Developer Manual

Paradiso Application

June 2019

Team Members

Team member	Email Address	Responsibilities
Marzieh AshrafiAmiri	mashrafi@uci.edu	Manager - Senior Software Engineer
Amir Hosein Afandizadeh Zargari	aafandiz@uci.edu	Recorder - Algorithm Developer, Software Engineer
Amir Hossein Aqajari	amiraj.95@uci.edu	Reflector - Quality Assurance Engineer
Rozhin Yasaei	ryasaei@uci.edu	Presenter

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	04/11/2019	Developer Manual
Rev. 1	04/25/2019	Developer Manual – Alpha Release (Add hierarchy, change in the figures of installation and packages used in python, update Gantt chart of the project and error messages)
Rev. 2	05/09/2019	Developer Manual – Beta Release (Fix alpha release issues(change coordinates and add scalability feature to GUI), update software components and connections of functions, add algorithms section, modify packages used in python and our timeline, add error messages)
Rev. 3	05/24/2019	Developer Manual – Beta Release V2. (Fix alpha release issues, update software components and connections of functions, add extra algorithms, modify our timeline, add error messages and extra explanation for formula used in our proposed algorithm in previous version)
Rev. 4	06/06/2019	Developer Manual – Final Release (Fix Beta release issues (fix branch and bound algorithm, add previous step button, show direction in GUI, add secure sign in), change software components and algorithms in finding the fastest route, delete installation, update error massages, plan and timeline)

Table of Contents

Table of Figures

Table of Tables

1. Hierarchy

First let's talk about the hierarchy of the Paradiso software.
The hierarchy is as below:

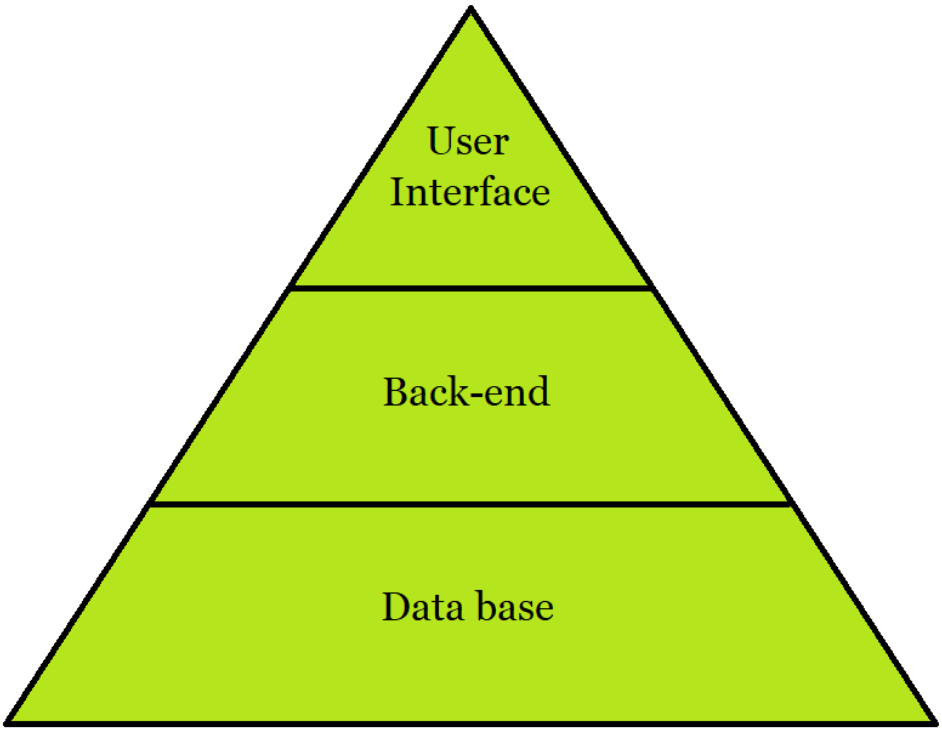

Figure 1 Hierarchy of project

On the top of the hierarchy, there is UI. User uses the user interface to input the requirements such as product number, start point, and end point to the system. Then after we need some computation on these inputs to calculate the required outputs.
These inputs insert to the Back-end which is the python codes and functions, to calculate the shortest path using the Data base.
After that the outputs will be inserted back to the User Interface to be displayed to the user. For example, the path from the start node to the product and from the product to the end node will be displayed on the software.

2. Software Architecture Overview

Paradiso is a developed application written in Python Programming Language. In this part, we will look deep into the process of producing Paradiso and how it is going to work.

2.1. Data Type and Structure

The main data structure used in producing Paradiso is the matrix. These are used for storing input, distance calculation, and other stuff. A more detailed description of the data structure can be found in section 4.2.

2.2. Software Components

In the following parts, you will understand what are the main component and functions in producing Paradiso.

2.2.1. Main

Table 1 Input and Output for Main Function

Input	Output
Void	Void

Functionality: This is the main function which acts as a core of our application. All other functions are called through Main and without main, others couldn't perform their functionality.

2.2.2. Read_Input

Table 2 Input and Output for Read_Input Function

Input	Output
Text file contains information of the warehouse	Matrix of N rows and 7 columns named "InputMat"

Functionality: This function transform the warehouse information stored in the excel file to a matrix of N*7. N is the number of products.

2.2.3. Generate_Dist_Mat

Table 3 Input and Output for Generate_Dist_Mat Function

Input	Output
Location of shelves and the worker	A matrix named "DistMat"

Functionality: If you consider having warehouse of size M*N, this function will calculate the distance between each two nodes (except location of shelves). The important point is that distance between an object to itself is zero, so, the output will be a matrix with diagonal equal to zero. This function is just called once and data will be stored in a .txt file near main program.

2.2.4. Exact_Shelf

Table 4 Input and Output for Exact_Shelf Function

Input	Output
"InputMat" and location of shelves	A matrix named "LocMat"

Functionality: This function calculates the exact shelf of each product by using the InputMat matrix and the location of each shelf. In other words, it will find a shelf for each product which contains that product. It will return a matrix of N*3 as an output. Each row represents an item, the first column is product ID and the second column is the shelf number. The third column is a four-digit binary number which shows how we can access the item, from North, East, South, and West.

2.2.5. Read_File

Table 5 Input and Output for Read_File Function

Input	Output
Input file including number of orders	Two matrix named “data_list” and “Input_Flag”

Functionality: This function will read the input file containing orders and store them in a matrix named “data_list”. Also, it will keep track of continuing orders starting from the first order. The matrix “Input_Flag” shows if a specific order is being processed or not.

2.2.6. DataBase

Table 6 Input and Output for DataBase Function

Input	Output
Text files used for orders and place of shelves	Storing all needed information for other functions

Functionality: This function contains all useful information needed for other functions.

2.2.7. Initialize_Map

Table 7 Input and Output for Initialize_Map Function

Input	Output
DataBase and location of worker (start point)	A table in the terminal which shows a map of the warehouse

Functionality: This function will print out a GUI map of the warehouse consisting of the locations of shelves, worker and all vacant spots by reading the information of DataBase and using the help of PyQt package.

2.2.8. Main_menu

Table 8 Input and Output for Main_menu Function

Input	Output
DataBase	A GUI for the beginning of application

Functionality: This function will show a GUI for logging in, signing-in and starting the application.

2.2.9. Enter_Start

Table 9 Input and Output for Enter_Start Function

Input	Output
DataBase	Showing the start point on GUI

Functionality: This function will show the start point by light blue and “S” sign on the GUI.

2.2.10. Enter_End

Table 10 Input and Output for Enter_End Function

Input	Output
DataBase	Showing the end point on GUI

Functionality: This function will show the end point by light blue and “E” sign on the GUI.

2.2.11. Search_ID

Table 11 Input and Output for Search_ID Function

Input	Output
DataBase	Location of Item

Functionality: This function find the accurate location of a specific item or list of items.

2.2.12. Branch_Bound

Table 12 Input and Output for Branch_Bound Function

Input	Output
DataBase	Optimal path and its cost

Functionality: This function is an implementation of branch and bound algorithm that will find the shortest path between start point, orders and end point.

2.2.13. Nearest_neighbor

Table 13 Input and Output for Nearest_neighbor Function

Input	Output
DataBase	Greedy path and its cost

Functionality: This function is an implementation of the greedy algorithm “nearest neighbor” that will find a greedy path between start point, orders and end point.

2.2.14. ui

Table 14 Input and Output for ui Function

Input	Output
All information of DataBase and start and point and fastest routs	A user-friendly GUI

Functionality: This function will show a user_friendly GUI. You can easily set orders, start and end points. Then by using two buttons you can view the rout step by step.

2.2.15. Print_Direction

Table 15 Input and Output for Print_Direction Function

Input	Output
The shortest path to items and location of the worker (start and end points)	Some lines in GUI showing the directions

Functionality: This function will print user-friendly instructions for the worker that explains the direction he/she should pass to pick up the item(s). The output will be like the figure below. This function is used in Show_Path file.

```
1. Go 1 steps east from (2,3) to (3,3)
2. Go 8 steps north from (3,3) to (3,11)
3. Go 7 steps east from (3,11) to (10,11)
```

Figure 2 Output Directions

2.3. Algorithms for Finding the Fastest Route

2.3.1. Brute Force Algorithm (used in Alpha Release)

The first algorithm is the original brute force algorithm in which we iterate over all the possible permutations of the list of the products exclude of start and end points to find the best route possible. Then we add start node and end node and find the best route.

The problem of using brute force is that it won't terminate for list of products with more than 10 different locations in the reasonable time (one minute).

2.3.2. Modified Brute Force Algorithm (in Beta Release version1)

The problem of brute force algorithm inspired us to modify this algorithm to get better results in larger number of products.

This second algorithm is called modified brute force algorithm in which we iterate over all the possible permutations of the list of the products exclude of start and end nodes, but the difference here is that when we reach to the specific proportion of the accuracy of the efficient path, that has been computed according to the length of the efficient route in the MST of the graph using Prime algorithm, we terminate the program and output the path. This specific proportion is calculated based on the number of items that user entered. The equation (1) shows this formula.

$$distance \leq \left(1.05 + \frac{number\ of\ items}{20}\right) * length_{MST} \qquad (1)$$

2.3.3. Brach and Bound Algorithm (used in Beta Release version2 and Final Release)

This algorithm uses the distance matrix and reduction method on this matrix. The cost function is defined in this algorithm and in each step we visit a node with the minimum cost function. The tree will be made eventually.

If in our tree we visit the first node, that branch will be terminated. Now, if it is the lowest cost and also we visit all nodes in the branch, the final answer of algorithm is ready.

2.3.4. Nearest Neighbor Algorithm (used in Final Release)

This greedy algorithm uses the distance matrix and will find a fast but not optimal route. In each step of this algorithm, it will visit the nearest location in the list. Then, among different paths, it will choose the best one.

2.3.5. Combined Algorithm Used in Final Release

This algorithm is used as our algorithm for Paradiso Final Release. First, we will run the branch and bound. Because our goal is having a fast path in one minute, if there is no result for "branch and bound" in 55 seconds, we will use nearest neighbor. Nearest neighbor is so fast and will give us the result in less than a millisecond.

For more information on timing and memory usage, you can see our test document.

2.4. Program Control Flow

As you can see from the chart below, there are some initial steps for using Paradiso, opening the application, signing up and then signing in. If all these three steps are done, users can use the application menu in order to search the location of the product. The user can see the route to the product that he/she chose unless the product is in the warehouse. For more information, you can read the user manual of Paradiso application.

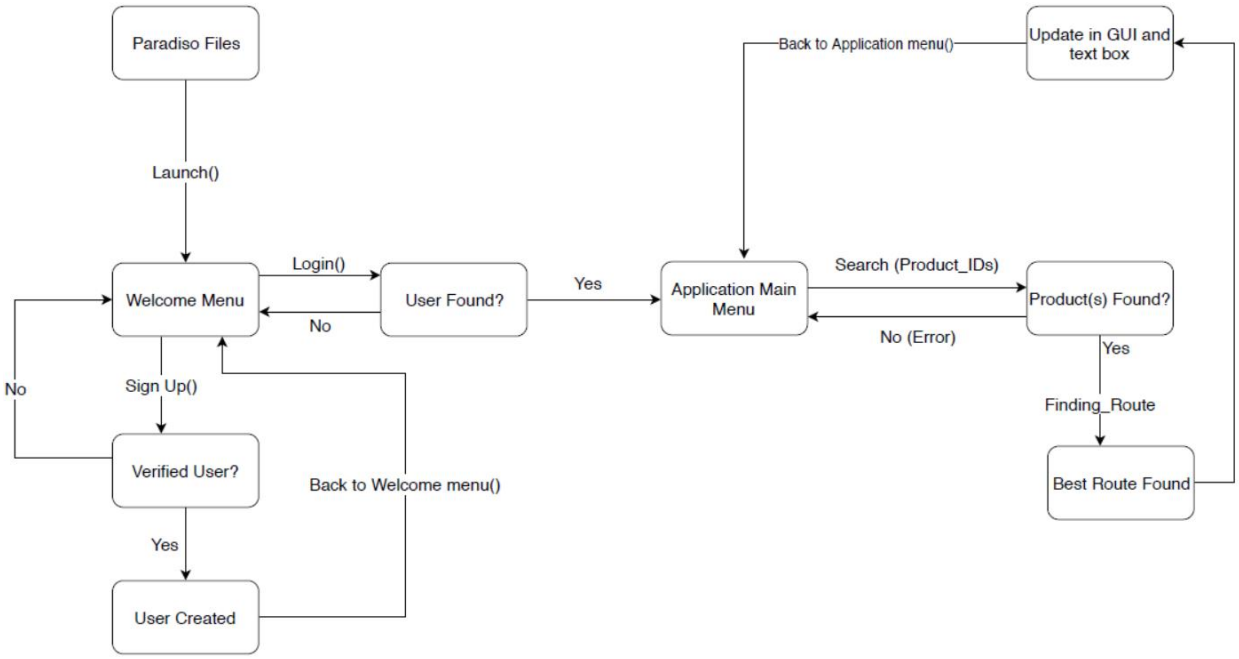

Figure 3 Program Control Flow

3. Documentation of Packages and modules

3.1. Description of Packages

For creating Paradiso, there are two important Python packages that we use.

- PyQt is one of the most popular Python bindings for Qt cross-platform C++ framework. We used PyQt as our UI layer for input buttons, text fields, etc.
- Numpy is a library for Python programming language that supports operations on large multi-dimensional arrays and matrices. We stored the input in a matrix so lots of matrix processing is needed which are supported by this package.
- Permutations is a standard python library that standardizes a core set of fast, memory efficient tools that are useful by themselves or in combination. We use this to calculate different permutations of routs.
- Collections is a package in python which includes specialized containers for data types. We use this package in our BFS algorithm which is implemented by queue.

3.2. Description of data structure

We read the input file (an excel file) and store them into a matrix. Each row in this matrix named, InputMat, is representative of a product. There exist seven columns. The first three ones are a description of the product, Product ID, X location and Y location.

The four remaining columns show how we can access shelves, from North, East, South, and West. “1” is a symbol for showing the possibility of access and “0” translates to a closed end for the shelf. Later we save these four columns in a four-digit binary number ranging from 0 to 15.

All the smallest path between two shelves are then calculated and stored in another matrix, named DistMat. We then use this matrix in finding the shortest path for picking up a product.

3.3. Connection of Functions

The functions that are used in developing Paradiso are mentioned in section 2.2. The flow chart below explains the connection between them.

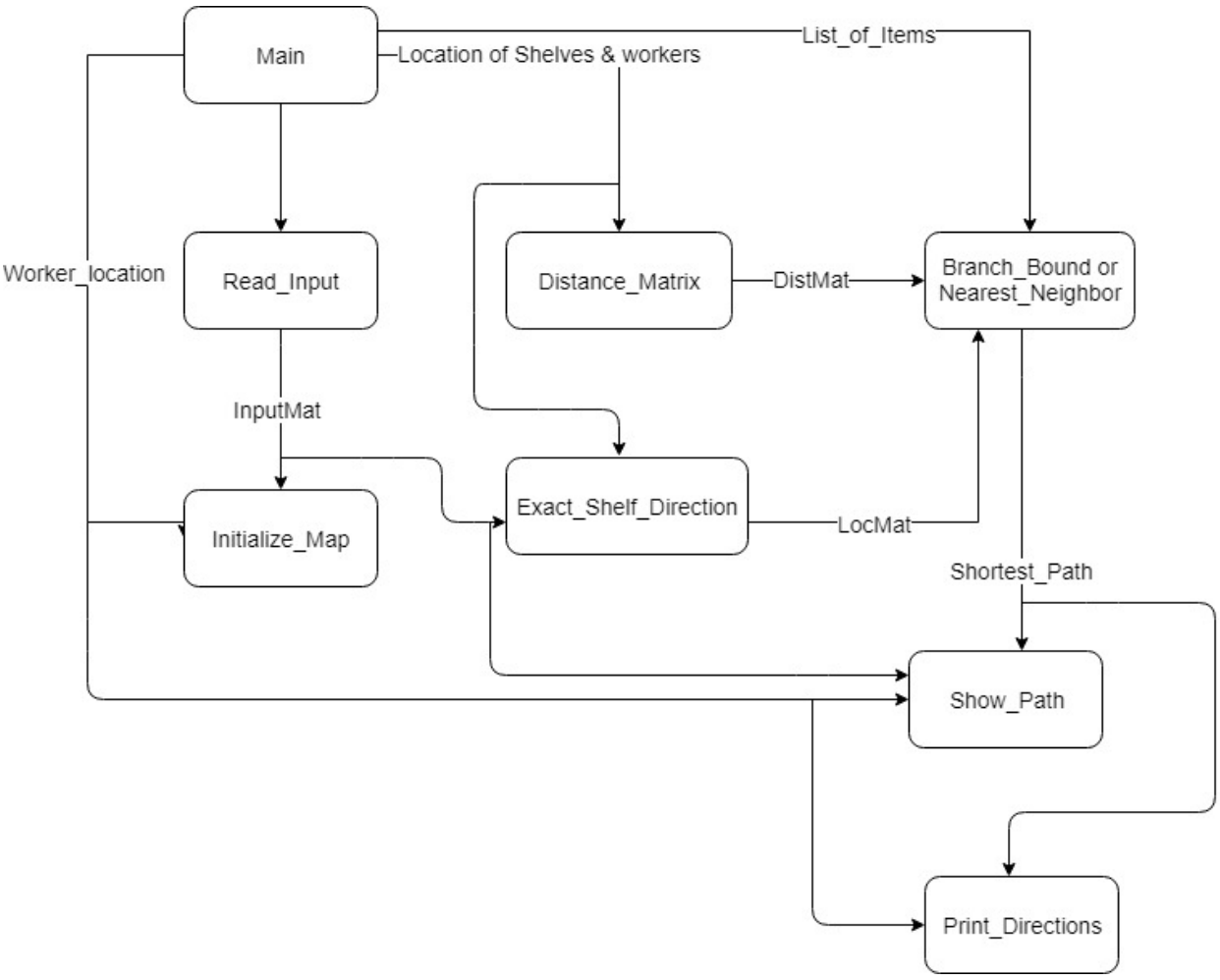

Figure 4 Hierarchy of Functions

4. Plan and Timeline

4.1. Partitioning of Tasks

All tasks are determined and listed in the figure11.

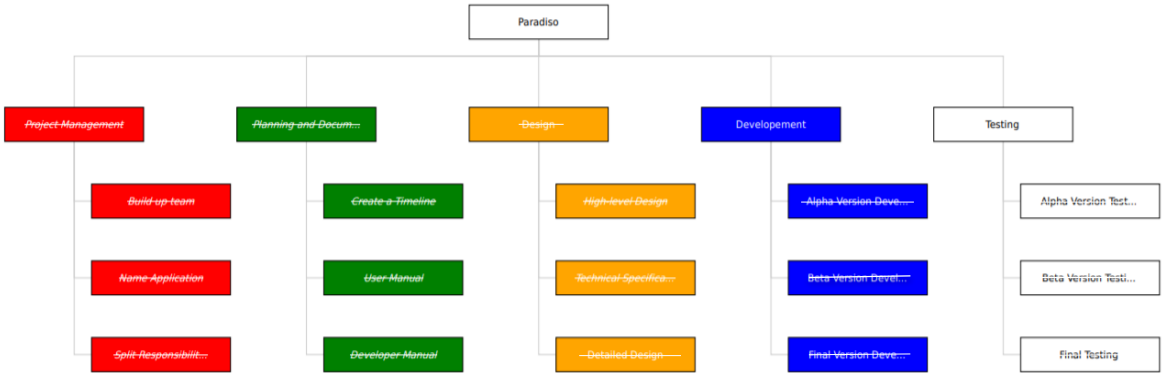

Figure 5 Partitioning of Tasks

4.2. Team Member Responsibility

The general responsibilities of team members are specified in the table9.

Table 16 Team Member Responsibilities

Team member	Detailed Responsibility
Marzieh AshrafiAmiri	Detailed design of Alpha and Beta Versions, Documentation of the final version
Rozhin Yasaei	Documentation of alpha and beta versions
Amir Hosein Afandizadeh Zargari	Design of GUI, Development of different algorithm used for routing in final version of the application
Amir Hossein Aqajari	Advanced design of secure sign-in, Test documentation of Alpha, Beta and final versions

4.3. Planning and Timeline

All tasks are scheduled and assigned in table10 and the timeline of the project is created as a Gantt chart in figure12.

Table 17 List of Tasks

Task Number	Task Title	Start Date	End Date	Complete%	Duration (days)
P-0	**Project Management**	4/3/2019	4/5/2019	100	2
P-0.1	Build up team	4/3/2019	4/4/2019	100	1
P-0.2	Name Application	4/4/2019	4/5/2019	100	1
P-0.3	Split Responsibilities	4/4/2019	4/5/2019	100	1
P-1	**Planning and Documentation**	4/5/2019	4/19/2019	100	14
P-1.1	Create a Timeline	4/5/2019	4/6/2019	100	1
P-1.2	User Manual	4/5/2019	4/13/2019	100	8
P-1.3	Developer Manual	4/12/2019	4/19/2019	100	7
P-2	**Design**	4/9/2019	4/23/2019	100	14
P-2.1	High-level Design	4/9/2019	4/13/2019	100	4
P-2.2	Technical Specification	4/13/2019	4/18/2019	100	5
P-2.3	Detailed Design	4/18/2019	4/23/2019	100	5
P-3	**Development**	4/19/2019	6/5/2019	100	47
P-3.1	Alpha Version Development	4/19/2019	4/24/2019	100	5
P-3.2	Beta Version1 Development	4/26/2019	5/8/2019	100	12
P-3.3	Beta Version2 Development	5/10/2019	5/22/2019	100	12
P-3.4	Final Version Development	5/24/2019	6/5/2019	100	12
P-4	**Testing**	4/25/2019	6/7/2019	100	43
P-4.1	Alpha Version Testing	4/24/2019	4/26/2019	100	2
P-4.2	Beta Version1 Testing	5/8/2019	5/10/2019	100	2
P-4.3	Beta Version2 Testing	5/22/2019	5/24/2019	100	2
P-4.4	Final Testing	6/5/2019	6/7/2019	100	2

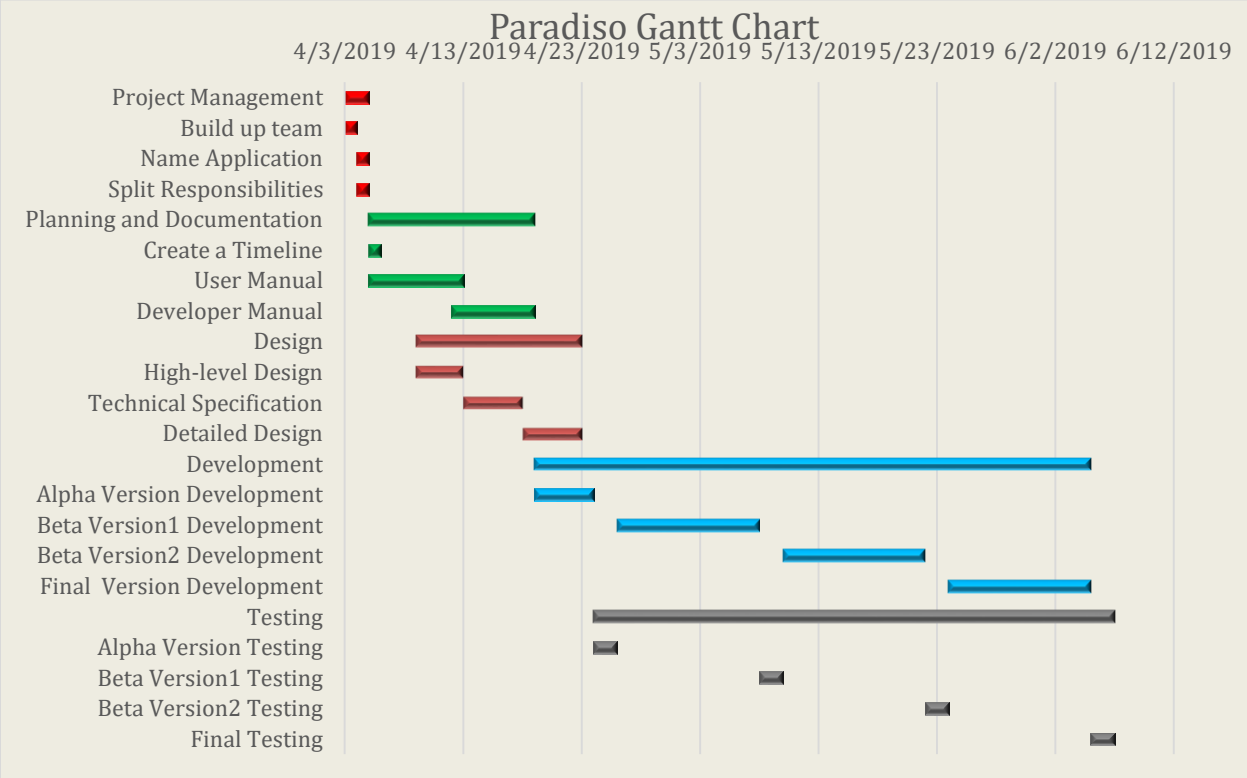

Figure 6 Gantt Chart of Project

5. Back Matter

5.1. Copyright

5.2. Error Messages

Table 18 Error Messages

No.	Error Message	Description
1	Wrong username or password!	You have entered the wrong username or password to Sign In.
2	Username already exists!	You have entered the username to sign up which is already in data base.
3	Password fields are not match!	Password fields for sign up form are not match!
4	Please complete all the fields!	You have not entered all the information to sign up!
5	You are not Signed In!	You clicked on sign out while you are not even signed in.
6	You are not Signed In! Use Sign In option to Sign In first.	You clicked on run the application before signing in.
7	You are already Signed In, use start the application button to run the application.	You clicked on sign in option after signing in to the application.
8	The product you enter is not valid. Please enter a valid product Id ...	The requested item does not exist in the warehouse database.
9	Product ID is not even a number ...	The entered product ID is not a number
10	Sorry, the product is not accessible now	There is not any valid route to the requested item.

11	The point you entered is out of range Please enter a valid location …	The entered value for point coordinates is outside of map.
12	Please fill all the locations or product ID	Some input information is missing which can be start point coordinates, end point coordinates, or product ID.
13	The point you entered is located on shelves Please enter a valid location …	The entered start/end point is located on shelves.
14	Your location is not even a number …	The entered start/end point may be negative number or may contain letters or symbols.
15	Please select one method to place your order …	Before clicking on Add item, you should choose one of the three methods.
16	Your order number is out of range	The number of order entered is not a valid order number.
17	This order already picked	The order number user entered has been picked already.

6. Extra – Modified Brute Force

In section 2.3.2, we proposed an algorithm of modified brute force using a formula. We came up with this formula in testing phase. At first, we wanted to use a fixed proportion of MST (1.5 * MST) for a condition of terminating our algorithm.

However, in testing we figure out that it is better to have a linear formula regarding to the number of inputs. We used the test cases below to find out the coefficient needed for the linear formula. In each test case, time used for finding the path was important for us. We wanted to limit this amount of time by 10 Sec. Therefore, we tried to find formula to achieve this. You can find the test cases and their running time below. Since, inputs smaller than 10 were fine with the actual brute force; we focus on the input sizes from 10 to 15.

6.1. Test case size 15

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592, 107055, 108001, 36274, 910143, 914087
MST: 99
1.5 * MST: $\begin{cases} Path\ length: 147 \\ Time: 19.406\ Sec \end{cases}$

$(1 + \frac{15}{20})$ * MST: $\begin{cases} Path\ length: 173 \\ Time: 0.0839\ Sec \end{cases}$

$(1.05 + \frac{15}{20})$ * MST: $\begin{cases} Path\ length: 177 \\ Time: 0.0039\ Sec \end{cases}$

6.2. Test case size 14

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592, 107055, 108001, 36274, 910143
MST: 97
1.5 * MST: $\begin{cases} Path\ length: 145 \\ Time: 1.98\ Sec \end{cases}$

$(1 + \frac{14}{20})$ * MST: $\begin{cases} Path\ length: 163 \\ Time: 0.108\ Sec \end{cases}$

$(1.05 + \frac{14}{20})$ * MST: $\begin{cases} Path\ length: 169 \\ Time: 0.019\ Sec \end{cases}$

6.3. Test case size 13

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592, 107055, 108001, 36274
MST: 97
1.5 * MST: $\begin{cases} Path\ length: 145 \\ Time: 31.59\ Sec \end{cases}$

$(1 + \frac{13}{20})$ * MST: $\begin{cases} Path\ length: 159 \\ Time: 0.0559\ Sec \end{cases}$

$(1.05 + \frac{13}{20})$ * MST: $\begin{cases} Path\ length: 163 \\ Time: 0.024\ Sec \end{cases}$

6.4. Test case size 12

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592, 107055, 108001
MST: 97
1.5 * MST: $\begin{cases} Path\ length: 145 \\ Time: 0.036\ Sec \end{cases}$

$(1 + \frac{12}{20})$ * MST: $\begin{cases} Path\ length: 151 \\ Time: 0.008\ Sec \end{cases}$

$(1.05 + \frac{12}{20})$ * MST: $\begin{cases} Path\ length: 157 \\ Time: 0.016\ Sec \end{cases}$

6.5. Test case size 11

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592, 107055
MST: 97
1.5 * MST: $\begin{cases} Path\ length: 145 \\ Time: 0.007\ Sec \end{cases}$

$(1 + \frac{11}{20})$ * MST: $\begin{cases} Path\ length: 145 \\ Time: 0.003\ Sec \end{cases}$

$(1.05 + \frac{11}{20})$ * MST: $\begin{cases} Path\ length: 151 \\ Time: 0.008\ Sec \end{cases}$

6.6. Test case size 10

Input: 365671, 366067, 1085968, 1, 1253, 6600, 6717, 366185, 33713, 55592
MST: 97
1.5 * MST: $\begin{cases} Path\ length: 133 \\ Time: 0.004\ Sec \end{cases}$

$(1 + \frac{10}{20})$ * MST: $\begin{cases} Path\ length: 133 \\ Time: 0.004\ Sec \end{cases}$

$(1.05 + \frac{10}{20})$ * MST: $\begin{cases} Path\ length: 149 \\ Time: 0.004\ Sec \end{cases}$