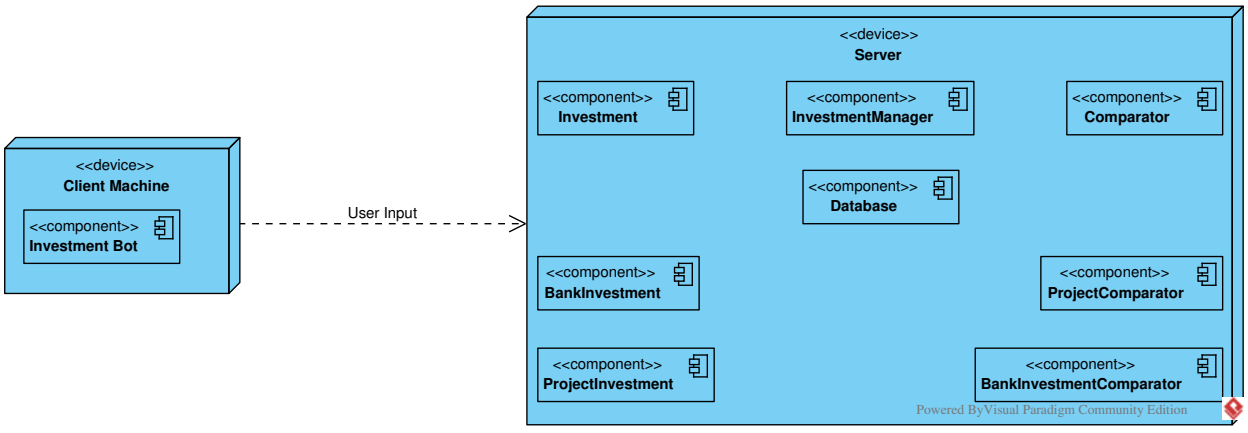

Figure 2.4: Deployment Diagram

The Deployment Diagram shows a simple overview of how the bot works and which devices

it uses. Since the bot works on the device of the user, there will only be devices for the nodes, and no actual server or anything else such as a cloud service. The deployment diagram only has two simple nodes, the Client Machine and the Server. The client machine is where the bot is located and how it is interacted with by the user (when they open it and run it on their devoce). The other Node is the Server, where all of the components for the bot are located (all the classes). Here we have every class of the bot and this is what allows the bot to actually function and send information back to the client. Even though the node is called Server, there is no actual server that the bot connects to, and everything is done on the local client's machine.

Package Diagram:

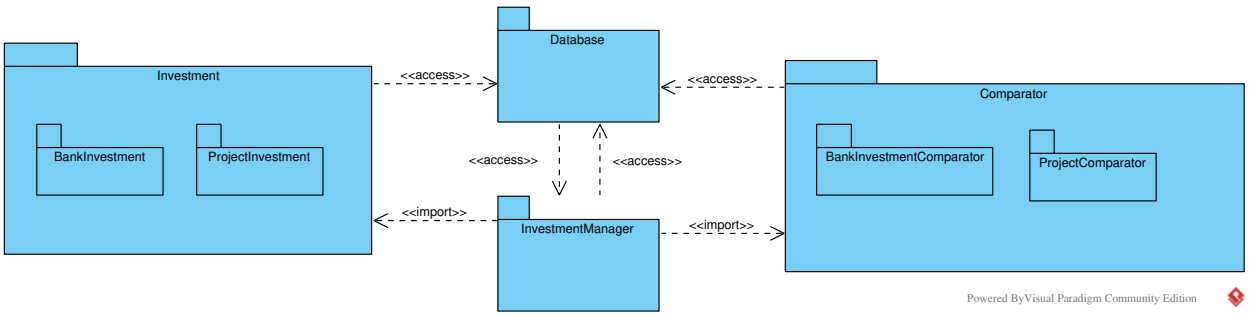

Figure 2.5: Package Diagram

The package diagram has 4 main packages, with two of them having sub packages. The main packages are the Database and the InvestmentManager. The database keeps all of the data that the other packages calculate and gives it back to the InvestmentManager whenever the client asks for any data. Both the Database and the InvestmentManager have access to each other so that they can see and use the data inside of one another. For the two other packages, we have the Investment and Comparator packages. The Investment package contains the BankInvestment and the ProjectInvestment packages. The investment package is the package that allows the bot to create investment data either for the Bank Investments or the Project Investments. On the other hand, we have the Comparator package. This package contains the BankInvestmentComparator and ProjectComparator packages. These packages allow the bot to compare and show the best

investments from either the Bank or the Project. The Investment project has access to the database so that it can calculate the needed data and the Comparator also has access to the database so that it can compare the different projects/bank investments the user asks for. The InvestmentManager also imports the Investment and Comparator packages so that it can use them to manage everything and get the needed results for the user.

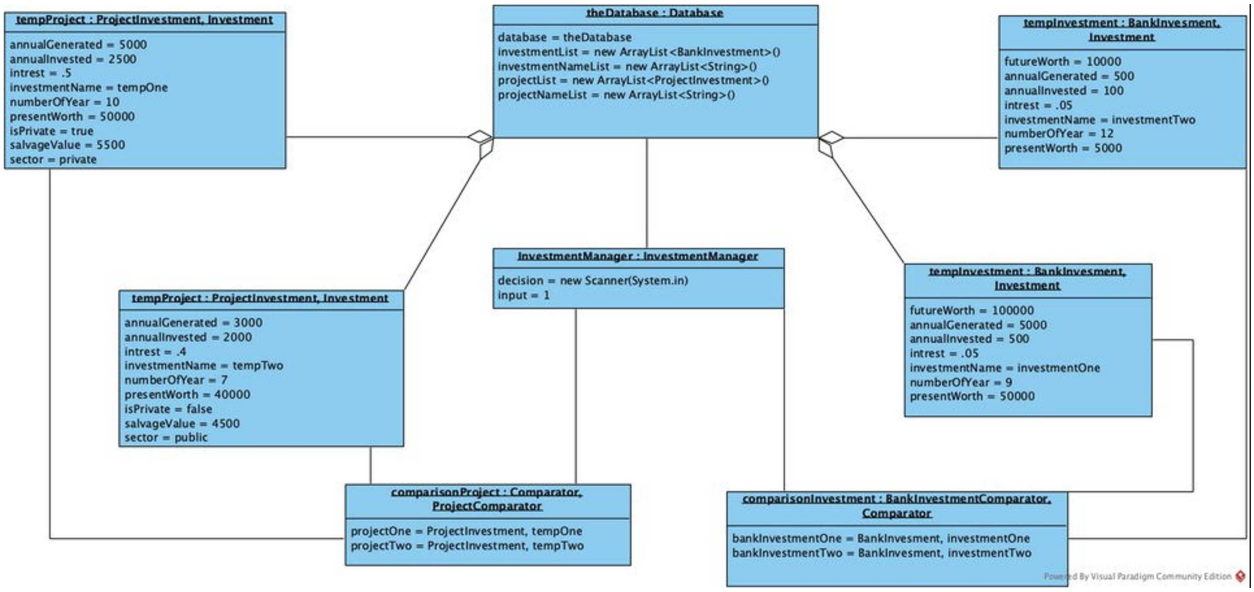

Figure 2.6: Object Diagram

For the object diagram, there have been two ProjectInvestment and two BankInvestment objects created, which are in the Database. They are also linked with the Comparators for the same reason as in the class diagram. The Database serves as the central repository where the two ProjectInvestment and two BankInvestment objects are stored. These objects represent specific investment scenarios, each with attributes such as annualGenerated, annualInvested, interest, numberOfYear, futureWorth, and presentWorth. The clear instantiation of these objects reflects how the system tracks and organizes investment data for both project and bank based investments. The ProjectComparator and BankInvestmentComparator objects are also linked to the investments, facilitating the comparison process. The comparator objects ensure that comparisons are handled systematically, likely using predefined criteria such as benefit/cost ratios or other performance metrics. In

addition, each investment object is shown with its current state, including values for generated income, invested amounts, and other attributes that influence decision making. By connecting these investment objects to the Database and Comparators, the object diagram demonstrates how the system integrates and organizes runtime objects to carry out its primary task: comparing projects and investments.

Composite Structure Diagram:

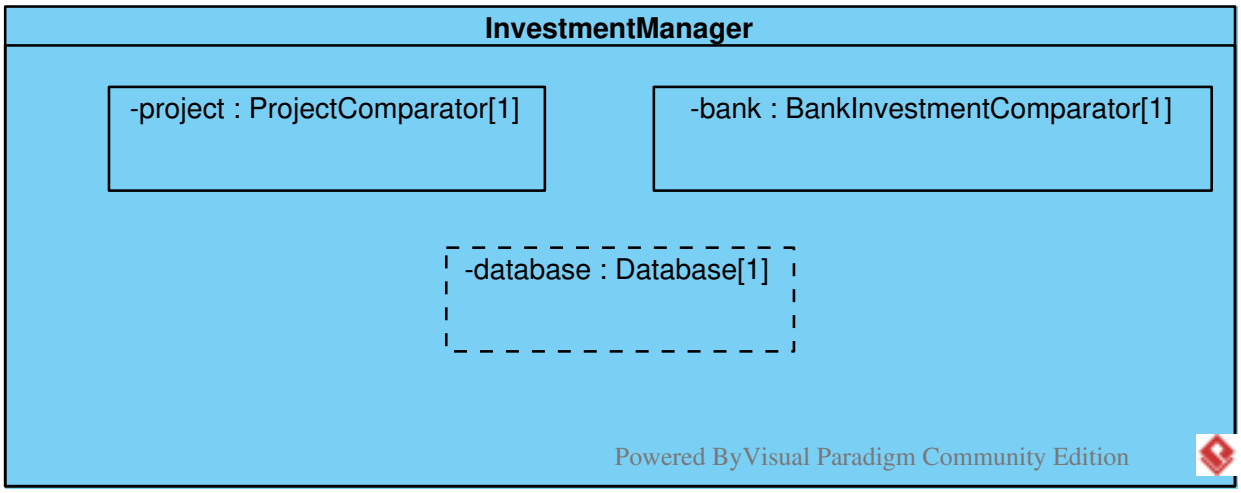

Figure 2.7: Composite Structure Diagram for InvestmentManager

This Composite Structure Diagram takes a look at the InvestmentManager class. It does not take into account the whole class diagram because the whole point of this diagram is to take a more detailed look inside of a class. In the Composite diagram, we have three different parts of the class that are connected to the InvestmentManager class. We have the aggregated parts of ProjectComparator and BankInvestmentComparator. We also have the external diagram of the Database (dotted box) because it is not aggregated to the InvestmentManager class. The names are self explanatory of the parts because they are related to the classes of the project (project, bank, and database names for the parts). The multiplicities of the parts are all 1 since that is all we need for this parts (classes), taken directly from our class diagram. There are also no connections between the parts because the only connection the parts have are with the class InvestmentManager.

Activity diagram 1:

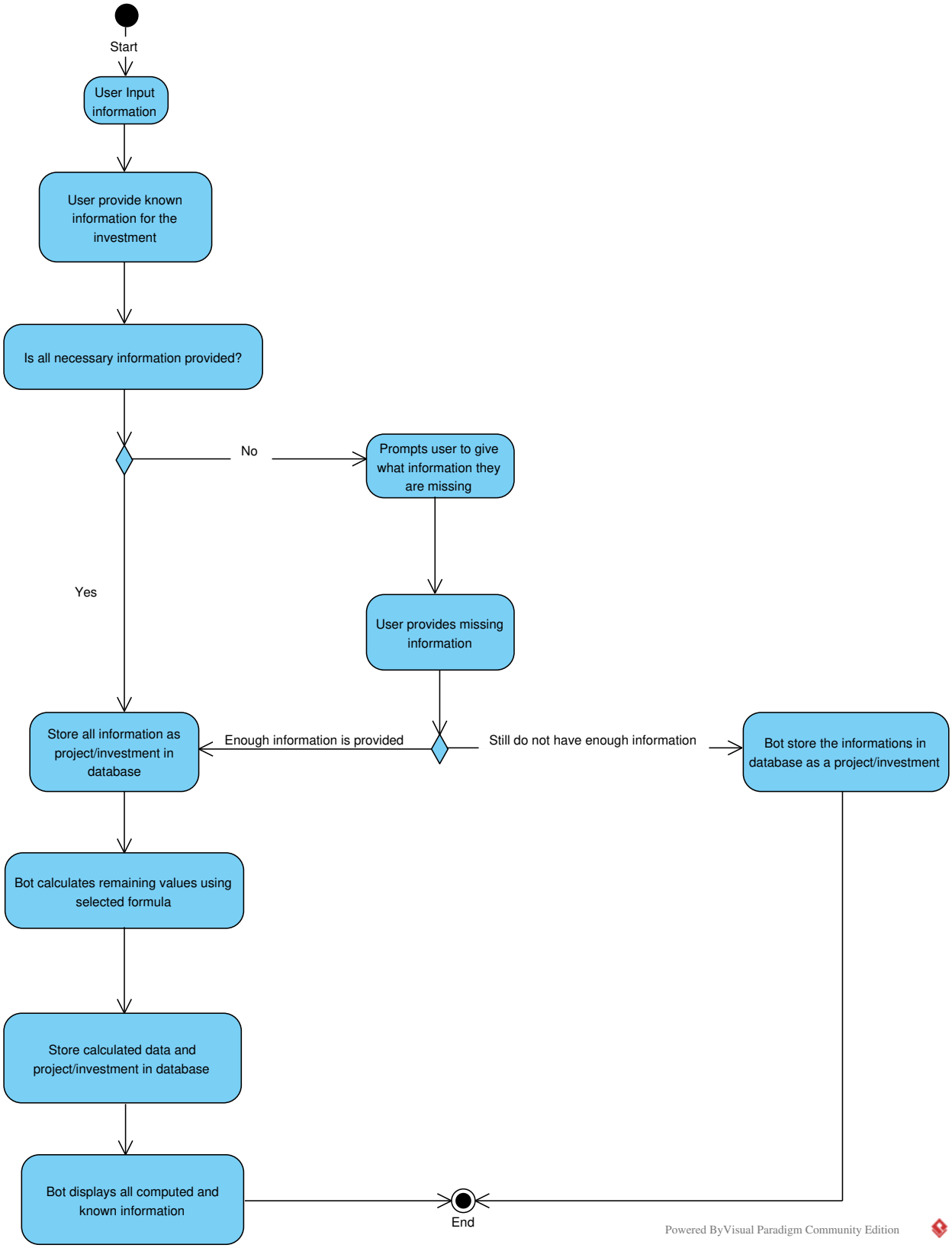

Figure 2.8: Activity Diagram for Use Case One

This activity diagram outlines the workflow for the investment comparison bot as it processes user input and conducts necessary calculations. The process begins with the user entering basic information about an investment, such as the initial amount, interest rate, and investment duration. This initial step ensures that the bot has some foundational data to work with, setting the stage for the subsequent steps in the workflow.

Once the user inputs their information, the bot evaluates whether all required information for the investment calculation is available. If any critical data is missing, the bot prompts the user to provide the missing information by specifying unknown variables. When the bot has all the required information, it proceeds to select the appropriate formula for the calculation based on the type and completeness of the data.

After completing the calculations, the bot displays all known and newly calculated information to the user. However, if the bot determines that it still lacks enough information after prompting the user, it informs the user of the insufficient data, indicating that it cannot proceed with the calculations. Activity diagram 2:

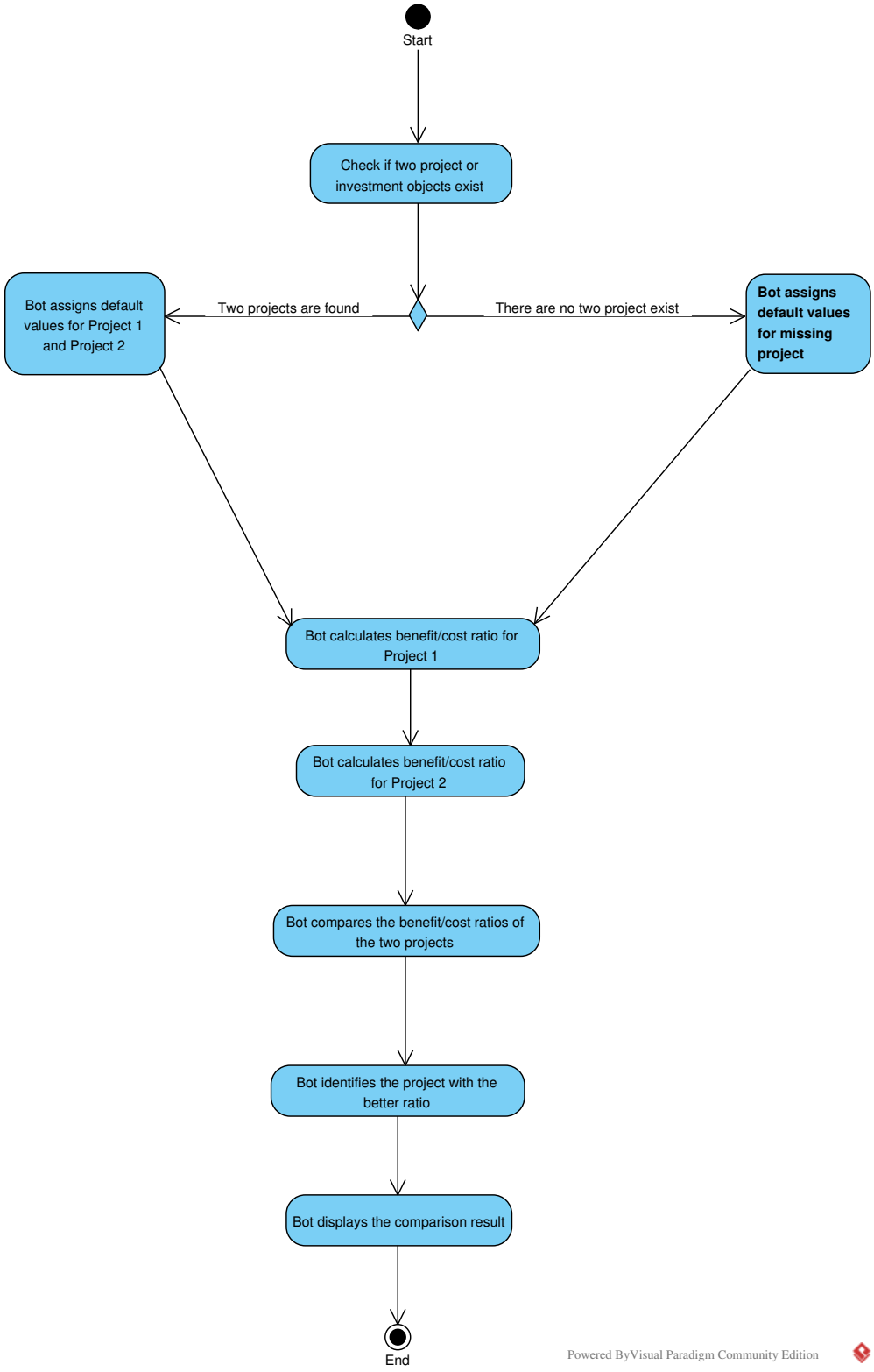

Figure 2.9: Activity Diagram for Use Case Two

Once two or more objects are found, the bot asks the user to specify the criteria for comparison. This might include parameters such as return on investment, risk level, or time to reach a financial goal. After the user selects the comparison criterion, the bot evaluates whether the chosen criterion is a simple variable or requires a calculation. If the criterion is a direct variable (ex: initial investment amount or current value), the bot retrieves these values directly for each object, creating a 2D array that organizes the objects and their values in preparation for sorting.

In cases where the criterion requires calculation (ex: projected future value or estimated returns), the bot performs the necessary calculations for each object. It then stores the calculated values, creating a list of objects with computed results. The bot then sorts these calculated values, arranging the objects in order according to the user's criterion. Finally, the ordered list is displayed to the user, showing the comparison results in a clear and accessible format.

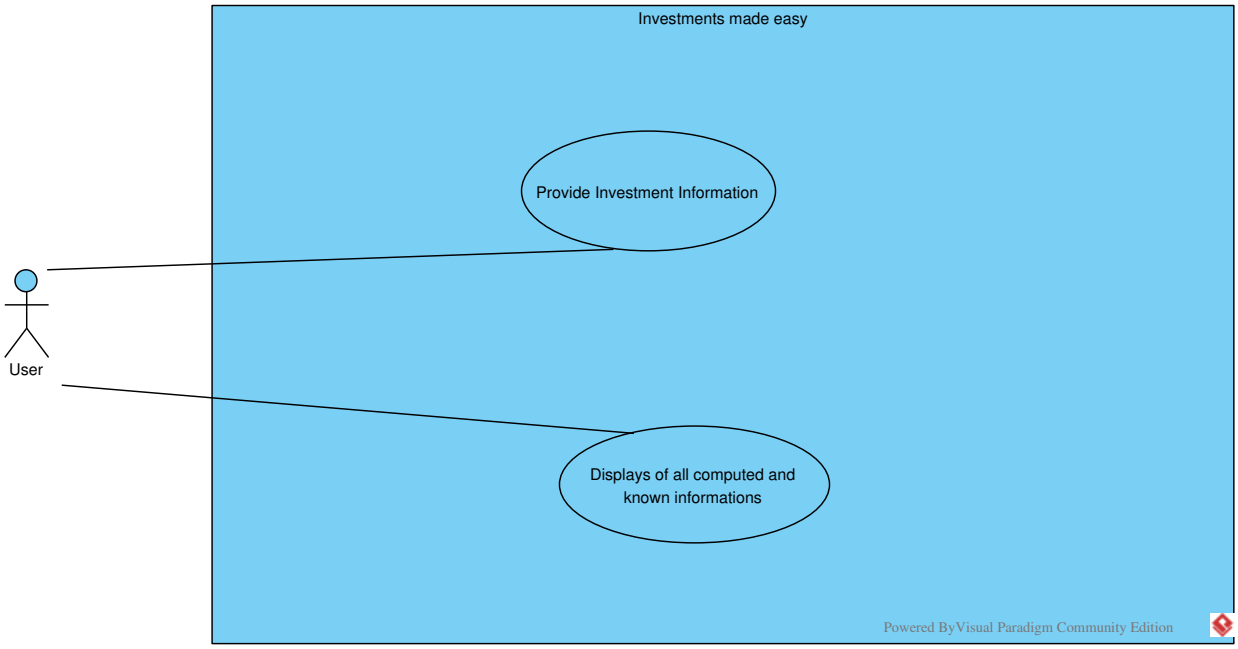

Figure 2.10: Use Case Diagram For Getting Data

This Use Case Diagram for Getting Data illustrates the interaction between the User and the app system. The user performs two primary actions that consist of provide investment information and watch all the display of all computed and known information. The first use case represents the

user inputting the required investment details into the system, such as project or investment related data. Once the system processes this input, it calculates additional values, validates completeness, and stores the information as needed. The second use case indicates the final output where the system displays all the processed, computed, and user provided investment data back to the user.

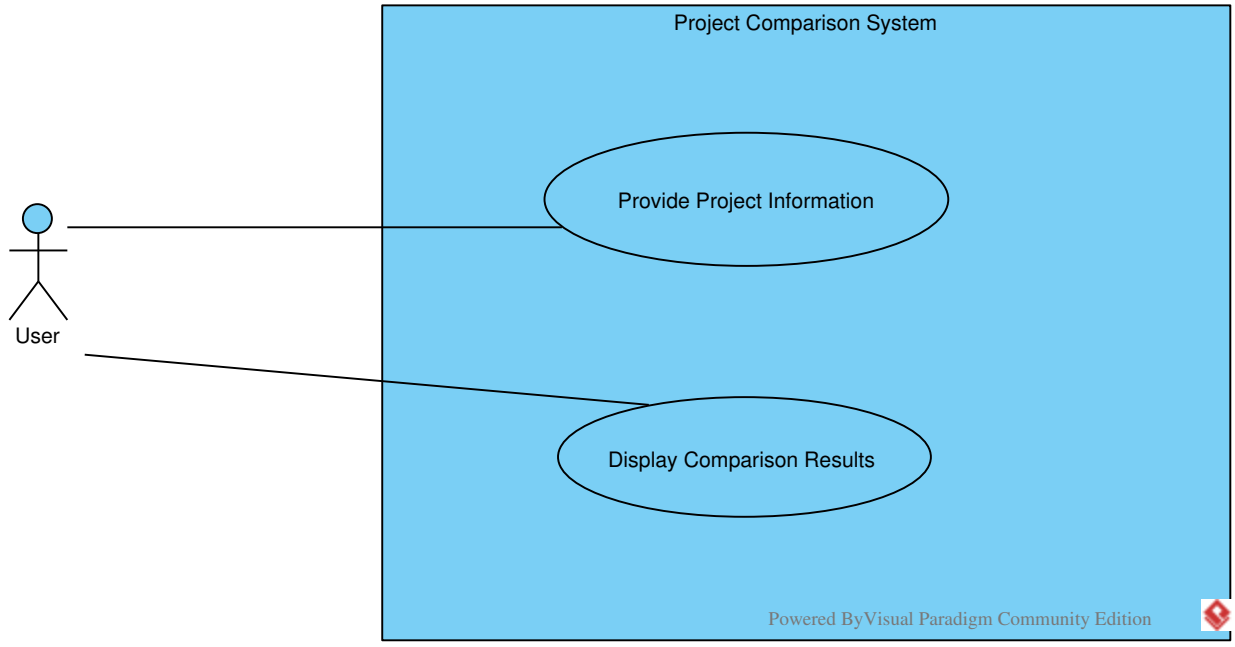

Figure 2.11: Use Case Diagram For Comparing Data

This Use Case Diagram for Comparing Data illustrates the interactions between the user and the app system. The user initiates the process by providing project information as input, which the system uses to perform the required computations. The system processes the data to calculate and compare key values such as benefit to cost ratios for various projects. Finally, the system displays the comparison results to the user, enabling them to assess and determine the better project based on the provided and computed data.

Design Sketches: Sequence flow Diagram:

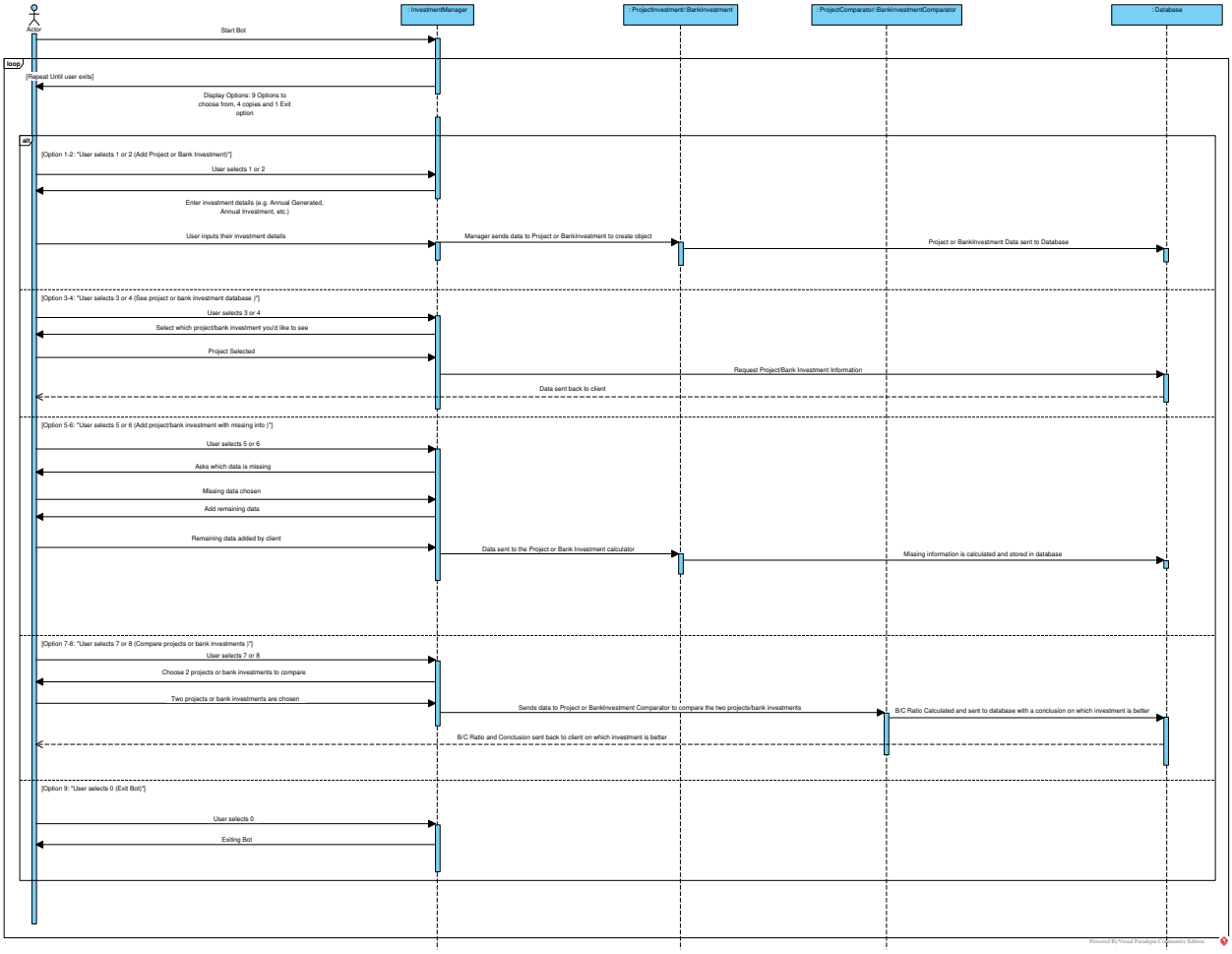

Figure 2.12: Sequence Flow Diagram for the Bot

The Sequence Flow Diagram shows all of the steps of the functions that the bot offers. In total, there are 9 options the user can take while having the bot activated. There are 4 copied functions withing the bot (8 total functions) and 1 function to exit out of the bot and finish its loop. The reason why the sequence flow diagram shows 4 copies of the functions and not every single function (which would make a total of 8 plus the 1 exit function) is because they have pretty much identical sequences in the bot, the only difference being is if the user wants to add or see a Project or a Bank Investment. In the beginning of the sequence diagram, the user (Actor) starts the bot, which would initiate a loop that would be indefinite until the user exits the bot, a bit more on that later. After the bot starts up, the user can choose from 9 different options. If the user decides to pick option 1 or 2, which have the same function but for different aspects (project or bank investment), the user would be able to add either a project or bank investment to the bot's database. These first two options don't send anything back to the user, it's only stored in the database. If the user decides to choose option 3 or 4 of the bot, they would be able to choose to see any added project or bank investment to the database. In this case, the user will receive something back from the database. All of the functions in the bot go through the InvestmentManager and will either go through the Project-Investment/BankInvestment object or ProjectComparator/BankInvestmentComparator, depending on what the user picks from the bot's options. Now, moving onto the user picking option 5 or 6, they would be able to add a project or bank investment to the database with missing info, which would be automatically calculated by the bot and keep the calculated info inside of the database without sending anything back unless asked with a differnt option of the bot's functions. The selection 7 and 8 are used for the comparison aspect of the bot. In these options, the user can choose to compare two different projects or bank investments and have the bot return the better choice to pick from by using their benefit/cost ratios. If the user decides to stop using the bot, they would have to pick the ninth option (in this case option 0) to exit out from the bot and end the loop.

Design Sketches: Interaction Diagram:

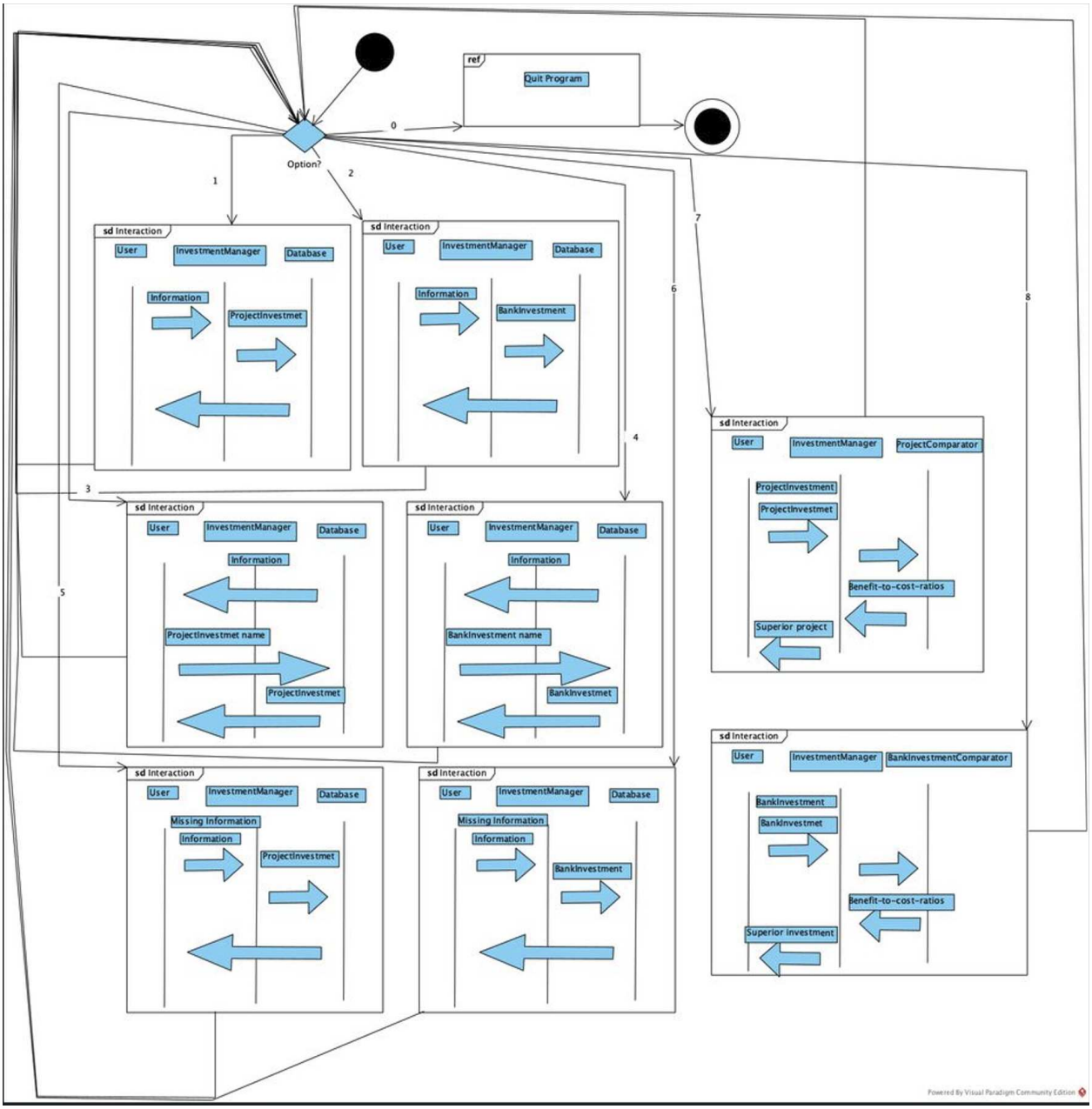

Figure 2.13: Interaction Diagram

For the Interaction Diagram, as you can see, the InvestmentManager often serves as a mediator, and the loop is still there until the user decides to actually end it. For options 1-6, it just goes between the user, InvestmentManager, and Database. The user provides whatever information is necessary, and then Investment Manager either (1-2): Creates the Investment object with it and

puts it back in the database, then returns to the user for another decision, (3-4): Requests the Database, and then asks for which Project/BankInvestment the user wants to see, and displays it, or (5-6), Creates an Investment concreate object based on it, while calculating the rest of the information. For when the user picks 7-8, the User gives the name of two Investments, which is then passed onto the Comparator. It retreives it, and then calculates the benefit-to-cost ratio for both, and finally returns the Investment with the higher ratio. All of these go back to the start until the user passes 0, which then ends the program.

Design Sketches: Communication Diagram:

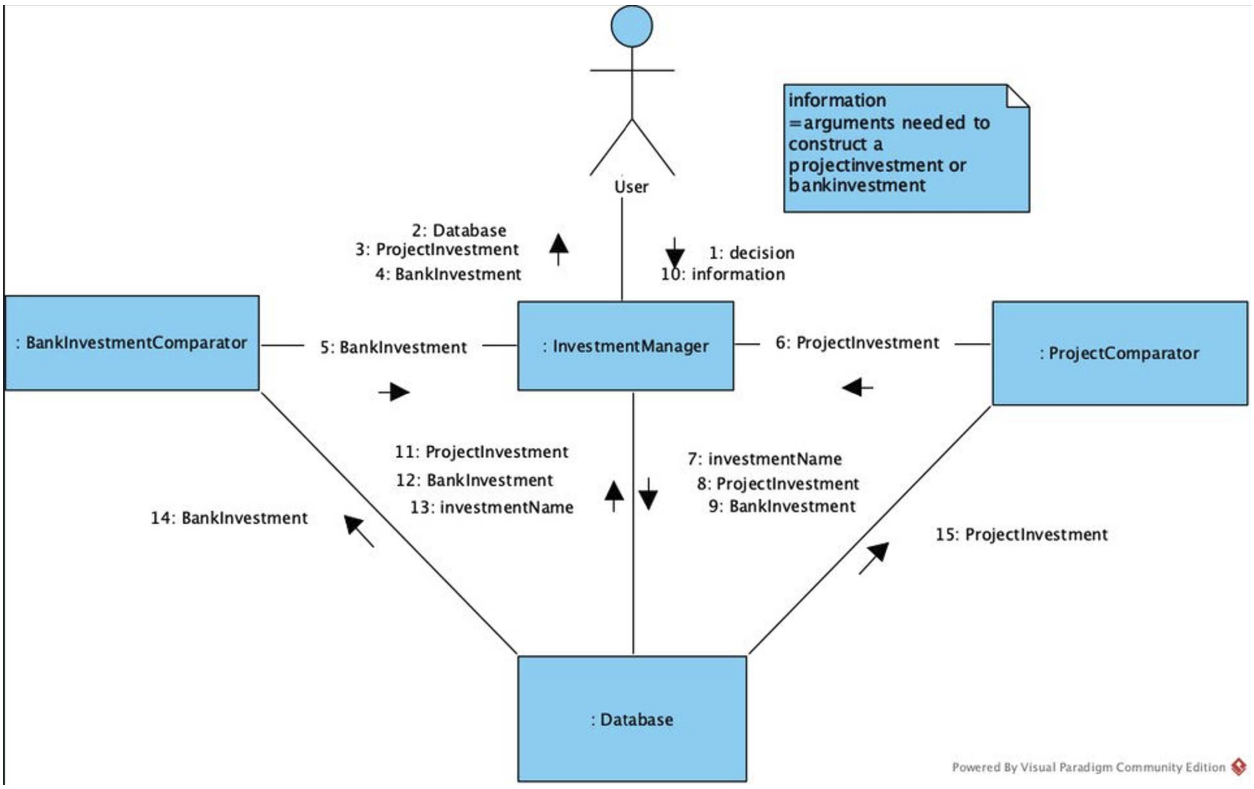

Figure 2.14: Communication Diagram

For the Communication Diagram, the User will pass on either their decision, or information asked to Investment Manager. It then sends out that information to Databse by creating one of those Investment objects, or give it a name so that it can send those projects to the comparators.

From there, the comparators will return the better investment to InvestmentManager, and they will in turn give it to the user. InvestmentManager here essentially acts like a hub, with which the user must interact with. It will also sometimes give the user just the database to see all the projects. The comparators evaluate the given investments based on predefined criteria such as future worth, interest rates, or annual generated values. Once the evaluation is complete, the comparators return the better investment choice back to the InvestmentManager. Acting as the intermediary, the InvestmentManager then relays this information back to the User, providing clarity on which investment is more beneficial. In scenarios where comparisons are unnecessary, the InvestmentManager may simply provide the user with access to the Database to display all projects and investments currently stored. This allows the user to directly view and analyze the available data.

Timing Diagram:

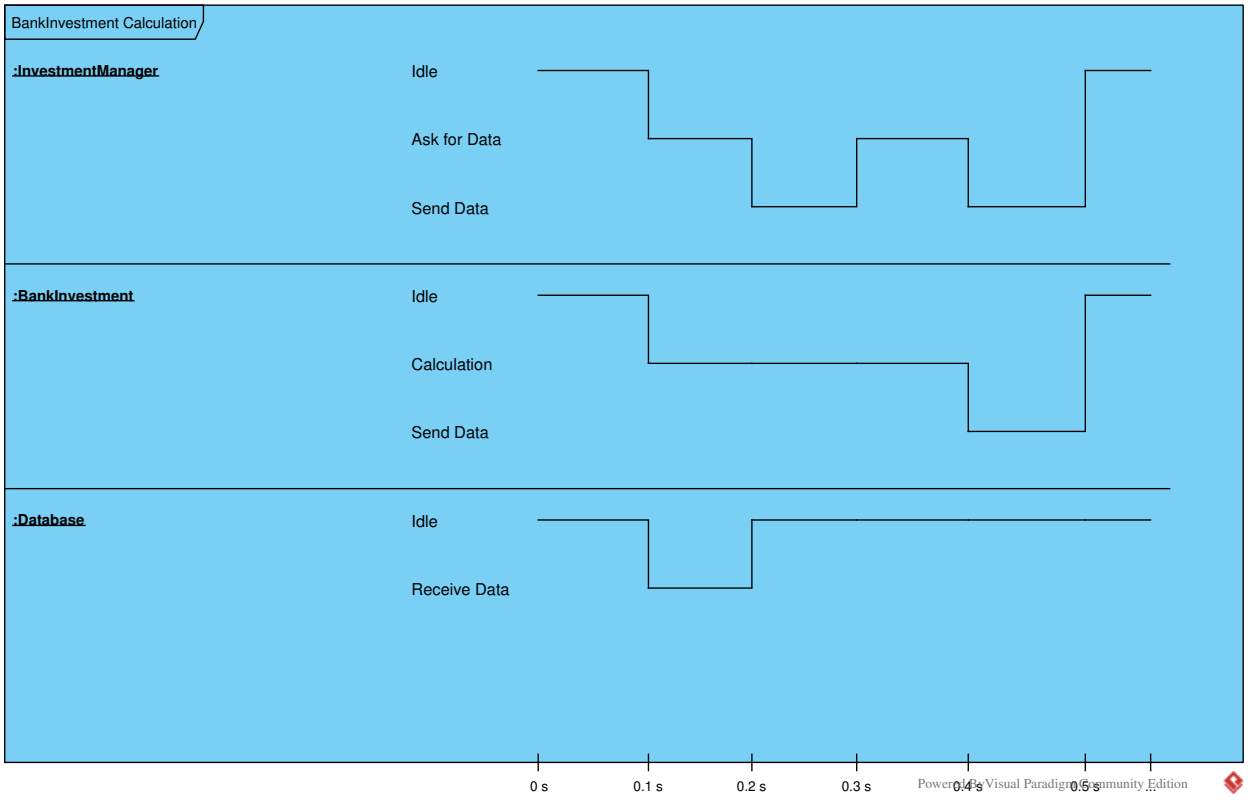

Figure 2.15: Timing Diagram for BankInvestment Calculation

This Timing Diagram takes a look at the BankInvestment calculation that the bot can do. This is the function that uses the InvestmentManager, BankInvestment, and Database classes in order to calculate any missing data that the user could possibly have for a potential bank investment. This diagram is pretty simple as it shows three different classes that have no more than three different states. This time diagram also does not account for the time that it takes the user to input the data, only the time it takes for the bot to do the process. The time key at the bottom could potentially not be fully accurate as full tests have not been done to get an accurate reading of how long it would take to accomplish this task. For the InvestmentManager class, it starts off in idle position, then instantly asks for data, once it gets the data from the user, it instantly asks for which part of the bank investment is missing, and then once it gets that information from the user, it sends the data to the BankInvestment class and goes into idle, all of which is done very quickly. The BankInvestment class starts off idle, but once it gets the data from the InvestmentManager, it takes a little bit longer to calculate the data, then sends it off to the database and goes back to idle. The Database class only has two states, one where it is idle and the other is when it receives the data from the BankInvestment class and then returns back to idle as it does not send anything back to the user unless asked with a different option from the bot, and it is the fastest class withing this Timing Diagram.

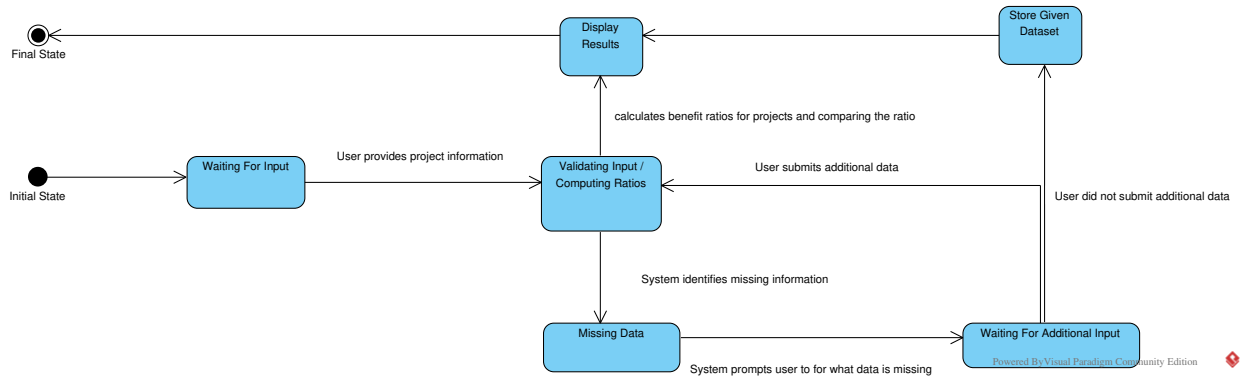

Figure 2.16: State Machine Diagram

The State Machine Diagram shows the transitions between different states of the system as the user interacts with it. Starting from the Initial State, the system enters the Waiting For Input state where the user provides project information. If the input is valid, the system transitions to the Validating Input and Computing Ratios state. Then, the system processes the provided data, calculates benefit/cost ratios for projects, and compares results. If the system identifies missing data during validation, it transitions to the missing data state, prompting the user for additional required information. From here, the system moves to Waiting for additional input, where it awaits further input from the user. If the user provides the missing information, the system re enters the Validating Input and Computing Ratios state. However, if the user does not submit additional data, the system transitions to Store Given Dataset, saving whatever information has been provided so far. Upon successful validation and ratio computation, the system proceeds to the Display Results state, where the calculated project comparison results are shown to the user. Finally, the process ends at the Final State, completing the interaction flow. This diagram effectively demonstrates how the system manages user inputs, handles incomplete data, and transitions between states to achieve its objective of computing and presenting project comparisons.

Chapter 3

Agile Processes

The main way that we as a team communicated between each other is through text messages. We told each and every group member what we will be doing and when we will be doing that. We also had some in person meetings in the Library on the Stevens Institute of Technology Campus.

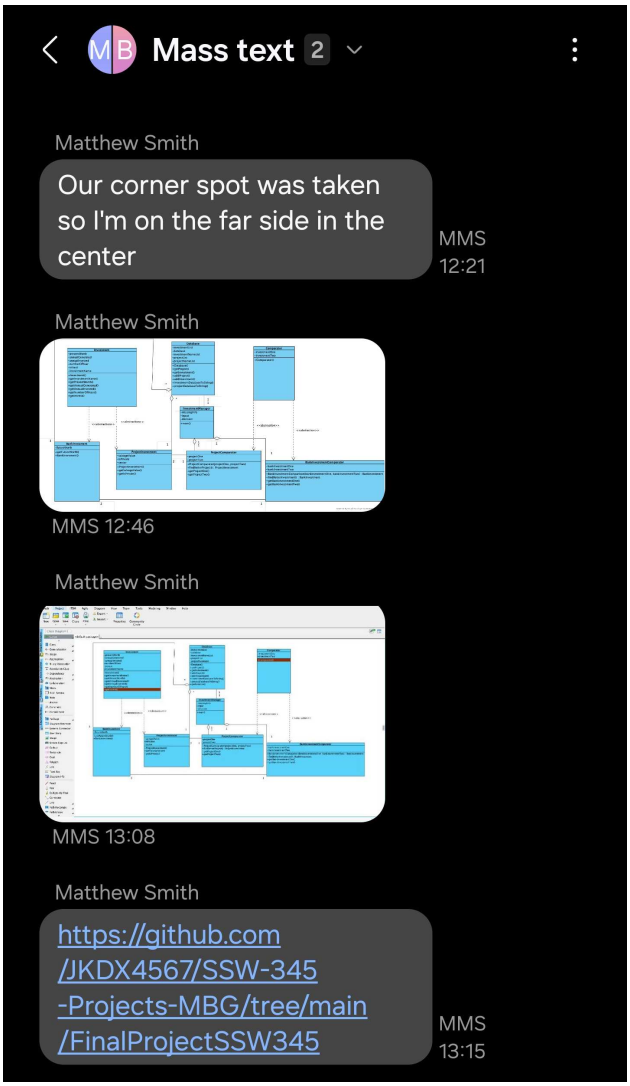

Figure 3.1: SMS Screenshot 1

Figure 2.1 is a screenshot of us talking about meeting up in the Library and reviewing our Diagrams with a link to our GitHub.

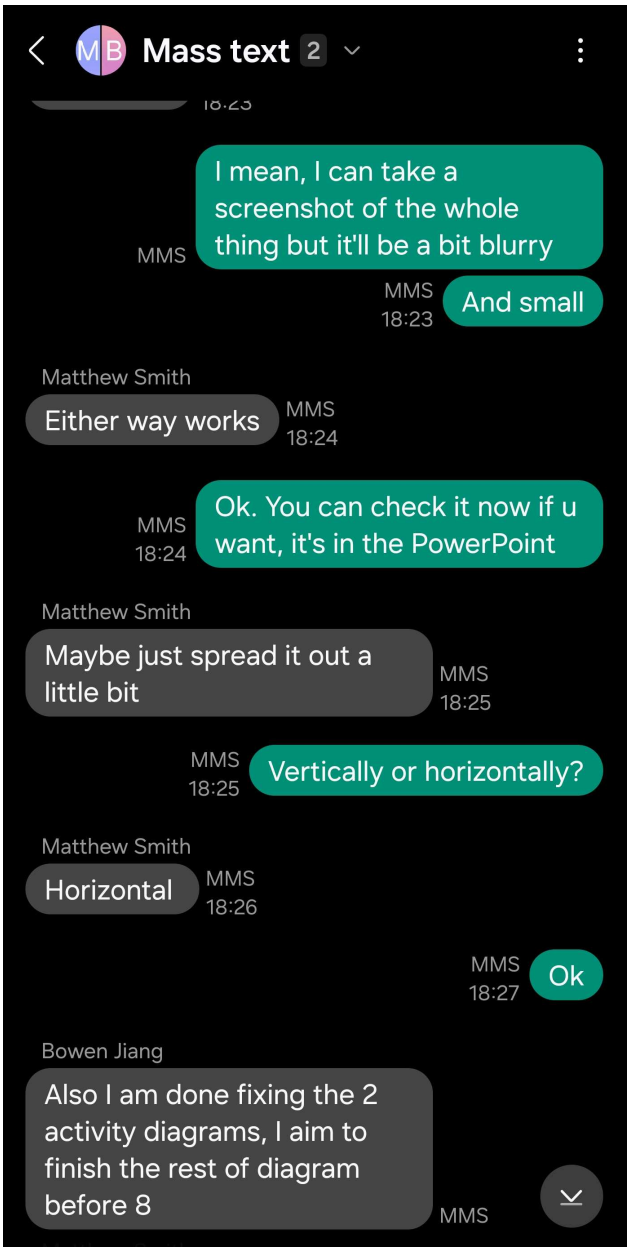

Figure 3.2: SMS Screenshot 2

Figure 2.2 is a screenshot of us talking about the PowerPoint that we were working on for our presentation and our progress on some of the diagrams.

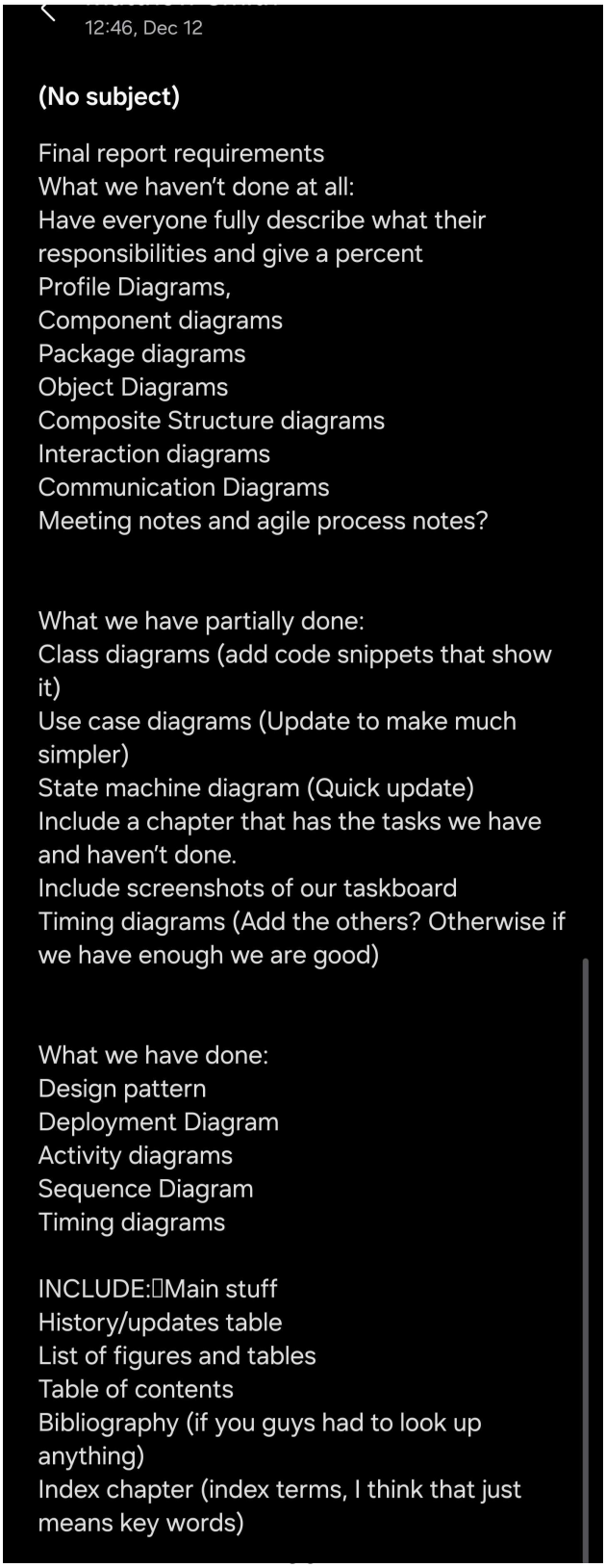

Figure 3.3: SMS Screenshot 3

Figure 2.3 is a screenshot of the progress reports of what we have completed and what we need to work on. These are a few screenshots of the communication we have between the teammates of the project group. We also have some documented notes of what we have accomplished below.

Meeting Notes for Final Project Report: 12/3/2024: Discuss the solution on how to fix the architecture diagram and sequence diagram, also figuring adjust and create classes for the class diagram.

12/4/2024: Finalized all the class necessary and started working on creating rest of the required diagrams and codes for the classes to make the app work at the minium level.

12/5/2024: Separate the task between teammates on coding and milestone Two report.

12/7/2024: Report on the statistics on the process of the codes and the milestone report.

12/10/2024: Completed presentation

12/12/2024: Delegated missing tasks for final report, completed Interaction diagram, Object diagram, and Communication diagram.

12/17/2024: Completed most of the diagrams and text for each diagram. A few more need to be done before handing in the full report. Some chapters have been added/modified and the organization of the document has been moved around.

12/18/2024: Adding all the missing diagrams and ready to wrap up everything and get ready to submit.

Chapter 4

Task and Progress Management

Below we have the ways that we kept track of our progress and the way we split our work between each other.

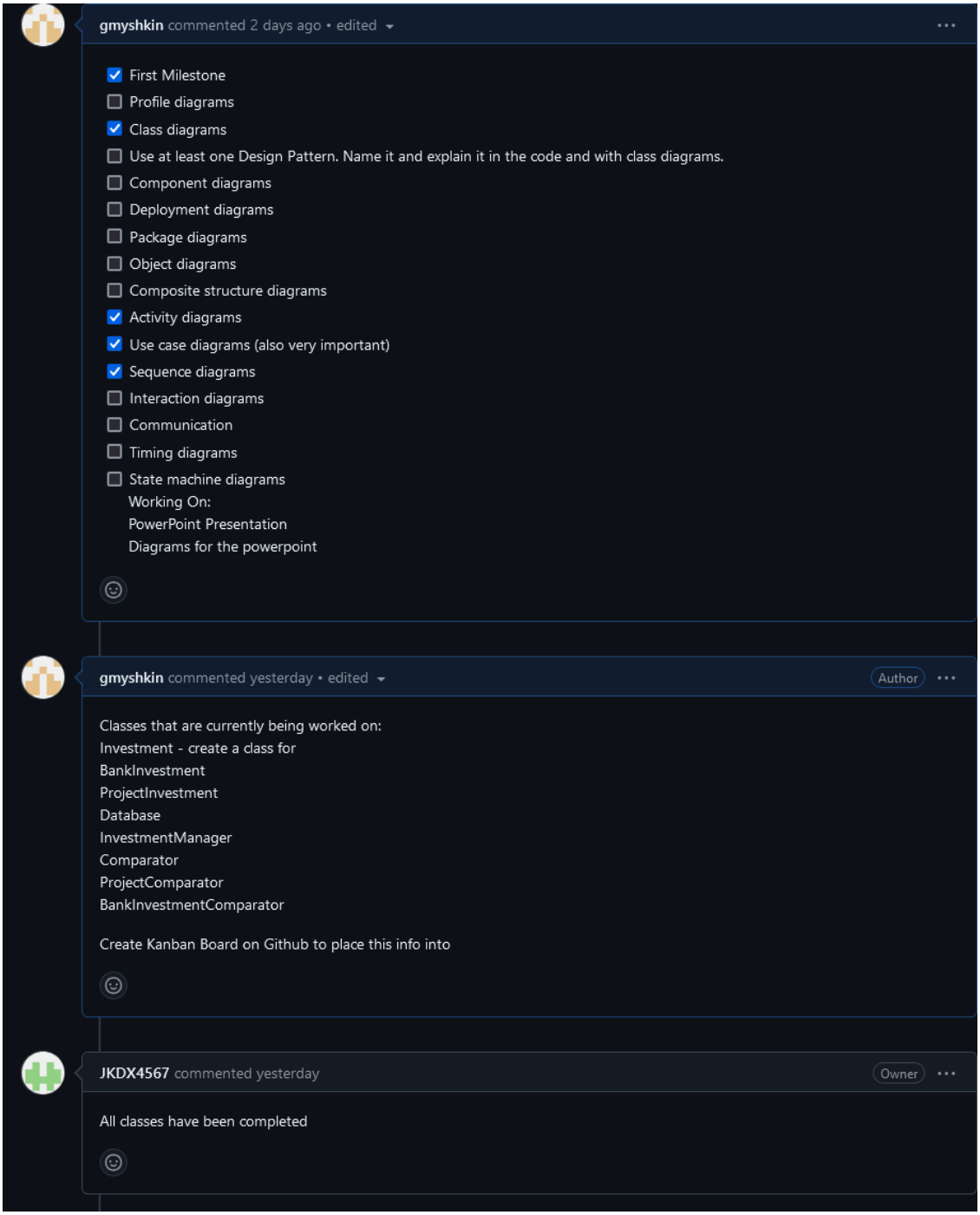

Figure 4.1: Final Project History Tracking

When we first started to track our progress, we created an issue in GitHub and created tasks (with an option to check mark them) in the issue (Figure 3.1). We also added any new comments with stuff that we have to add/modify and whenever we finished a task, we put that down as a comment too. We realized that this wasn't the best way to keep track of our progress so we decided to move onto a Kanban that was created in GitHub.

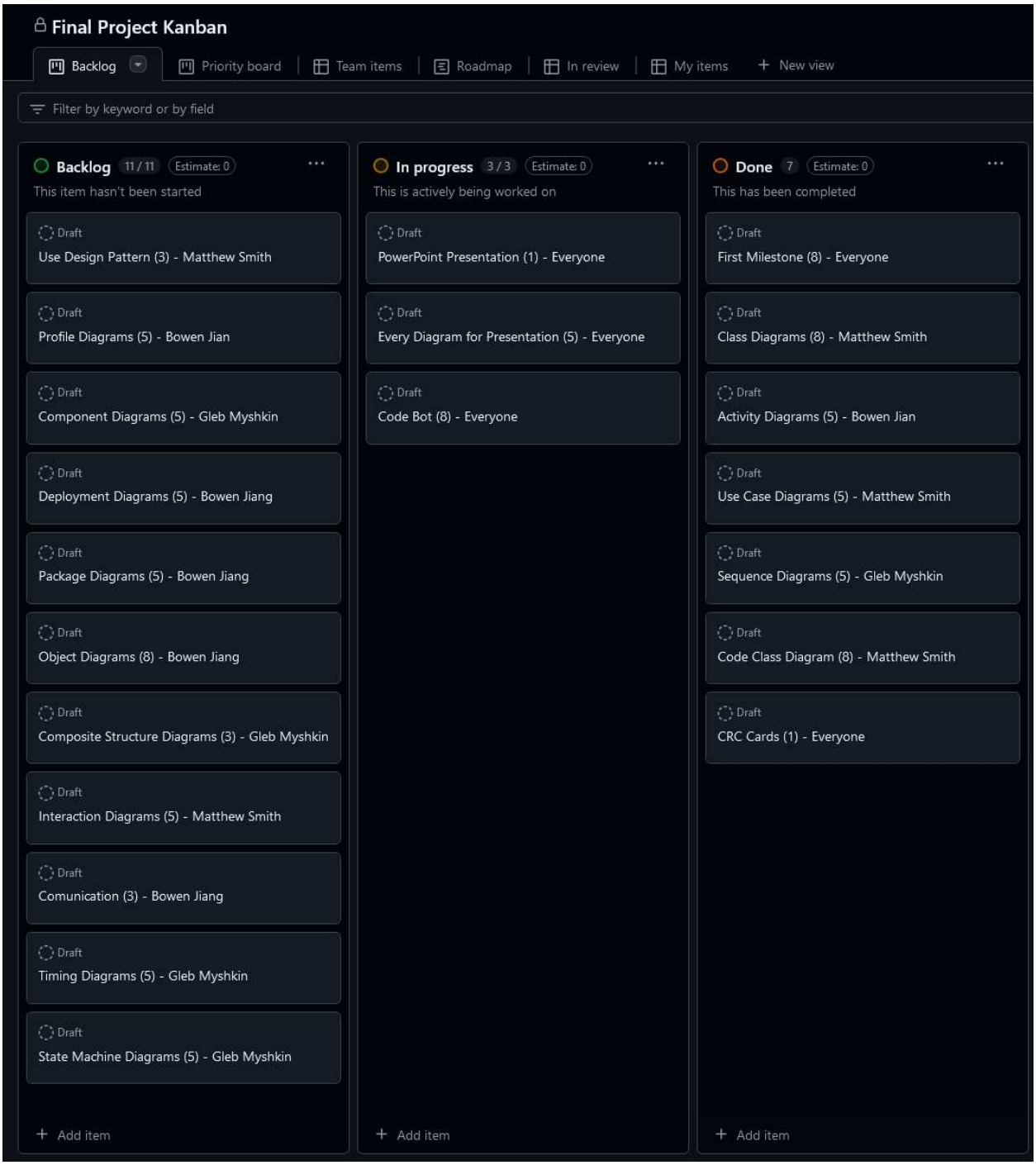

Figure 4.2: Final Project Kanban

Figure 3.2 is the Kanban that we created. It has three columns, Backlog (stuff we have to work on), In Progress (stuff we are currently working on), and Done (stuff we have finished). This makes

it very simple to add/modify a task we are working on so that we know what we have left. We also added numbers next to the name of the tasks to indicate their complexities. We have the numbers: 1, 3, 5, and 8, indicating from easiest to hardest respectively. We also added names to the tasks to indicate who's working on who. There are also some tasks that have "Everyone" in the name to indicate that everyone took (or will take) a part in completing/working on.

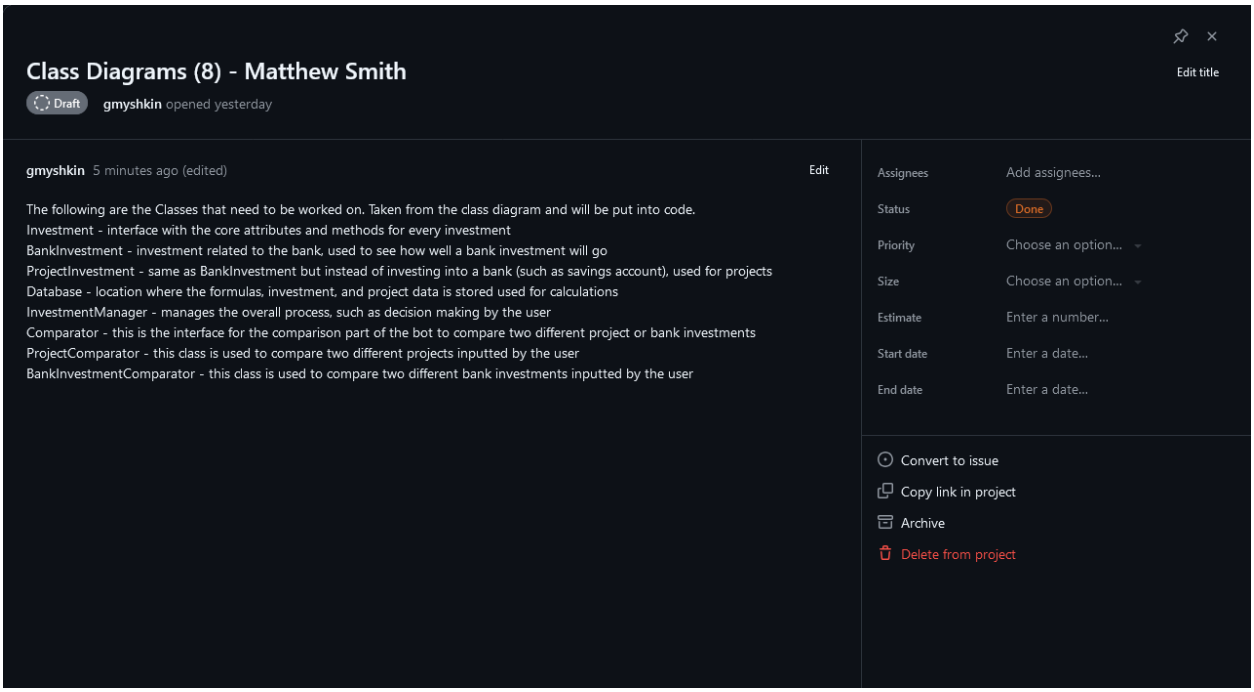

Figure 4.3: Descriptions for Classes from Class Diagram

Figure 3.3 shows an example of the descriptions of one of the tasks that we assigned. This is the example of the Classes that we were working on, and finished. We were not able figure out how to show the descriptions of each task in the main backlog screen, but pressing each individual task allows us to add/edit the description of the task.

In the end, we decided to keep our task screenshot as they were because we decided it to be best to show how our task management process was instead of just showing a kanban board that has every single task on the Done side, with no real value to our final report.

Process Reflection:

We have finished our project and were able to add every diagram that was asked of us and every chapter that we needed to add. We also have a working bot that is able to do a variety of calculations when a user starts it up. We added some meeting notes that we had and when they were created. Some exact dates were not listed as they were not major points in our project creation. We have a list of tables and figures and made sure that our index has all of the parts of the final project report that we added.

Bibliography

[1] (2002) Visual paradiam. [Online]. Available: https://www.visual-paradigm.com/

[2] (2008) Github. [Online]. Available: https://github.com/

[3] (2015) Visual studio code. [Online]. Available: https://code.visualstudio.com/

[1] [2] [3]

Index