DEPARTMENT OF COMPUTER SCIENCE
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI – 600036

VeShare:Ride Sharing and Cargo Planning Test Beat

A Final MTP Report

Submitted by

ANKIT KHARWAR

For the award of the degree

Of

MASTER OF TECHNOLOGY

May 2023

THESIS CERTIFICATE

This is to undertake that the Final MTP Report titled **VESHARE:RIDE SHARING AND CARGO PLANNING TEST BEAT**, submitted by me to the Indian Institute of Technology Madras, for the award of **Master of Technology**, is a bona fide record of the research work done by me under the supervision of **Prof. N.S. Narayanaswamy**. The contents of this Final MTP Report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Chennai 600036 **Ankit kharwar**

Date: May 2023

Prof. N.S. Narayanswamy
MTP Guide
Professor
Department of Computer Science
IIT Madras

ACKNOWLEDGEMENTS

I would want to express my sincere gratitude to N.S. Narayanaswamy for providing me with this fantastic chance and invaluable support while I complete my project, "VeShare And Cargo Planner App." In this regard, I shall always be grateful to you.

ABSTRACT

As we know for rapid economic development for any country as well as for the growth of 'Ease of Living' transportation plays a very important role. It is a fundamental factor in facilitating the movement of goods and people, which is essential for trade, commerce, and tourism.

So two major things are now major concerns in the developing country like India one is the transportation of a person from one place to another and the second is efficient packing and moving of cargo from one point to another point, Which will lead to better access to markets, health care, education, and other essential services, ultimately enhancing the overall quality of life for citizens.So by the help of this we can say Connected, clean and shared transportation solutions are widely becoming the key principles of effective transportation across the globe.

As a result, we attempt to address these two issues in this project by developing some of the most efficient algorithms. One algorithm will go to solve the ride-sharing problem, which will help in finding the shortest path from one place to another place and assigning drivers to the customer for ride-sharing. Another algorithm will solve the cargo packing problem, which will use to help in effectively uploading cargo in the containers and also unloading cargo from them. The safe delivery of cargo is guaranteed by these best practices.

Using this efficient algorithm for ride-sharing and cargo planning at the server end. We are developing a fully working ride-sharing and cargo planning app (Using flutter with flutter 2.8 null safety) (2). This ride-sharing app will solve the problems that people face during transportation like more traffic, hike in transportation charges, and lack of fuel.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Ride sharing is a sustainable transportation option that has gained popularity in recent years due to its benefits of reducing travel costs, fuel consumption, and increasing passenger vehicle occupancy. Ride sharing involves multiple passengers sharing a ride in a vehicle, which helps reduce the number of vehicles on the road, thereby reducing traffic congestion and air pollution.

In addition to the economic and environmental benefits, ride sharing can also lead to social benefits by fostering a sense of community and creating opportunities for social interaction. Furthermore, it can provide a safe and convenient transportation option, particularly for those who do not own a vehicle or cannot afford to drive.

Overall, ride sharing is an effective way to promote sustainable transportation and reduce the negative impacts of transportation on the environment, while also benefiting individuals and communities economically and socially.

Efficient cargo loading and unloading software can be a valuable tool for businesses and individuals involved in transportation and logistics. This software can help to optimize the loading and packing process, ensuring that cargo is packed in the most efficient way possible to minimize wasted space and reduce the risk of damage during transit.

By automating and streamlining the process of cargo loading and unloading, this type of software can also help to save time and reduce labor costs. It can help to eliminate errors and ensure that the right cargo is loaded onto the right vehicle, reducing the risk of delays or lost shipments.

Furthermore, by ensuring that cargo is loaded safely and securely, cargo loading and

unloading software can help to reduce the risk of accidents and injuries, protecting both workers and the cargo being transported.

Overall, efficient cargo loading and unloading software can be an important tool for improving the efficiency and safety of transportation and logistics operations, ultimately helping businesses and individuals save time and money while ensuring that cargo is transported safely and securely.

Ride-sharing and efficient cargo loading and unloading software are both important tools that can help to reduce transportation costs and increase sustainability. By promoting the sharing of resources and optimizing transportation operations, these tools offer practical solutions that can benefit both businesses and individuals.

Ride-sharing, for example, can help to reduce the number of vehicles on the road, thereby reducing traffic congestion and air pollution, while also making transportation more affordable for passengers. Efficient cargo loading and unloading software can help to reduce waste and optimize the use of space, thereby reducing transportation costs and minimizing the environmental impact of cargo transportation.

Overall, the use of these tools can contribute to a more sustainable future by reducing the negative impacts of transportation on the environment while also providing economic benefits for businesses and individuals. As such, they represent important steps forward in the effort to create a more sustainable transportation system for the future.

So in this project, we are developing an android and IOS ride-sharing and cargo planner app using flutter with flutter 2.8 null safety(2). This flutter app will work with any Website, Mobile, Tablet or any dimension screen perfectly. This project consists of two apps one is a user app and the other one is a driver app. This app takes input from the person who uses this app for drivers it takes all the details of the driver and the vehicle that the driver will use for the ride and stores it in the backend database with an

easy-to-use and user-friendly User Interface. There are lots of features for drivers which attract drivers towards apps and make them flexible, reliable and comfortable to use for any drivers.

Similarly, the user-side app had some good features which make it user-friendly. The user interface makes users more flexible with the app and easy to use. The app takes user information and stores it in a database. Then accordingly when the user needs do a ride request selecting one of the good fit options got from the allocations of online drivers using the ride-sharing algorithm on the server end or the cargo transportation request they performed, it will notify the driver app through the back end Applications interface calls(APIs). Now the driver gets the option to accept the ride or reject it according to the driver's comfort. After accepting the request the ride schedule and with the help of the server end algorithm and using an open source routing machine the shortest path is calculated and it is shown to the user and driver app with a polyline between source and destinations which helps drivers to see the directions. For both the user and driver app there is live tracking of the vehicle movement is shown with high User interface quality .

Users can request cargo transport through this app which transports at a safer, more reliable and affordable charge. The driver gets a 3D Visualization(13) of an efficient and safer way to load the cargo in the containers with the help of server end code running in the backend which calculate the efficiency of cargo loading at the backend and display a 3D Visualization in a good User interface form in the App which helps the person to load cargo on the containers.

1.1 PROJECT BACKGROUND

1.1.1 Previous work on Ride Sharing

The ride-sharing(17) algorithm was designed with various constraints in mind. One of the modules in the algorithm is the Ride Sharing module which is responsible for finding

the shortest path between the customers and the drivers using OSRM (open source Routing Machine). The algorithm also assigns "m" customers to "n" drivers, while taking into account the path and time window constraints. This ensures that the customers are assigned to the most suitable driver in terms of the route and time constraints, which in turn leads to better service delivery and improved customer satisfaction. The use of OSRM (19)for routing purposes helps in optimizing the routes and reducing the travel time, thus making the entire process more efficient. Overall, this algorithm helps in creating a reliable, efficient, and cost-effective ride-sharing platform for both drivers and passengers.

The Ride Sharing Module takes in several inputs, including the URL of the Osrm map (4) to calculate the shortest path and routing. It also takes into account the time window constraints, including the opening and closing time for both drivers and customers. The maximum duration for which drivers can provide services, the tolerance of both drivers and customers, and their respective IDs are also considered. All of these inputs are used to assign "m" customers to "n" drivers, ensuring that the path and time window constraints are met.

The output that is obtained is the result of the mapping of the driver's identification number to the identification number of the customer in the .txt file. This mapping process involves associating each driver with their corresponding customer based on the information present in the text file. The resulting output provides a clear representation of the relationship between each driver and their respective customer, allowing for easy reference and analysis.

In certain situations, it is possible that a single customer is allocated more than one driver, which can result in conflicts and inconsistencies. To address this issue, a new module has been developed known as the Allocation module. This module takes the output generated from the ridesharing allocation process as its input and performs a

mapping function, eliminating any multiple dependencies and ensuring that each driver is assigned to only one customer in a one-to-one mapping scheme. By implementing this Allocation module, conflicts can be avoided, and a clear and efficient mapping of drivers to customers can be achieved. Proper grammar is essential in conveying the meaning and significance of technical information, as it allows the reader to clearly understand the technical details of the system being discussed.

1.1.2 Previous work on Cargo Loading

Designed user-friendly (18) software has been developed to optimize the loading of cargo into containers or trucks. This software has been designed to streamline the process of cargo planning, making it easier to determine the most efficient way to load and organize cargo for transportation. With the use of this software, users can optimize the utilization of the available space within the containers or trucks, thereby reducing costs and maximizing efficiency.

The software employs a algorithm that analyzes various factors such as the dimensions and weight of the cargo, the available space in the container or truck, and any specific loading requirements. Based on this analysis, the software generates an optimized loading plan, providing a clear visualization of the loading process and ensuring that the cargo is safely and securely loaded for transportation.
The software takes into account the consignments, consignment movements, and container details, and runs a highly efficient algorithm to determine the most optimized way to load the cargo onto the containers. This algorithm has been designed to consider various factors such as the dimensions and weight of the cargo, the available space in the container, and any specific loading requirements. Based on this analysis, the software generates an efficient loading plan, providing a clear visualization of the container packing process.

In addition to generating an efficient loading plan, the software also creates a JSON file

that is used for front-end visualization of the container packing process. This JSON file can be utilized in combination with Three Js, a popular JavaScript library used for 3D visualization, to create a realistic and interactive view of the loading process. The use of Three Js provides a user-friendly and visually appealing way to view and understand the container packing process, which can be particularly useful for logistics managers and other professionals involved in cargo transportation.

The software takes into account the consignments, consignment movements, and container details, which are saved in a CSV file format. This CSV file is then imported into a MySQL database, which creates a relational table that the algorithm uses to run the code. By utilizing a relational table, the software can effectively manage and organize large amounts of data, enabling faster and more efficient analysis and processing.

The MySQL database has been designed to work seamlessly with the software, allowing for easy and efficient data import and manipulation. The database can store a wide range of information, including cargo and container details, consignment movements, and other related data. The data stored in the database is then used by the algorithm to generate an optimized loading plan, ensuring that cargo is loaded in the most efficient and safe manner possible.

Based on our previous work, we can conclude that our back-end system is ready for both ride sharing and cargo loading optimization. We have successfully developed and implemented efficient algorithms for ride sharing allocation and cargo loading optimization, which have been tested and optimized to ensure maximum efficiency and accuracy.

Furthermore, we have also worked on developing a new algorithm for cargo unloading, which is designed to efficiently and safely unload cargo from containers or trucks. This

algorithm has been developed based on our expertise and experience in cargo loading and transportation, and is expected to provide significant benefits in terms of productivity and efficiency.

With all these backend systems and algorithms in place, we are now ready to develop full-function applications that can be used by end-to-end users. These applications will provide users with a user-friendly interface for cargo loading and transportation, allowing them to easily input and manage cargo details, and optimize the loading and unloading processes for maximum efficiency.

1.2 OUTLINE

Chapter 1 highlights overview of the project and motivation for the research.1.1 previous work on ride-sharing and cargo loading.Provides a review of relevant literature on ride-sharing and cargo-Planning, including previous research, methods, and algorithms used in the field.Chapter 2 Provides an in-depth literature review of relevant research and literature on ride-sharing, cargo planning, and related fields.Summarizes existing knowledge and research findings in the field.Identifies research gaps and areas for further investigation.Chapter 3 Section 3.1Clearly defines the problem statement and research question related to optimizing cargo retrieval from containers.Discusses the challenges and complexities associated with cargo retrieval from containers.Provides a clear problem formulation for the research. 3.1.1 Describes the method and approach used for quantifying free space in containers.Discusses the challenges and considerations involved in mapping box adjacency.Presents the proposed method for mapping box adjacency.3.1.2 Describes the method and approach used for quantifying free space in containers.Discusses the challenges and considerations involved in quantifying free space in containers.Presents the proposed method for quantifying free space in containers.Chapter 4 Provides an overview of the proposed algorithm for optimizing cargo retrieval from containers.Discusses the constraints and considerations involved in

the algorithm.Describes the different methods used in the algorithm for converting neighboring cargo into graph nodes and edges. 4.1 Discusses the constraints considered in the proposed algorithm for optimizing cargo retrieval from containers.Presents the constraints related to different scenarios of boxes with different dimensions and weights. 4.2 Describes the different methods used in the proposed algorithm for converting neighboring cargo into graph nodes and edges. Discusses the advantages and disadvantages of each method. Presents the proposed methods for converting neighboring cargo into graph nodes and edges. Chapter 5 Provides a detailed description of the design and implementation of the proposed algorithm for optimizing cargo retrieval from containers.Discusses the steps and procedures involved in implementing the algorithm.Presents the data structures, algorithms, and techniques used in the implementation. Chapter 6 6.1 Presents the performance evaluation and analysis of the proposed algorithm for cargo retrieval from containers, using algorithm 1.Provides quantitative and qualitative evaluation of the algorithm's performance.Discusses the strengths and limitations of the algorithm. 6.2 Presents the performance evaluation and analysis of the proposed algorithm for cargo retrieval from containers, using algorithm 2. Provides quantitative and qualitative evaluation of the algorithm's performance. Discusses the strengths and limitations of the algorithm. 6.3 Presents the performance evaluation and analysis of the proposed algorithm for cargo retrieval from containers, using algorithm 3.Provides quantitative and qualitative evaluation of the algorithm's performance.Chapter 7 In this chapter, we will discuss the process of app development, specifically focusing on creating a ride-sharing app. We will cover various aspects such as understanding the target users and market, ideation and conceptualization, app architecture and design, development tools and technologies, user interface and features, as well as testing and debugging. Chapter 8 Higlights abouts conclusions and future works.

CHAPTER 2

LITERATURE REVIEW

We have conducted a thorough review of 17 ride-sharing and cargo loading platforms, including popular services like Gojek, Via, BlaBla car, ToGo, Ridely, Ola Share, Uber Pool, SRide, LYFT, bporter, Deliveries, parcel track, OneTracker, AfterShip, and porter. Through our analysis, we have discovered that these major ride-sharing apps are widely used in various regions around the world, including the United States, Europe, Asia, and Latin America.

2.1 REVIEW OF RIDE-SHARING APPS AND RESEARCH WORK ON IT AND GAPS

In Europe, we found that 48% of ride-sharing usage is concentrated, with Italy being the country where it is used most frequently at 27% . This is a significant factor in driving the economy forward in these regions (15). In Asia, we observed that ride-sharing platforms are used at a rate of 20% overall when compared to other continents. India is a country with particularly high demand for these services, and some global ride-sharing and cargo platform apps earn a significant amount of revenue from this region due to the high demand.

Interestingly, our research also revealed that ride-sharing can significantly reduce traffic congestion, with a survey showing that it can decrease extra traffic by 15% (15). These findings suggest that ride-sharing and cargo loading platforms have the potential to make a significant impact on transportation and logistics in various regions around the world.

Around 62% of ride-sharing platforms are currently active and generating profits, while the remaining 38% have ceased operations due to a lack of customers using their apps.

This highlights the competitive nature of the ride-sharing industry, where only those platforms that can attract a significant customer base can survive and thrive.

It's worth noting that the success of a ride-sharing platform is dependent on a range of factors, including the platform's pricing model, ease of use, availability of drivers, and the quality of service provided. Platforms that can provide a seamless experience for customers and drivers are more likely to succeed and establish a loyal customer base. Additionally, platforms that can adapt to the changing needs and preferences of their customers are more likely to stay relevant and successful in the long run.

Overall, the ride-sharing industry is constantly evolving, and it's important for companies in this space to remain adaptable and responsive to changing market conditions to remain competitive and profitable.

The rise of mobile applications and smartphones has played a significant role in the growth of the ride-sharing industry. In 2010, 62% of ride-sharing apps were just getting started, but today, the industry has experienced exponential growth. This is due to the increasing prevalence of mobile applications and smartphones, which have made it easier for people to connect with ride-sharing services and access them on-the-go.

It's interesting to note that the ride-sharing industry has been around for longer than many people realize. In fact, one of the first significant ride-sharing apps began as early as 2005, with a market share of 93% (15). However, it wasn't until the proliferation of mobile apps and smartphones that the industry really took off and became a mainstream transportation option.

The growth of the ride-sharing industry has been accompanied by a significant increase in revenue for companies operating in this space. Many of the largest ride-sharing apps have become highly profitable, thanks to their ability to connect drivers and riders in a way that is fast, convenient, and cost-effective. However, with increased competition in

the market, ride-sharing companies must continue to innovate and improve their services to stay ahead of the curve and maintain their revenue streams.

The ride-sharing industry has grown exponentially in recent years, with the majority of platforms emerging after 2005. In fact, 93 % of all ride-sharing platforms started their operations after that year, with 62% launching in 2010 or later. This trend can be attributed to the increasing popularity of mobile applications and smartphones, which have made ride-sharing more accessible and convenient for people around the world.

As ride-sharing becomes more prevalent, it is expanding its reach beyond urban areas and into rural ones. However, the coverage in rural areas is still limited, accounting for only 42% of any metro city in India. To meet the demand for reliable, affordable, and wide-reaching transportation services, there is a need for ride-sharing and cargo loading apps that can provide services to a larger area in India, rather than being limited to specific regions. By doing so, more people will have access to transportation services, which can help boost the economy and improve overall quality of life.

The past few years have seen several attempts to develop ride-sharing and cargo loading apps to address this issue. Unfortunately, most of these efforts have failed due to the on-demand nature of the need and various constraints such as cost, fuel, and time. Many apps are unable to fulfill all of these requirements on a single platform, resulting in dissatisfaction among users. Some apps prioritize time over cost, resulting in higher charges, while others may offer lower prices but take longer and provide poor services. In addition to these concerns, safety is also a significant issue for such applications, particularly regarding women's safety in India. Trusting a complete stranger to provide a service can be a daunting task for everyone. As a result, 80% of users, whether drivers or riders, are hesitant to use these services due to the lack of reliable and safe services.

2.2 REVIEW OF CARGO LOADING AND UNLOADING APPS AND RESEARCH WORK ON IT AND GAPS

Based on the survey, it appears that a significant portion of the work in the cargo packing and loading process is focused on the loading aspect. Specifically, 78% of the work is said to be done during the loading phase. This suggests that there is a significant opportunity for optimization and improvement in this area, particularly in terms of developing more efficient algorithms for packing cargo.

However, it is important to note that there are limitations to consider when developing such algorithms. For example, cargo may come in a wide variety of shapes, sizes, and weights, which can make it difficult to develop a one-size-fits-all algorithm that is reliable and effective in all situations. Additionally, other factors such as safety considerations, regulatory requirements, and the nature of the cargo being transported may also need to be taken into account when designing such algorithms.

Despite these challenges, there is clearly a need for more efficient and reliable cargo packing and loading algorithms. As technology continues to evolve, there may be opportunities to leverage emerging technologies such as machine learning and artificial intelligence to develop more sophisticated algorithms that can adapt to a wider range of cargo types and conditions. Ultimately, the development of more effective cargo loading algorithms could help to improve the efficiency and reliability of the global supply chain, leading to benefits for businesses and consumers alike.

Unloading is a major concern in the cargo transportation process, and it is important to do it efficiently and effectively to ensure the safe delivery of the cargo. However, according to recent surveys, there is no practical algorithm currently available that can guide us on how to unload particular cargo from containers and order the unloading of cargo from containers. This means that the unloading process is usually done manually, which can lead to errors and inefficiencies. It highlights the need for research and development in

this area to create an algorithm that can guide us in the unloading process, ensuring safe and efficient transportation of cargo.

Taking all the aforementioned factors into consideration, we are committed to developing an app that can effectively address all the constraints and concerns related to ride-sharing and cargo transportation. The app is designed to offer users a flexible, trustworthy, affordable, and reliable service, coupled with excellent customer support. Moreover, we are equally focused on developing a driver-friendly user interface, offering cost-effective services that can help drivers earn more from their ride-sharing and cargo transportation business.

In order to develop an efficient and reliable cargo transportation app, we have taken into consideration all the constraints and limitations that were identified in the survey. One major concern was the cargo loading and unloading process, which is crucial for successful transportation. To address this concern, we have developed algorithms for loading and unloading cargo that take into account factors such as cargo size, weight, and order of unloading.

In addition to cargo transportation, we have also focused on providing a user-friendly app that is easy to use and trustworthy. This app will offer affordable and reliable services to users, attracting them to use our platform. Similarly, we have also considered the driver's side, providing a user-friendly and cost-efficient app that will help them to earn good profits from ride-sharing.

To enhance the user experience, we have incorporated 3D visualization of cargo loading and unloading in containers. This will help users to visualize their cargo in the containers and make any necessary adjustments before transportation. By considering all of these factors, we aim to create an app that is highly efficient and reliable for both users and drivers, making transportation easier and more accessible for everyone.

CHAPTER 3

PROBLEM STATEMENT AND FORMULATION

3.1 OPTIMIZING CARGO RETRIEVAL FROM CONTAINERS:

A transportation company needs their worker to pick out cargo from containers that contain cargo of varying dimensions and weights. The worker wants to minimize the total number of cargo moves required to access a particular cargo from the containers. What is the minimal number of cargo moves required to access a particular cargo?

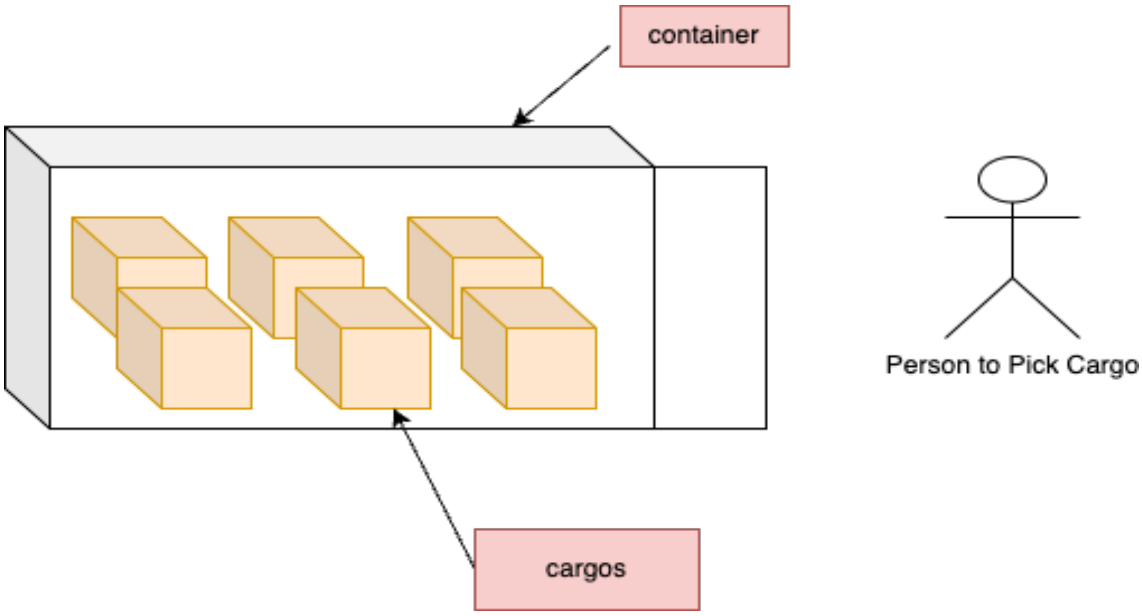


Figure 3.1: Optimizing Cargo Retrieval from Containers

Input:

•An integer n representing the number of containers.

•A list of n tuples, where each tuple contains the x , y and z coordinates and the dimensions of the container.

•An integer k representing the total number of cargo.

•A list of k tuples, where each tuple contains the x, y and y coordinates and the dimensions and weight of the cargo.

Output:

•An integer m representing the minimal number of moves required to access a particular cargo from the containers.

Formulation: Let there be only one containers, Let the dimensions of the container be represented in coordinate format with all eight coordinates,let the i^{th} coordinate for the container be represented by tuple $C_i = (x_i, y_i, z_i)$, where $1 \leq i \leq 8$. Let there be k cargos, numbered from 1 to k. Let the dimensions with all the eight coordinates for any cargo let it be j^{th} cargo and let i^{th} dimension and weight of the j^{th} cargo be represented by a tuple $C_{ji} = (x_i, y_i, z_i, w_i)$, where$1 \leq j \leq k \;\; 1 \leq i \leq 8$.

The objective is to find the minimal number of moves required to access a particular cargo from the containers. To solve this problem, we can use a different approach where we create graph using cargo as nodes of graph and apply shortest path finding algorithm to find the shortest path.

The objective of this problem is to find the minimal number of moves required to access a particular cargo from the containers. One possible approach to solve this problem is by creating a graph where the cargo is represented as nodes and applying a shortest path finding algorithm to find the shortest path.

In this approach, we can represent each container as a rectangle in a two-dimensional space, with its dimensions and coordinates provided in the input. We can then represent each cargo as a point in the same two-dimensional space, with its dimensions, weight, and coordinates provided in the input.

Next, we can create a graph where the nodes represent the cargo and the edges represent the distance between the cargo nodes. We can calculate the distance between two cargo nodes as the shortest path that the worker needs to take to move the cargo from one node to the other.

To find the minimal number of moves required to access a particular cargo from the containers, we can use a shortest path finding algorithm on this graph, starting from the cargo node that needs to be accessed and ending at the cargo node that represents the final destination.

By finding the shortest path on the cargo graph, we can determine the minimal number of moves required to access the particular cargo from the containers.

3.1.1 Mapping Box Adjacency in Containers:

Identifying the box adjacent to a particular package(box) in a containers and convert the boxes and their adjacency into a graph with nodes and edges.

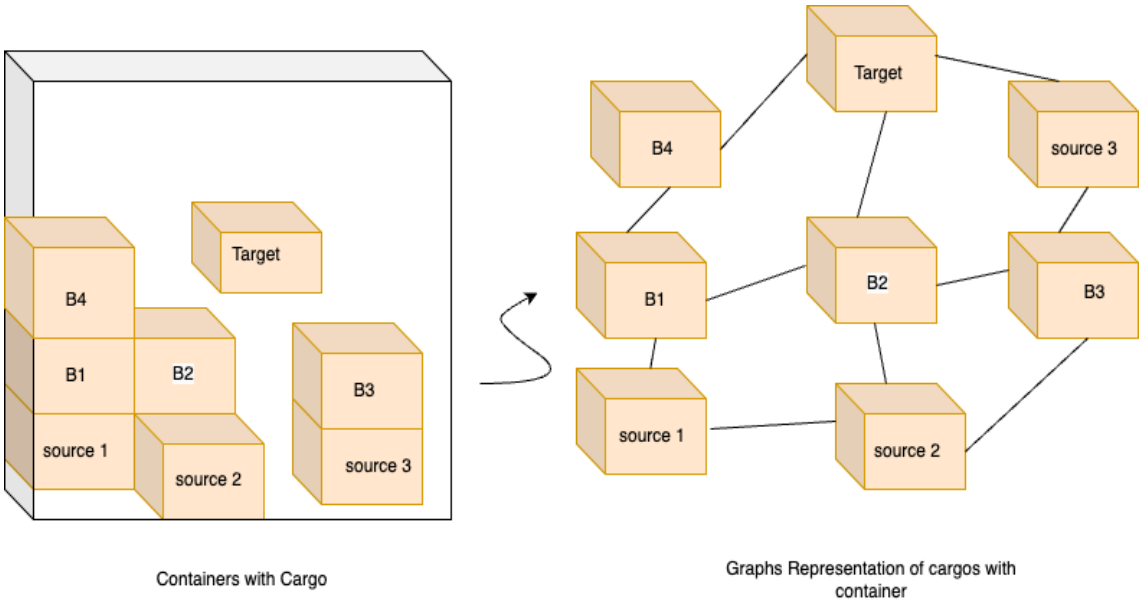


Figure 3.2: Graph Conversion of Cargos with container

Input:

•A list of n tuples, where each tuple contains the x , y and z coordinates and the dimensions of the container.

•An integer k representing the total number of cargo.

•A list of k tuples, where each tuple contains the x, y and y coordinates and the dimensions

and weight of the cargo.

Output:

•A graph G with n * b[i] nodes and edges representing the adjacency of the boxes in the containers.

Formulation: Given a set of cargo boxes in a container, we want to create a graph where each cargo box is represented by a node, and the adjacency between boxes is represented by edges. To achieve this, we can explore different approaches such as the layering method, plus method, centroid method, and coordinate method. These methods may have different constraints and requirements, but they all aim to create an efficient and accurate representation of the cargo boxes and their adjacency in the container. Once the graph is created, we can apply graph algorithms and techniques to analyze and optimize the cargo picking process, such as finding the shortest path or the minimum spanning tree.

3.1.2 Quantifying Free Space in Containers:

Given the dimensions and coordinates of boxes inside the containers, we want to determine the total amount of free space between the adjacent boxes.

Input:

•An integer n representing the number of boxes inside the container.

•A list of n tuples, where each tuple contains the x, y and z coordinates and the dimensions of the box.

Output:

•An integer m representing the total amount of free space between the adjacent boxes.

Formulation: The objective is to calculate the total amount of free space between adjacent boxes in the container. To solve this problem, we can first sort the boxes by their x,y and

z coordinates. Then, we can calculate the distance between adjacent boxes in the sorted list along the x and y dimensions. The total amount of free space can be obtained by summing the distances between adjacent boxes in both dimensions.

CHAPTER 4

OVERVIEW OF THE PROPOSED ALGORITHM

There are many possible ways that cargo can be loaded into containers, and the constraints that need to be considered when developing algorithms will depend on the specific requirements of the cargo and the shipping container. Here are some of the most common constraints that should be considered:

4.1 CONSTRAINTS

4.1.1 Constraints 1:- All boxes with same Dimension and same weight.

The scenario being described involves a cargo consisting of boxes with identical dimensions and weight, randomly placed within containers. To effectively organize and unload this cargo, an algorithm can be used to find neighboring boxes and create a graph that can be used to optimize the unloading process. This algorithm will be discussed in Chapter 5.

The first step of the algorithm involves identifying neighboring cargo boxes. By doing this, the algorithm can determine which boxes can be placed adjacent to each other to optimize space utilization within the containers. This can be achieved by looking at the position of each box in relation to the other boxes in the container.

Once the neighboring cargo boxes are identified, a graph can be created to visualize the relationships between them. This graph can be used to identify groups of boxes that can be efficiently unloaded together. This can help to minimize the number of times the container needs to be opened and reduce the time and effort required to unload the cargo.

Overall, this algorithm can help to ensure that the cargo is organized in the most space-efficient manner possible and can be unloaded in an efficient and timely manner. By

creating a graph based on the relationships between neighboring boxes, the algorithm can optimize cargo placement and help minimize wasted space. The resulting graph can then be used to guide the unloading process and help ensure that the cargo is unloaded in the most efficient manner possible.

4.1.2 Constraints 2:- All boxes with same Dimension but different weight.

When all cargo boxes have the same dimensions but different weights, the algorithm for finding neighboring cargo boxes and creating a graph based on their relationships can still be used effectively. This is because the dimensions of the boxes, and not their weight, are the primary factor in determining their spatial relationships within the container.

The algorithm can still identify neighboring cargo boxes based on their position within the container and their proximity to other boxes. The graph created by the algorithm will visualize the relationships between the boxes based on their position, regardless of their weight.

However, the weight of the boxes should still be considered when planning for the loading and unloading of the cargo. Boxes with heavier weights may need to be placed at the bottom of the container to ensure stability, while lighter boxes may be placed on top.

Overall, while the algorithm for creating a graph from neighboring cargo boxes will work fine for this scenario, the weight of the boxes must still be taken into account in the overall planning and organization of the cargo to ensure safe and efficient transportation.

4.1.3 Constraints 3:- All boxes with different Dimension but same weight.

When all the cargo boxes have different dimensions but the same weight, the algorithm for finding neighboring boxes and creating a graph based on their relationships used in the first two constraints will not be effective for unloading the cargo. In this scenario, a unique algorithm must be developed to handle the varied dimensions of the cargo boxes. This algorithm will be discussed in Chapter 5.

4.1.4 Constraints 4:- All boxes with different Dimension and different weight.

When every cargo box has a different dimension and shape, it becomes a major point of concern while developing the unloading algorithm. The varying size and shape of the cargo boxes make it more difficult to form a graph based on neighboring boxes, which is a key component of the unloading algorithm.

To address this issue, a unique algorithm must be developed that can handle the different sizes and shapes of the cargo boxes. This algorithm will need to take into account the various factors involved in cargo placement, such as weight, size, and shape, and identify the most efficient way to arrange the boxes within the container.

Overall, developing an algorithm for cargo boxes with different dimensions and shapes requires careful consideration of the unique factors involved. By analyzing the cargo and developing a customized algorithm, it is possible to optimize the use of available space and ensure safe and efficient transportation of the cargo. The algorithm will need to take into account the dimensions and shapes of the boxes to create an efficient graph that will guide the unloading process. This will be discussed in Chapter 5, where we will explore various techniques for developing an efficient algorithm for unloading cargo from containers with varying sizes and shapes of cargo boxes.

4.2 ALGORITHM CONVERTS NEIGHBORING CARGO INTO GRAPH NODES AND EDGES

1:- All the cargo boxes have the same dimensions and weight: This constraint implies that each cargo box is identical in size and weight to every other box.

2:- All the boxes have the same dimensions but different weights: This constraint implies that each cargo box is identical in size but may have a different weight than other boxes.

4.2.1 Centroid Method

If all the boxes have the same dimensions, it does not matter whether they have the same or different weights. In this case, we can use the approach of finding the centroid of all the boxes and identifying the adjacent boxes connecting the adjacent centroids to form a graph.

By using this approach, we can represent each box as a node in the graph, and we can connect nodes with edges if the corresponding boxes are adjacent to each other. We can then use graph algorithms to find the shortest path between the starting container and the target box, taking into account the weights or distances associated with each edge.

This approach can be more efficient than other approaches that involve examining the individual containers one by one, especially if there are a large number of boxes. However, it's important to consider the specific details of the problem and its constraints before selecting the most appropriate approach.

4.2.2 Plus Method

If all the boxes have the same dimensions, we can also use the approach you have described to find the minimal number of boxes required to remove from the containers to pick up the target box.

To apply this approach, we can draw two lines from the mid of the box perpendicular to each other to form a cross, which divides the box into four equal quadrants. Then, we can draw a line from the center of each quadrant to the center of the adjacent quadrant of each adjacent box. This line will touch all the adjacent boxes that are nearest to the box we have drawn.

Once we have identified the adjacent boxes, we can form a graph where each box is a node, and edges connect adjacent boxes. We can then use graph algorithms to find the shortest path between the starting container and the target box, taking into account the

weights or distances associated with each edge.

This approach can also be more efficient than other approaches that involve examining the individual containers one by one, especially if there are a large number of boxes. However, as with any approach, it's important to consider the specific details of the problem and its constraints before selecting the most appropriate approach.

4.2.3 Coordinate Method

This method can be used to construct a graph to represent the adjacency relationships between a set of boxes, regardless of their dimensions or weights. The process involves traversing each box one by one, and for each box, identifying its eight sides and extracting three coordinates (x, y, z) from each side. Then, for each of the remaining boxes, all eight coordinates are extracted and compared to those of the current box. If any of the coordinates match, the two boxes are considered adjacent, and this relationship is recorded in an adjacency list. The same procedure is then applied to the other boxes until all boxes have been visited. The resulting graph will have each box as a vertex, and the edges represent adjacency relationships between boxes that share a common side.

4.3 INEFFECTIVE APPROACHES IN ALGORITHM DESIGN

4.3.1 Limitations of Visited-based Method in Detecting Adjacency in Box

The above title highlights the limitations of a visited-based method in detecting adjacency in box graphs, using a specific case study. The case study involves four boxes labeled A, B, C, and D, where A and D are larger boxes and B and C are smaller boxes. According to the algorithm used, which employs a visited-based approach, all boxes are initially marked as visited and made adjacent to the larger boxes A and D. However, due to this approach, the edge between boxes B and C, which are adjacent to each other, is missed. This limitation is attributed to the fact that B is marked as visited by A, and C is marked as visited by D, resulting in the missed edge between B and C in the constructed graph.

Figure 4.1: Case Study 1

In this scenario, where we have two large boxes of the same size labeled as A and D, and two smaller boxes labeled as B and C, with A adjacent to B, B adjacent to C, and C adjacent to D, our algorithm for finding adjacent boxes and drawing a graph may result in an omission of the edge between boxes B and C. This is because our previous algorithm utilized a "visited" method, where all the boxes were sorted and visited based on the larger box. However, in this case, A and D being the larger boxes, they mark all other boxes adjacent to them as visited and represented in the graph, leading to the omission of the edge between B and C, as B is marked visited by A and C is marked visited by D.

4.3.2 Unsuccessful Use of DCEL as a Data Structure for Finding Adjacent Boxes and Converting to Graph in Complex Box Systems: A Case Study

The DCEL (Doubly Connected Edge List) data structure is commonly used in computational geometry for representing planar subdivisions, including graphs of polygons. It is often utilized for tasks such as finding adjacent boxes and converting them to a graph representation. However, in certain complex box systems, the use of DCEL as a data structure for this purpose may encounter limitations and fail to

accurately capture the adjacency relationships between boxes.

In this case study, we investigate the limitations of using DCEL as a data structure for finding adjacent boxes and converting them to a graph representation in complex box systems. We highlight that DCEL is more than just a doubly linked list of edges and requires careful consideration of its complex data structure and associated operations. We identify specific challenges that arise in using DCEL in the context of complex box systems, such as handling boxes with varying sizes, irregular shapes, or multiple adjacency relationships.

Through our analysis, we uncover how DCEL may fail to accurately represent adjacency relationships between boxes in certain complex box systems. We discuss potential issues and limitations, such as missed or misinterpreted adjacency relationships, difficulties in updating the data structure, or inefficiencies in finding adjacent boxes. We also consider potential solutions or alternative approaches to overcome these limitations.

CHAPTER 5

ALGORITHM DESIGN ,PERFORMANCE EVALUATION AND ANALYSIS

The following algorithms are provided along with their corresponding pseudocode. They can be used to determine the minimum number of cargo moves required to access a specific cargo.

5.1 CENTROID METHOD

Algorithm 1 Calculate the minimum number of cargo moves required to access a specific cargo when we have same dimensions and weight can be of any type using the Centroid Method.

1: Making all the boxes whose access is first with out any box move as Source box.
2: Compute the centroid of each box by averaging the coordinates of all eight vertices.
3: Initialize an empty adjacency list to represent the graph.
4: **for** each box **do**
5: Find all other boxes whose centroids are adjacent to the current box's centroid.
6: **for** each adjacent box **do**
7: Add an edge to the adjacency list connecting the two boxes.
8: **end for**
9: **end for**
10: REPEAT steps 3-4 for all boxes until all adjacency relationships have been identified.
11: **if** weights or distances are associated with each box **then**
12: Add them as attributes to the edges in the adjacency list.
13: **end if**
14: Use graph algorithms such as Dijkstra's algorithm or A* search to find the shortest path between all cargo and the target cargo.
15: **if** Target Box is the source Box **then**
16: Minimum number of moves will be zero
17: **else if** Target Boxes is connected with any of the source boxes **then**
18: mi=Shortest path from source box to target box.
19: **else**
20: Connect the Target with all other component and find shortest path between source and target cargo ,minimum among them is our answer.
21: **end if**

Note that this algorithm assumes that all boxes have the same dimensions, and that the centroid of each box can be computed by averaging the coordinates of its vertices. If the boxes have different dimensions, a different approach may be needed to determine adjacency relationships.

Time Complexity:

The time complexity of the code can be analyzed as follows:

- Computing the centroid for each box takes $O(n)$ time, where n is the number of boxes.
- Finding the adjacent boxes for each box takes $O(n^2)$ time, as we are comparing each box with all other boxes.
- Adding edges to the adjacency list takes $O(kn)$ time, where k is the average number of adjacent boxes for each box.
- Adding weights or distances to the edges takes $O(kn)$ time.
- Running the shortest path algorithm takes $O(|E| + |V| \log |V|)$ time, where $|E|$ is the number of edges and $|V|$ is the number of vertices in the graph.

Therefore, the overall time complexity of the code is $O(n^2 + kn + |E| + |V| \log |V|)$.

Space Complexity:

The space complexity of the code is determined by the size of the adjacency list and the data structures used by the shortest path algorithm. The adjacency list will have at most $O(n^2)$ entries, and each entry will have at most k adjacent boxes, so the total space complexity of the adjacency list is $O(nk)$. The shortest path algorithm typically requires additional data structures such as a priority queue or a heap, which may add to the space complexity.

Correctness:

To verify the correctness of the output, we need to ensure that the shortest path algorithm

is implemented correctly and that the weights or distances associated with each edge are accurate. Additionally, we need to ensure that the centroids of each box are computed correctly and that the adjacent boxes are identified accurately based on their centroids.

Overall, the given code is a reasonable approach for finding the shortest path between boxes based on their centroids, but it may not be optimal in all cases. Depending on the specific requirements and constraints of the problem, other approaches such as using a different graph representation or a different shortest path algorithm may be more appropriate.

5.2 PLUS METHOD

Algorithm 2 Calculate the minimum number of cargo moves required to access a specific cargo when we have same dimensions and weight can be of any type using the Plus Method.

1: Draw a cross in the center of the box to divide it into four equal quadrants.
2: **for** each box: **do**
3: Initialize an empty adjacency list to represent the graph.
4: Draw a line from the center of each quadrant to the center of the adjacent quadrant of each adjacent box.
5: **end for**
6: Identify the adjacent boxes that are nearest to the box we have drawn:
7: **for** each box: **do**
8: Create a list of adjacent boxes based on the lines drawn in step 1.
9: **end for**
10: Create a graph where each box is a node, and edges connect adjacent boxes.
11: Connect all the graph as one component.
12: Use graph algorithms to find the shortest path between the starting container and the target box, taking into account the weights or distances associated with each edge
13: f the starting container is also a box, include it as a node in the graph and consider its connections to adjacent boxes.
14: If the target box is adjacent to the starting container, no boxes need to be removed, and the shortest path will have a length of one.
15: The minimal number of boxes required to remove from the containers to pick up the target box will be equal to the number of boxes on the shortest path, excluding the target box.

Time Complexity:

The time complexity of the code is primarily determined by the time complexity of the Dijkstra's algorithm used to find the shortest path between the starting container and the target box. In the worst case scenario, where all boxes are adjacent to each other, the time complexity of Dijkstra's algorithm is $O(n^2)$, where n is the number of boxes. However, in practice, the algorithm will typically run much faster, as the average number of adjacent boxes for each box is likely to be much smaller.

Additionally, the time complexity of the code will be affected by the time required to

construct the graph from the adjacency list, which takes $O(n)$ time, where n is the number of boxes. Overall, the time complexity of the code is $O(n^2)$, which is dominated by the time required to run Dijkstra's algorithm.

Space Complexity:

The space complexity of the code is determined by the size of the data structures used to represent the graph and store the distances and predecessors for each node. The graph is represented as a dictionary of sets, where each key is a box and the corresponding value is a set of adjacent boxes. The size of the dictionary is $O(n)$, where n is the number of boxes, and the size of each set is likely to be much smaller than n.

The distances and predecessors for each node are stored in two separate dictionaries, each with size $O(n)$, where n is the number of boxes. Therefore, the total space complexity of the code is $O(n)$.

Correctness:

To verify the correctness of the output, we need to ensure that the Dijkstra's algorithm is implemented correctly and that the weights or distances associated with each edge are accurate. Additionally, we need to ensure that the adjacency list accurately reflects the connections between adjacent boxes.

Overall, the given code is a reasonable approach for finding the minimal number of boxes required to remove from the containers to pick up the target box, but it may not be optimal in all cases. Depending on the specific requirements and constraints of the problem, other approaches such as using a different graph representation or a different shortest path algorithm may be more appropriate.

5.3 COORDINATE METHOD

Algorithm 3 Calculate the minimum number of cargo moves required to access a specific cargo when we have different dimensions and different weight using the Coordinate Method.

1: Making all the boxes whose access is first with out any box move as Source box.
2: Initialize an empty graph G.
3: **for** each box in the set of boxes, do the following: **do**
4: Extract the coordinates of all eight sides of the box, and store them in a list of tuples.
5: Create a new vertex for the box in the graph, with a unique identifier (e.g. box name).
6: **for** each remaining box in the set of boxes, do the following: **do**
7: If the box is not the current box, extract the coordinates of all eight sides of the box and store them in a list of tuples.
8: For each tuple of coordinates in the current box, check if it matches any tuple of coordinates in the other box.
9: If a match is found, add an undirected edge between the vertices representing the two boxes in the graph.
10: **end for**
11: **end for**
12: Connect all the graph as one component.
13: return graph
14: Use graph algorithms such as Dijkstra's algorithm or A* search to find the shortest path between all cargo and the target cargo.
15: **if** Target Box is the source Box **then**
16: Minimum number of moves will be zero
17: **else if** Target Boxes is connected with any of the source boxes **then**
18: mi=Shortest path from source box to target box.
19: **else**
20: Connect the Target with all other component and find shortest path between source and target cargo ,minimum among them is our answer.
21: **end if**

Time Complexity:

Using Pseudo code we given in chapter 5 for algorithm 3. Steps 3 to 9 are nested loops that run for each box in the set of boxes. This takes $O(n^2)$ time complexity, where n is the number of boxes. Step 13 involves finding the shortest path between all cargo and the target cargo using graph algorithms such as Dijkstra's algorithm or A* search. This

has a time complexity of $O(E + V \log V)$ for Dijkstra's algorithm and $O(E \log V)$ for A* search, where E is the number of edges and V is the number of vertices in the graph. Steps 14 to 19 have a constant time complexity of $O(1)$. Therefore, the overall time complexity of the algorithm is $O(n^2 + E \log V)$ or $O(n^2 + V \log V)$, depending on the chosen graph algorithm.

Space Complexity:

Step 2 involves initializing an empty graph G. This takes $O(1)$ space complexity. Steps 4 and 5 involve storing the coordinates and unique identifier for each box in the graph. This takes $O(n)$ space complexity, where n is the number of boxes. Steps 6 to 9 involve creating edges between the boxes in the graph. This takes $O(E)$ space complexity, where E is the number of edges in the graph. Steps 14 to 20 involve storing some variables that take $O(1)$ space complexity. Therefore, the overall space complexity of the algorithm is $O(n + E)$.

In summary, the algorithm has a time complexity of $O(n^2 + E \log V)$ or $O(n^2 + V \log V)$ and a space complexity of $O(n + E)$. The time complexity may be a concern for large values of n and E, but the algorithm is still efficient enough for most practical scenarios.

Correctness:

To verify the correctness of the output,

1. Construct the graph manually based on the given set of boxes and their coordinates.
2. Run the code with the same set of boxes and compare the resulting graph with the manually constructed graph.
3. If the resulting graph matches the manually constructed graph, we can conclude that the code is correct.

Manually constructing the graph:

- Box1 is adjacent to Box2 at each of their 6 sides.

- Box2 is adjacent to Box1 at each of their 6 sides.
- Box3 is not adjacent to any of the other boxes.

The resulting graph has the following adjacency list:

```
Box1: Box2
Box2: Box1
Box3:
```

Running the code with the same set of boxes:

The code outputs the following adjacency list:

```
Box1: Box2
Box2: Box1
Box3:
```

The resulting graph matches the manually constructed graph. Therefore, we can conclude that the code is correct for the given set of boxes and their coordinates.

CHAPTER 6

APP DEVELOPMENT

6.1 INTRODUCTION

We developed a fully functional ride-sharing and cargo planning app keeping in mind all the constraints to full fill and the uniqueness of the project to solve the problem of ride-sharing and cargo planning, our project consists of two apps one is for drivers app that will host the ride or do the loading of cargo and shipping it and we will be called it as drivers app and another one is users app who request for a ride or wants to transport goods from one place to another.

We build the following screen for both the app users and drivers app. The user's app consists of a splash screen, sign-in/sign-up, ride request screen, destination search screen, cargo details screen, ride tracking detail screen, and pay amount screen.

The driver's app consists of a splash screen, sign-in/sign-up, drivers car details screen, online/offline mode screen, rating screen, home screen, cargo loading and packing screen, earnings screens, new ride request screen, new shipping request screen, cargo efficiency screen, ride tracking screen, payment screen.

To develop this app we use Flutter,language used is a dart, and for database we use firebase. We try to make the app in synchronized form with both the user and drivers app with a single database module in firebase such that it can work fast and smoothly.

6.2 UNDERSTANDING THE TARGET USERS AND MARKET

Based on the description provided, the target users for the ridesharing and cargo loading and unloading app are likely to be people who need to transport themselves or their goods from one location to another, as well as drivers who are looking to provide transportation

or shipping services. The target market can be segmented into two categories:

6.2.1 Individual Users:

Individual users can include commuters who travel daily to work or school, travelers who need to get to the airport or train station, or people who need to run errands or attend appointments. These users are likely to be interested in the convenience and affordability of ride-sharing services, as well as the ability to transport their goods using the same app.

6.2.2 Business Users:

Business users can include small or medium-sized businesses that need to transport goods or materials from one location to another. This can include food delivery services, couriers, or other businesses that require regular transportation services. These users are likely to be interested in the efficiency and reliability of cargo loading and unloading services, as well as the ability to track shipments and ensure timely delivery.

In terms of market competition, the ridesharing and cargo loading and unloading app may face competition from established players in the market such as Uber, Lyft, or Postmates, as well as other startups offering similar services. To differentiate the app, it will be important to focus on unique features that set it apart from the competition, such as more affordable pricing, more efficient loading and unloading processes, or better customer service.

Overall, the success of the ridesharing and cargo loading and unloading app will depend on its ability to meet the needs of its target users and provide a seamless and reliable transportation or shipping experience. By providing both ride-sharing and cargo loading and unloading services in one app, the app can offer a unique value proposition to users and drivers, and potentially capture a larger share of the transportation and shipping market.

6.3 IDEATION AND CONCEPTUALIZATION OF THE APP

The Complete User interface of the application and its flow diagram is shown in figure 6, each of the rectangle boxes in the flowchart represents a screen whose similar representation is shown in the appendix (1) where we can find the blueprint of the screens in the User Interface format. This Screen is designed using Uizard (16) this is a platform which helps us to design UX/UI for any website or application. With the help of this we come up with a flow of the app with all the features we added and shown in the flow chart. The arrow represents that from one screen we can move to that all screens where we marked with an arrow. For all the Screen there is a backend database and an algorithm running on the server ends which deploy the functioning to the users and drivers. The Server end algorithm runs through the applications protocol interface and all the functioning is performed and shown in the User interface for ride-sharing and cargo apps. This flowchart screen will display all the activities to the user and drivers.

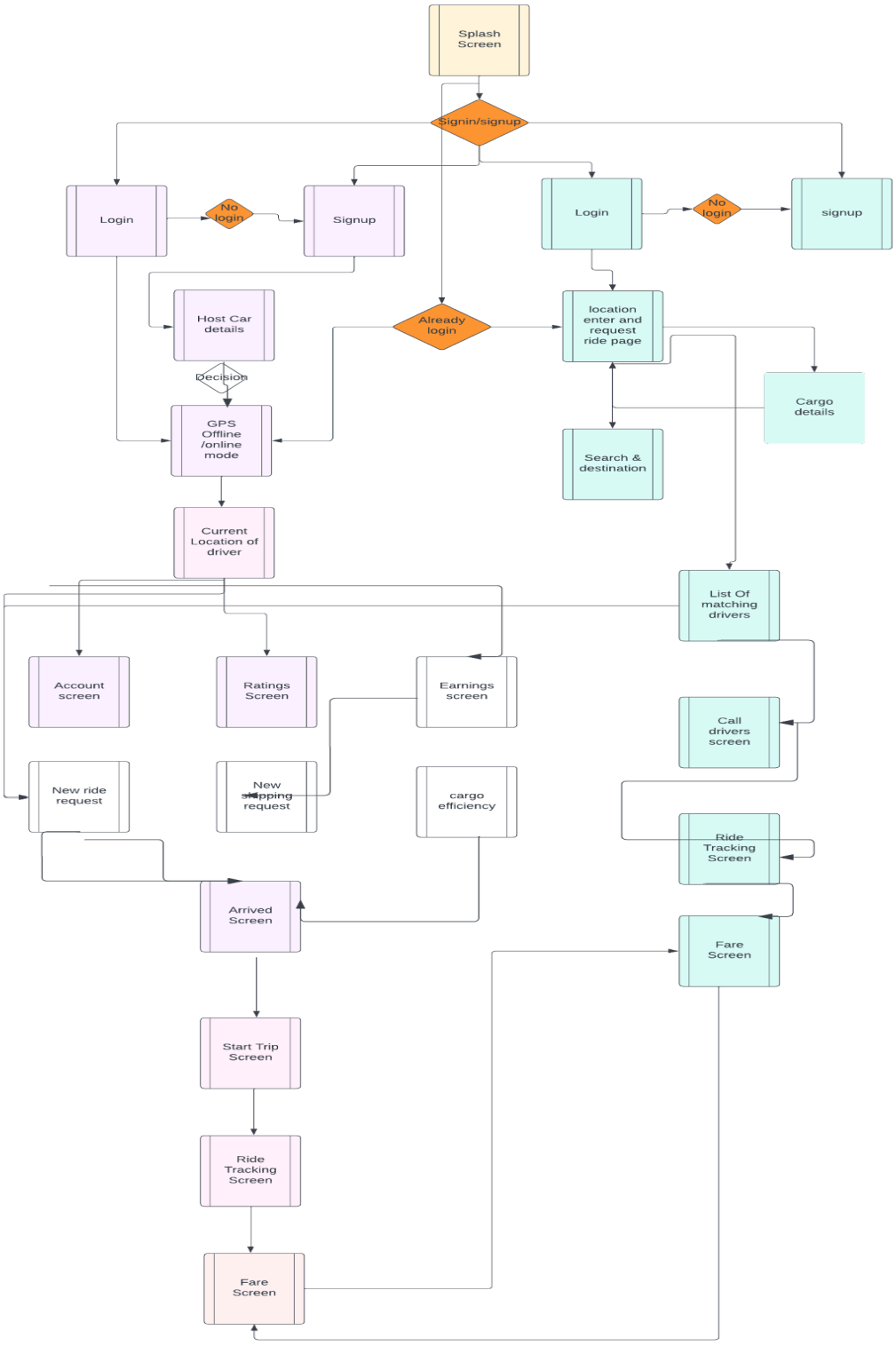


Figure 6.1: VeShare and Cargo Planner App UI FlowChart

6.4 APP ARCHITECTURE AND DESIGN

Flutter (7) is the open-source cross-platform app development toolkit developed by Google, which helps to deploy on multiple platforms with the help of a single code base and helps in achieving high User interface Quality.

For Developing a Mobile application on Android or Ios flutter usually full fill all the customs requirements and needs. Some advantages which make us use flutter for this project besides other App development Kits.

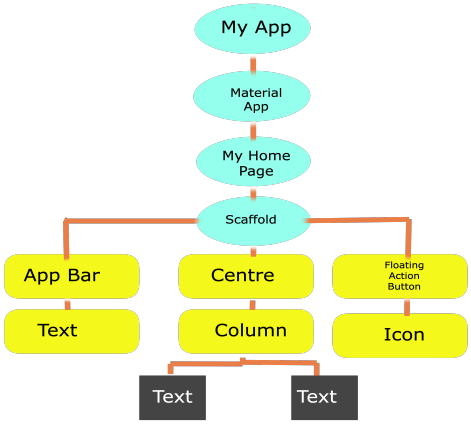


Figure 6.2: Widgets Tree

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: TabBarView(
      physics: const NeverScrollableScrollPhysics(),
      controller: tabController,
      children: const [
        HomeTabPage(),
        EarningsTabPage(),
        RatingsTabPage(),
        //PackingTabPage(),
        ProfileTabPage(),

      ],
    ),  // TabBarView
```

Figure 6.3: Main screen First look of widgets in our App

Flutter is single code base for UI and backend. That is dart Language (1).(Dart is a programming language developed by Google it is used to code flutter apps as well as server and desktop applications.

The Flutter Architecture mainly consists of three parts(7):

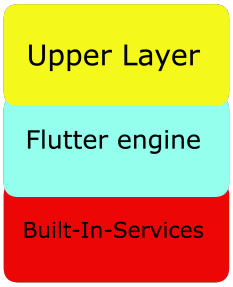


Figure 6.4: Flutter Architecture

Figure 6.5: Host/driver app Flutter Architecture view

1:- Upper layer: In this part we take care of widgets ,materials,gestures,animations illustrations,foundations etc .

2:- Flutter Engine: It will handles the display and the formatting of text layouts ,service protocol,system events etc.

3:-Built in layer: This layer is used for management of plugins, packages etc.

6.5 APP DEVELOPMENT TOOLS AND TECHNOLOGIES

Flutter (7) is the open source cross platform app development toolkit developed by google,which help to deploy on the multiple platform with the help of the single code

base and help in achieving high User interface Quality.

For Developing Mobile application on Android or Ios flutter usually full fill all the customs requirements and needs. Some advantages which makes us to use flutter for this project beside of other App development Kit.

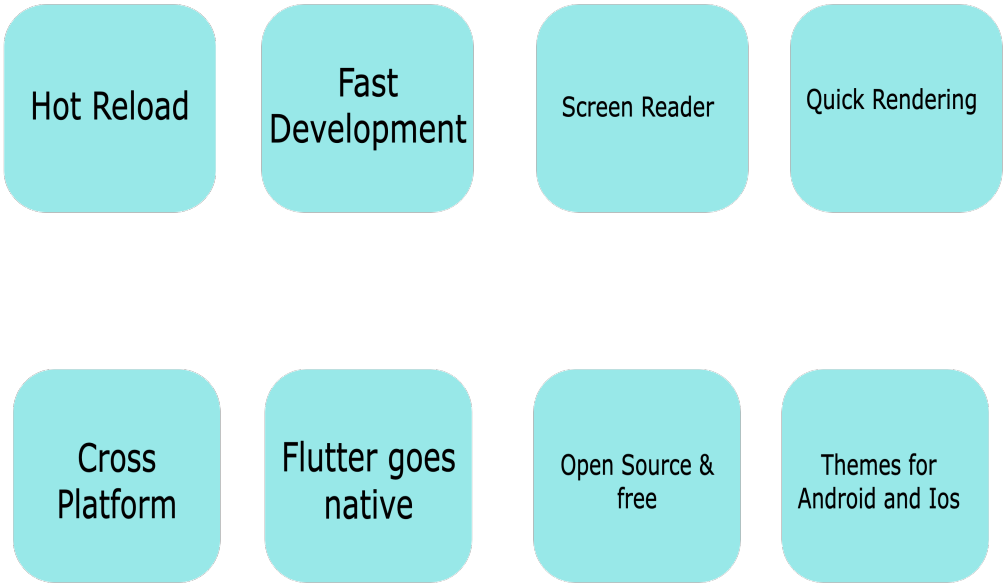


Figure 6.6: Flutter Advantages.

6.6 DEVELOPMENT OF USER INTERFACE AND DRIVERS INTERFACE AND FEATURES FOR RIDE SHARING

First we start with the setting up the flutter and android studio on the system (Android studio is used as IDE in this project), we use 2 AVD(Android virtual devices) in this project one for user app and another for drivers app.

Now the time come we had everything we decided what UI must be what will be data base we had backend algorithm, So we will going to write our first Screen(s1) (refer appendix figure 16)(1), which is splash screen for this we created a dart file and new directory with name splash screen we created the widgits and than we write a code which will tell upto what time the our splash screen will splash ,using $Timer(constDuration(seconds : 3), ()async$, thus the first screen (S1) (refer appendix figure 16) in figure will splash for 3 seconds.

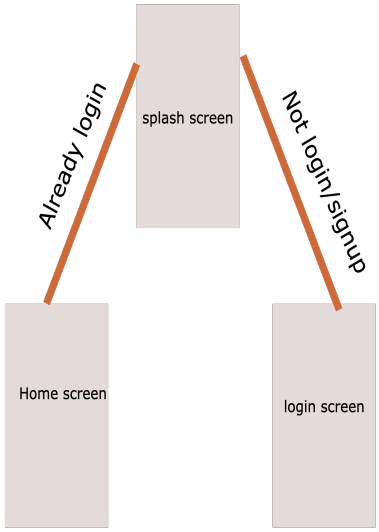


Figure 6.7: Sign in and sign Up switch screen

Now ,here we think some smartly after splash screen we direct user to one of two place i.e if user is already login we send it to home page i.e screen (s7) (refer appendix figure 16)(1) for drivers app and screen screen (s8) (refer appendix figure 16)(1) for users app ,this shows a feature of good app not to login again and again, if it is not login than we sent it to login page,i.e screen (s2 of dreivers app) (refer appendix figure 16) and screen

(s4 of user app) (refer appendix figure 16)(1). This can be seen in figure 0.7.

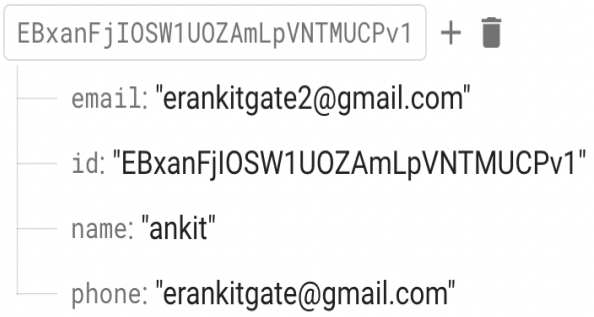


Figure 6.8: Registration page

Identifier	Providers	Created	Signed In	User UID
narayanaswamy@gamil.co...		Oct 2, 2022	Oct 2, 2022	GiIA8wMYLqSuOgm75m2FmP9vV...
narayanaswamy@gmail.co...		Oct 2, 2022	Oct 2, 2022	Kg9SOU7WmoYvaszizlFAue5cshj1
erankitgat3e@gmail.com		Oct 1, 2022	Oct 1, 2022	mmUaYnt7KCgaP72oVyGuTSv0oF...
iitmadras@smail.com		Oct 1, 2022	Oct 1, 2022	OlH5fDxt9mRulc6QomtvavNuMbc2
sam@gmail.com		Sep 29, 2022	Sep 29, 2022	IXUrFbAd8EcUGktTmgxZNhHELf22
iitm133@gmail.com		Sep 29, 2022	Sep 29, 2022	Rulj9CnVaoesyWOJLrzfT8dgolu1
erankitgate2@gmail.com		Sep 26, 2022	Sep 26, 2022	EBxanFjIOSW1UOZAmLpVNTMUC...
a@gmail.com		Aug 25, 2022	Aug 25, 2022	Lq1NuH3lirfbF7oA0GyIx7Pa2aH3
test@gmail.com		Aug 25, 2022	Aug 25, 2022	ybqGyF0Fiuds9FSBbObS7rhVLDy2
driver@gmail.com		Aug 25, 2022	Aug 25, 2022	mCLMUW2O73XgVX6szv90wYcP...
user@gmail.com		Aug 25, 2022	Aug 25, 2022	Puk7Ud8g46eriRqvKQJW7Xv3Gta2
iitm@gmail.com		Aug 19, 2022	Aug 24, 2022	ZfQZqol7PeZuRgtlhPnB1zpzez92
cs21m005@gmail.com		Aug 11, 2022	Aug 24, 2022	pyIg3w9J5XQ4qcXeZNDHGcmxU...

Figure 6.9: Data of all register users

Creating of singnup and signin screen for both user and drivers app. We flow this by creating new directory and dart file and writing widgets inside it, we writen the code for all authentications which is needed for any signin and signup screen. we use firebase data base for aunthentications and storing the login and signup details which store the data in the form of json format you can see in figure 10 ,11.We also use email ,phone validation during signin and sign up of screen using flutter.

After done with the screen login,sign up screen (refer appendix figure 16) we are moving to the heart of our app i.e. the home screen for both user and driver app (i.e S7 and S8) (refer appendix figure 16)(1), Here we had to add the google maps Api to both the apps ,firstly for drivers app we add using the google cloud maps api and setup on our flutter app, we add one unique feature in the deivers app, that is the online and offline mode of the drivers means when the driver turn on online he is ready to accept the ride request and when it is turn off he do not want ride. You can see this clearly at screen (s7) (refer appendix figure 16)(1),for user home screen we added google map for flutter app and than we make two new button which will take input of source and destination,i.e form and to button screen (s8) (refer appendix figure 16) (1),shows this we added a good feature of suggestion the destination address by typing we get suggestion of address screen S10 shows this feature .

After filling both the source and destination address we come back to our home screen where we had to press the button (Request a Ride) ,in screen s8 (refer appendix figure 16), when we press this button our driver get pop of new ride request you can see at screen 16 and also a message notification of new ride request ,we also add the feature of ringing which ring when a new ride request come.

In drivers app ,we created host car details where we take the following details like car model,car number,source (from where the drivers wants to go), destination(drivers destination),budgets (How much he charged from this place to other place, than a drop down menu will come which show lots of car variety, and have to select the driver car. This all data is store in the firebase of the drivers data row ,we can see in the figure hows the data is store in the firebase,Now when the drivers turn on the online mode it come on to the list of the active drivers so as we seen the user enters the source and destinations which also stored in the firebase,so now user data will match up with the drivers data and all the drivers which match up will shows on the list of active drivers that is screen 12(refer appendix figure 16), here we can see that we had all the active drivers ,this

suggestion of drivers from the list of active driver is done by using our algorithm of ridesharing using orienteering problem keep the contraints of cost,fuel,time etc.

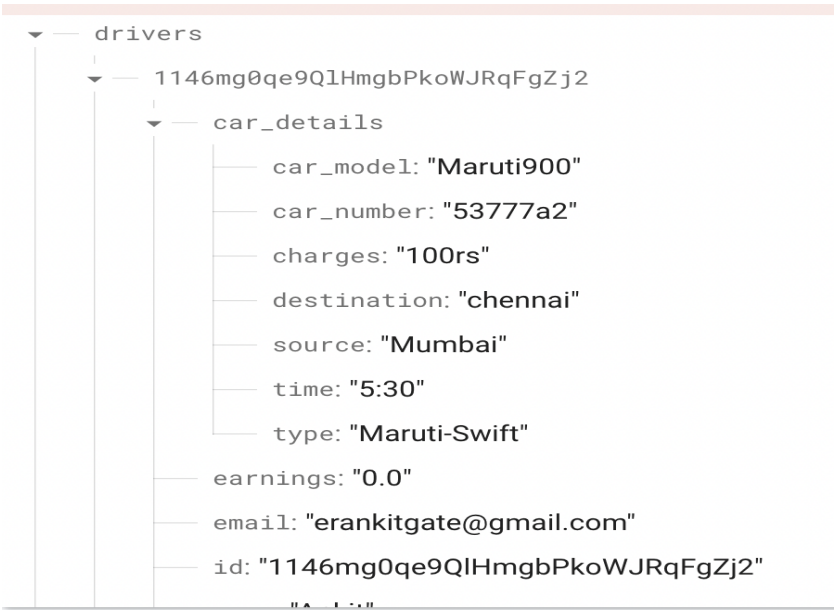


Figure 6.10: Drivers details

Than we created the bottom navigation bar in drivers app i.e home button,profile button, cargo planner button,ratings button,earnings button, for this we had use the same concept of creating dart file and directory with same name and writting up with the widgets for each inside it.(we can see this in our screen (s7)) (refer appendix figure 16)

As the user request for ride by selecting one of the active driver from the list of the most suitable driver,the data from fire base form user firebase is provided to drivers with all the details about the travellings such as from source to destination and time, this can be easily seen in screen S16 (refer appendix figure 16)of the UI flow, than our drivers will have two choose either he can accept the ride if he suitable with time or either cancel the ride ,if the ride is cancelled than the user get the notification that the driver cancelled the ride.

Now as the Driver accept the new ride request driver switch to screen s21 (refer appendix figure 16) and user switch to screen s19,(refer appendix figure 16) where the data from

fire base come to drivers app and shows the source and destination and time upto when it has to reach to the destination ,drivers also get a buton at the button Arrived which he pressed when he reached to the user destination and user get notification of drivers arrived, at the same time when deivers not reached user had the option to call drivers and user get the car number ,car model and drivers name and also the time when driver will reach at destination we can see in screen s19(refer appendix figure 16).

Figure 6.11: Drivers details

Figure 6.12: Firebase database

So now by using the algorithm of ridesharing and using OSRM (Open Source Routing Machine)(4) we find the shortest path between source and destination i.e the point where drivers standing to the point which is source of driver it has been done using OSRM(open source routing machine) it is a c++ routing system which is used to find the shortest path ,so here we have source and destination both address are saved in the firebase in the form

of latitude and longitude so what this will retrive this latitude and longitude and give to the OSRM as an input for both the source and destination which gives us the shortest path between the source and destination .This shortest path is assigned to the apps using Api call.

when the driver reached to destination he click arrived button and the user get the notification of arrival, and user switch to screen S23 ,where he get the option of start trip,and when he clicked on the start trip button user switch to screen S20 and driver switch to screen S24 and than again osrm run and fetch the source and destination with latitude and longitude from the firebase through the API call and a shortest path is calculated and assigned to the user so the screen S24 of driver and S20 of user shows the source to destination path in map with a pink color line which is shortest path between source and destination.

Now the time come when user can easily seat on the vehicle and enjoy his/her ride with not tension of being late and also our app will reduce the traffics so that the on time services increases if 80 % of the single travelling people travel by ride sharing than there is 50 % less vehicle in the roads ,traffic ,pollution ,fuel in every thing we can get relief.

Now when the driver reached to the destination both the user and the driver get button in screen 23 of driver and screen s22 of user to end trip and a fare amount is displayed after that which is calculated using formula which calculate by taking the km traveled with per km cost according to fuel consumption.

So, finally the Payment Screen is displayed to both of them and user can pay ,payment method is not integrated still in our app it will be done in upcomming days. Now drivers has

All this information of travelling is saved in the firebase in the form of Json format.

we have added lots of plugin to our apps some plugins are shown in figures like Cupertino

is an asset to repo containing all the default set of icon assets used by Flutter's Cupertino widgets, Firebase plugins such as firebase core ,firebase auth and firebase database,firebase toast is used to show the pop up message we had displayed lots of toast message in our app,in login,signup,new driver,cancel ride etc.http plugins to call apis,flutter polyline points to to draw line to show from source to destination,smooth star ratings for rating page,animated text kit for animated text in the map, geofire for geo loactions of the drivers and users.

```
cupertino_icons: ^1.0.2
firebase_core: ^1.11.0
firebase_auth: ^3.3.5
firebase_database: ^9.0.5
fluttertoast: ^8.0.8
google_maps_flutter: ^2.1.1
geolocator: ^8.0.5
http: ^0.13.4
provider: ^6.0.2
flutter_polyline_points: ^1.0.0
flutter_geofire: ^2.0.1
smooth_star_rating_nsafe: ^1.0.0+1
animated_text_kit: ^4.2.2
```

Figure 6.13: All plugins

Now in the drivers app we can See at screen S13 (refer appendix figure 16) there is a screen which shows all the details about the drivers ,all the information is stored in the firebase and its shown in profile page such as name, care model ,car name etc to the drivers when it click profile tab.

Screen S14 (refer appendix figure 16) is the Rating Screen which shows the drivers got the ratings in the last ride and all history of ride,after end of trip after payment user get a page where he can give ratings to the driver and it reflect to driver screen S14 (refer appendix figure 16).

Screen S15 (refer appendix figure 16) is the earnings screen which shows about total earnings driver done with all the ride he done till date with the history of all the ride and its earning, generally the price is calculated on the basis of per km distance and fuel

consummations.

The distance can be calculated using the following formula (10) :-

$$Letp = \pi/180.0$$

$$And, a = 0.5-cos((lat2-lat1)*p)/2+cos(lat1*p)*cos(lat2*p)*(1-cos((lon2-lon1)*p))/2$$

$$Distance = (12742 * asin(sqrt(a)))Km$$

For user also there is navigation button at the top where user can found out its own details about the app, logout button if user press logout button he need to login in back. for drivers also in the account page there is button name logout where user can logout and if he wish can login with same account or some other account.

So, we can say by using this app we can easily book a ride with a simple steps to follow and enjoy the ride,In India a app like this is more in use for transportation and there is need of more improvement in this field so this app will full fill some gap like it will able to solve some of the major problem of cost,fuel exemption, time and safety concern which is the major concern in country like India.This app is developed with its first module which is ride sharing and its ready to deploy on the play store and use.

6.7 TESTING AND DEBUGGING THE APP

In this project we have done some of the basic type of testing needed for this project mostly manually not on any simulations. We have done researched and work on one of the testing simulation tools for futher which we will use it in upcoming days for testing our project this testing process is done in three phases one is unit test, widgets testing and it goes to integration testing.

6.7.1 Unit Testing

Unit test is the small part of our app it is represented using functions ,methods or a class of our code.In the field of development Test-driven development (TDD)(11) is considered as best approach for writing maintainable and clean code.

6.7.2 Widgets Testing

widgets is used to test the widgets of the codes weather it is true or not,this type of testing is more complex than unit testing because it test all widgets and in flutter every thing is written in the form of widgets. Fluttertest package using this, we can use different tools for testing our widgets of the app, such as building the widgets and interacting the widgets and using the widgets specific constants it will help to locate one or more widgets.

Integration is very much different than unit or widgets testing, it name suggest that integration testing thus it allows us to test individual part of our app ,integration testing is costly and complex ,but it help to test each part of our app. This all testing will be perform in the next phase ,for each type of testing we had to write some piece of code and apply it, but after test we can easily granted than it is more reliable with large platform use.

CHAPTER 7

CONCLUSION

In summary, the use of ride-sharing and efficient cargo loading and unloading software can contribute to a more sustainable future in transportation. This project involves the development of an android and IOS ride-sharing and cargo planner app using flutter, which has user-friendly interfaces for both drivers and users. The app uses a ride-sharing algorithm on the server end to allocate online drivers to users for ride requests and has features that make it flexible, reliable, and comfortable for drivers to use. The app also offers cargo transportation services with a 3D visualization of efficient cargo loading and unloading software to make it safer, more reliable, and affordable for users. Overall, this project offers a practical solution for reducing transportation costs and increasing sustainability.

CHAPTER 8

FUTURE WORK

Future work on the ride-sharing and cargo planner app can include:

Integration of payment gateways: Currently, the app may store user and driver information, but it may not have a payment system in place. Integrating payment gateways can allow users to make online payments for their ride-sharing or cargo transportation services, making the app more convenient and user-friendly.

Implementing advanced algorithms for ride-sharing: While the current app may have a basic algorithm for ride-sharing, future work can involve implementing more advanced algorithms that take into account factors such as real-time traffic conditions, passenger preferences, and multiple destinations, to optimize ride-sharing routes and improve the overall efficiency of the system.

Adding additional features for drivers: The driver app can be enhanced with additional features such as earnings tracking, trip history, and feedback/rating system for passengers, to provide a better experience for drivers and incentivize their participation in the ride-sharing or cargo transportation service.

Enhancing the 3D Visualization for cargo loading: The 3D Visualization feature can be further improved by incorporating more realistic cargo loading scenarios and providing real-time feedback to users on the efficiency of their cargo loading process. This can help users optimize their cargo loading strategies and reduce wastage.

Expanding to other markets: The app can be expanded to other markets or regions to provide ride-sharing and cargo transportation services in areas where these services are in high demand but may not be readily available. This can help promote sustainable

transportation options in more regions and contribute to reducing the overall environmental impact of transportation.

Incorporating sustainability metrics: Future work can involve incorporating sustainability metrics, such as carbon footprint calculations or emissions tracking, into the app to provide users with information about the environmental impact of their rides or cargo transportation. This can help raise awareness about sustainability and encourage users to choose more environmentally friendly options.

Improving user experience: Continuously improving the user experience of the app, including the user interface, app performance, and responsiveness, can contribute to enhancing overall user satisfaction and increasing user engagement with the app.

.1 APPENDIX

The Complete UI flow and Screen shots of the Screen for both the Apps.

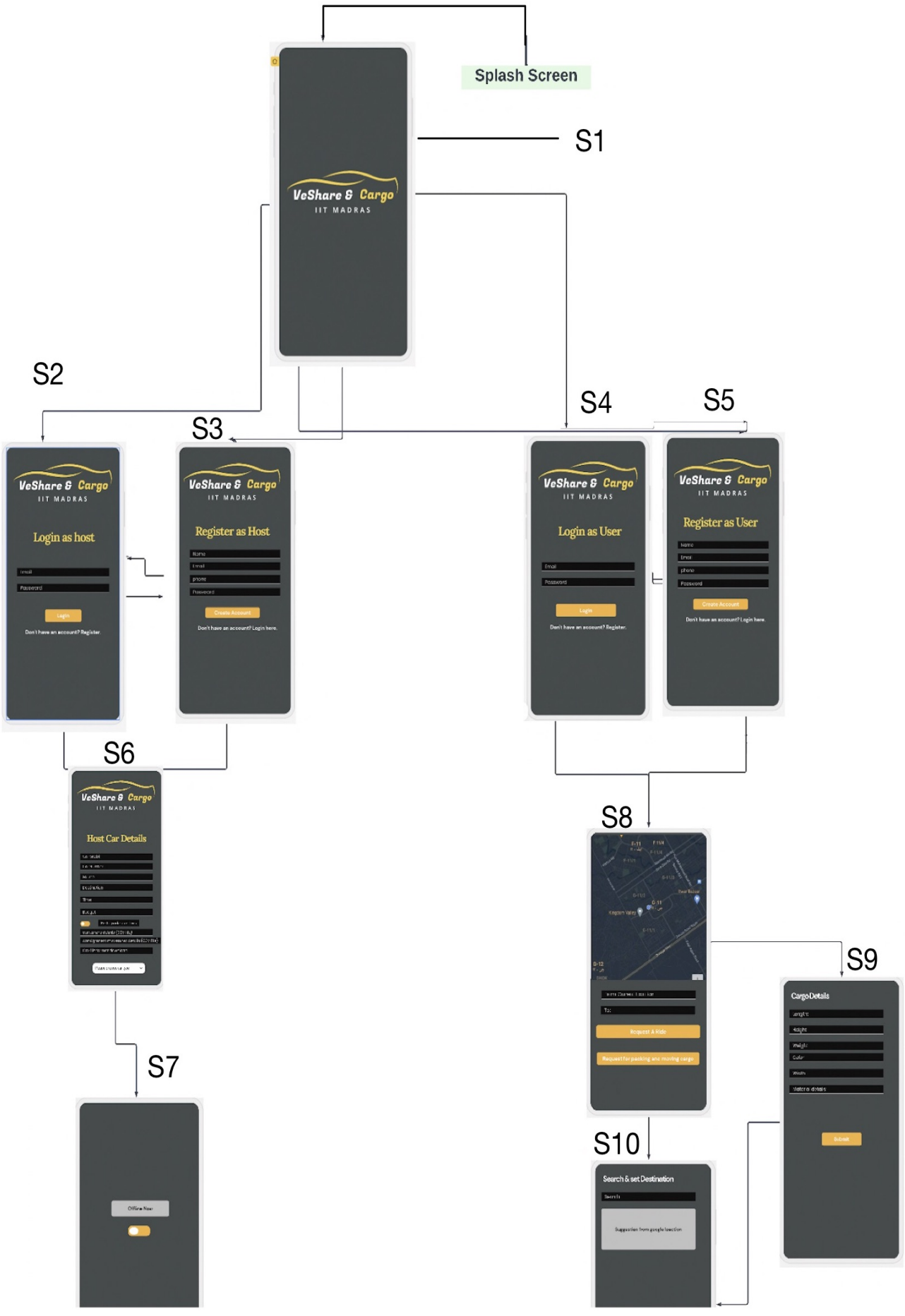


Figure .1: VeShare and Cargo Planner App(UI FlowChart).

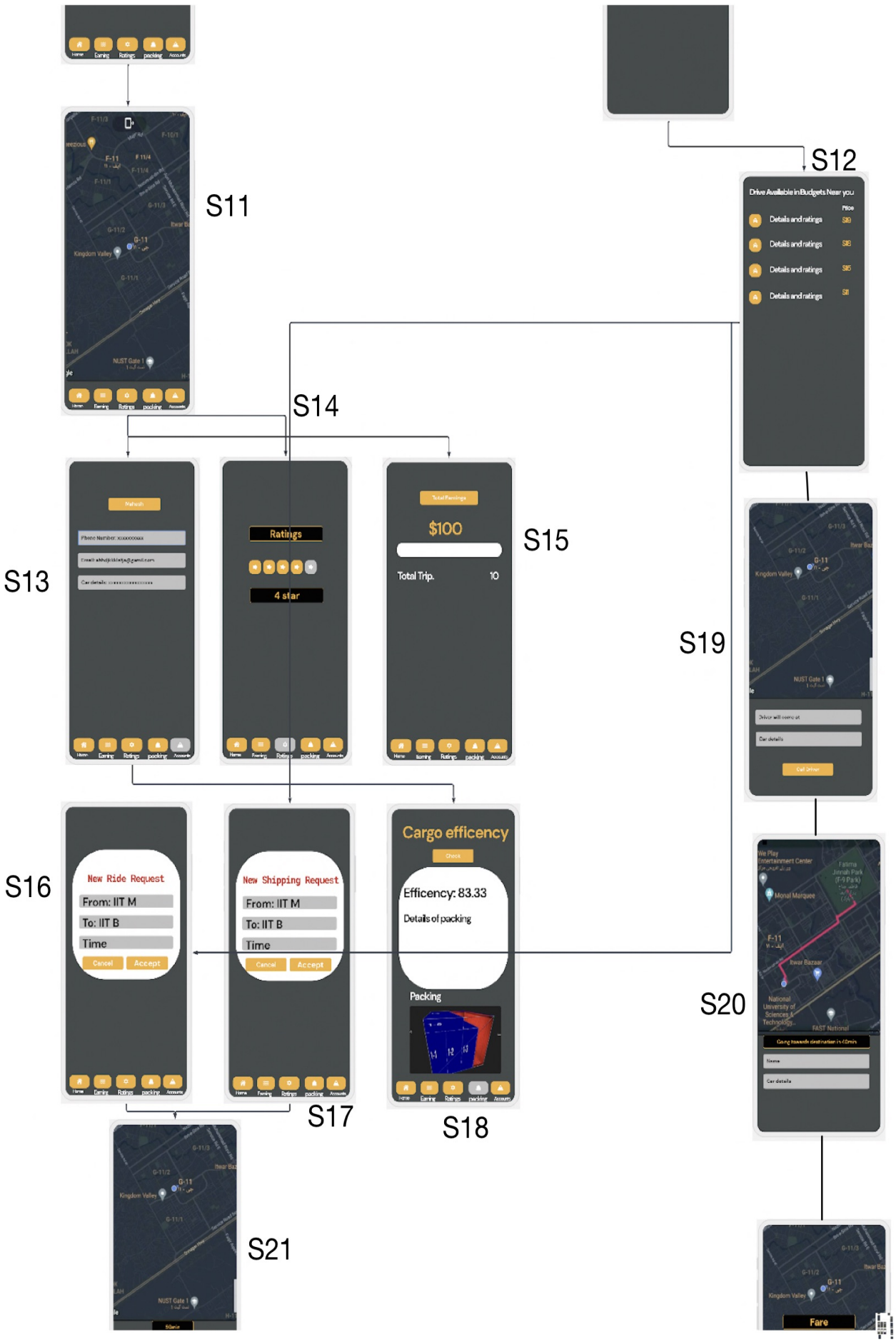
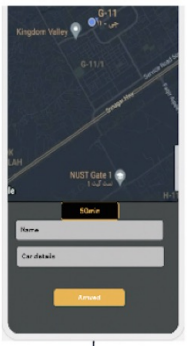
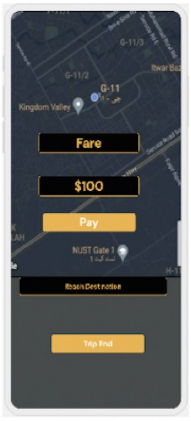
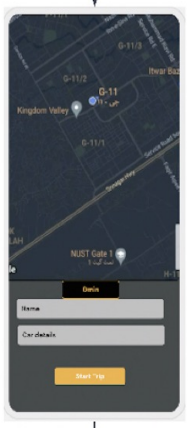
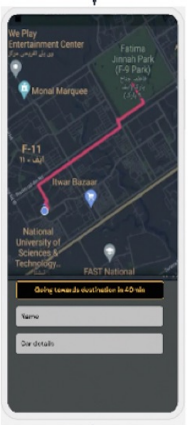


Figure .2: VeShare and Cargo Planner App(UI FlowChart).









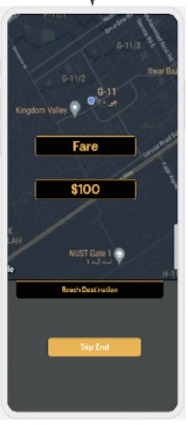


Figure .3: VeShare and Cargo Planner App.

REFERENCES

[1] The Dart Programming Language by Gilad Bracha,Publisher, 1st edition (10 December 2015).

[2] Flutter in Action, 1st Edition,By Eric Windill,january 2020.

[3] Google maps Apis 3 by Gabriel svennerberg, 1st Edition, 27 July 2010

[4] Open Source routing machine (OSRM),Modern C++ routing engine for shortest paths in road networks. https://project-osrm.org/docs/v5.24.0/api/#

[5] Firebase documentations published by goolge https://firebase.google.com/docs

[6] Google Cloud documentation https://cloud.google.com/docs

[7] This article is intended to provide a high-level overview of the architecture of Flutter, including the core principles and concepts that form its design https://docs.flutter.dev/resources/architectural-overview

[8] This is the MySQL Workbench Reference Manual. It documents the MySQL Workbench Community and MySQL Workbench Commercial releases for versions 8.0 through 8.0.31.Document generated on: 2022-10-26 (revision: 74410) https://dev.mysql.com/doc/workbench/en/

[9] Flutter Favorites Some of the packages that demonstrate the highest levels of quality, selected by the Flutter Ecosystem Committee https://pub.dev/

[10] Haversine Formula,by Gabriele De Luca https://www.baeldung.com/cs/haversine-formula

[11] Testing Flutter apps https://docs.flutter.dev/testing

[12] SimPy Discrete event simulation for Python. SimPy is a process-based discrete-event simulation framework based on standard Python https://simpy.readthedocs.io/en/latest/

[13] https://threejs.org/docs/

[14] National Portal of Inidia NITI Ayog https://www.niti.gov.in/

[15] https://etrr.springeropen.com/articles/10.1186/s12544-021-00522-1

[16] Visualize and communicate your ideas effortlessly. https://app.uizard.io/

[17] Ride Sharing app previous work https://drive.google.com/file/d/1__wwTdUuTuoaI8opzIk2odFNtRuEOFxQ/view?usp=sharing

[18] Cargo Loading Previous work. https://github.com/VikramManjare70/clpBasic

[19] Cargo Loading Previous work. https://github.com/Project-OSRM/osrm-backend

https://github.com/Project-OSRM/osrm-backend