Bilkent University

Department of Computer Engineering

Senior Design Project

Project name: Wheelancer

Final Report

Onat Korkmaz	21704028
Muharrem Berk Yıldız	21802492
Muhammed Maruf Şatır	21702908
Ümit Çivi	21703064
Erdem Ege Eroğlu	21601636

Supervisor: Fazlı Can
Jury Members: Erhan Dolak, Tağmaç Topal
Innovation Expert: Murat Ergun

May 6, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

The shipping sector has developed significantly in the last decades due to the advancements in transportation and other technologies such as e-commercing. All of these advancements led to a dramatic increase in the amount of cargo that has been shipped by individuals and companies. In order to tackle this huge demand, many companies that focus on the delivery of cargo have been founded throughout the last decades. Nevertheless, the abundance of the cargo companies caused some problems. For example, people are struggling to choose a trustworthy cargo company that will not harm their goods. Moreover, the disappearance of their cargo makes people hesitate to utilize cargo services. There are also well-known companies such as UPS that have accomplished billions of package deliveries in 2018[1]. Although these companies can deliver goods without a problem, the cost can be expensive even if you want to send a small package. Last but not least, in the pandemic, demand for cargo shipment has increased due to the fact that people started to use e-shopping more frequently. Moreover, the pandemic caused many people to lose their jobs. Some of these people who own a vehicle started to use apps like Uber, BlaBlaCar to gain the money they need to sustain a healthy life. Therefore, in this project, we aim to design an application that will retain the good qualities of cargo companies while reducing the price for the transportation and help people by enabling them to have another source of income in the pandemic.

With this final report, we will provide the functional and non-functional requirements to give a brief overview of the application. Then, we aim to explain the final architecture, design and implementation details and testing details. While doing this, we will also consider the new knowledge we have acquired during the whole year. Other than these technical issues, we will also consider various ethical, social, economical issues. At the end a user manual and a section that describes our future plan, will be given.

2. Requirements Details

2.1 Functional Requirements

In our application users are either customers or couriers.

2.1.1 Common functionalities

User is able to;

- Register to the system using his TC identity number
- Verify the account using email verification
- Login to the system using email and password
- Logout from the system

2.1.2 Customer functionalities

Customer is able to;

- See their current package list
- Create packages by entering the necessary information such as length, width etc. and upload a photo for later usage for verification
- See the detailed info page for a specific package
- Delete the cargo if the cargo is not assigned to a courier
- See the last known location of the package if it is assigned to a courier
- Rate the courier if the delivery is completed
- See the verification of the delivery as a photo after the delivery is completed
- See offers made to a package by a specific courier
- See an offer in a detailed manner i.e courier details etc.
- Accepting or rejecting offers made by couriers for a specific package
- Increase your in-app balance by a specified amount

2.1.3 Courier functionalities

Courier is able to;

- Upload driver license to the application for verification
- Upload car information such as capacity to the application for later usage
- See their current transportations
- See the packages inside a specific transportation by clicking on it
- Create new transportation by specifying the vehicle, starting location and time
- Add new packages to the transportation if there is capacity in the vehicle
- Make offers to the packages that can be carried
- See the detailed route of the package before sending offer
- Confirm the pick-up of package if the offer is accepted

- See the route of the package while delivering it
- Finish the delivery by uploading a photo of package when delivered

2.2 Non-functional Requirements

2.2.1 Accessibility

● Since the new and useful features such as “turn by turn navigation”, Android Marshmallow 6.1 or newer version is required in order for users to run our system. Node.js was used for the REST API [2].

2.2.2 Accuracy

● The application will ask for delivery specifications. Particularly, if there are fragile items in the cargo, drivers can be informed about the cargo box in advance.

2.2.3 Availability

● The application uses the GPS signals of drivers, so even if he/she does not keep the internet connection open, the client can be informed about the location of the cargo.
● Our initial audience will be Turkey, thus, bug fixes and updates should occur at midnight. So that it is aimed to protect our users from inaccessibility to the system.

2.2.4 Backup and Recovery

● Our database will be MySQL. Therefore, MySQL undertakes backup and recovery of the data.
● Personal data and cargo preferences should be stored in local and global storage so as to any data will not lost.

2.2.5 Extensibility

● It is open to developments on the basis of new features and new functionalities. Such as:

Payment methods,
Security precautions,
Chat.

2.2.7 Reliability

● Users need to sign the user agreement in order to provide application reliability. Just in case, their information may be shared with the court or police.

● Any crush on account of software, should not occur in the system.

2.2.8 Security

● The users must sign up with their private credentials.
● The user passwords are hashed before receiving from user
● The users are given JSON Web Token to provide authentication and authorization for user in a session lifetime [3].
● The application provides the integrity of the customer's private information.

2.2.9 Testing

● The web server, mobile application and database will be tested regularly to eliminate errors or any mistakes. Some sort of test:
    ○ Load tests to measure performance according to actual user behaviors.
    ○ Reliability tests will be made to control the web server and function properly.

2.2.10 Usability

● The users are able to give their suggestions as feedback to developers.
● The user will not spend time learning the functions of the application. The GUI
will be intuitive and user friendly so user interactions are easy and obvious.

3. Final Architecture and Design Details

As we mentioned in our previous reports we used client server architecture.

3.1 Subsystem Decomposition

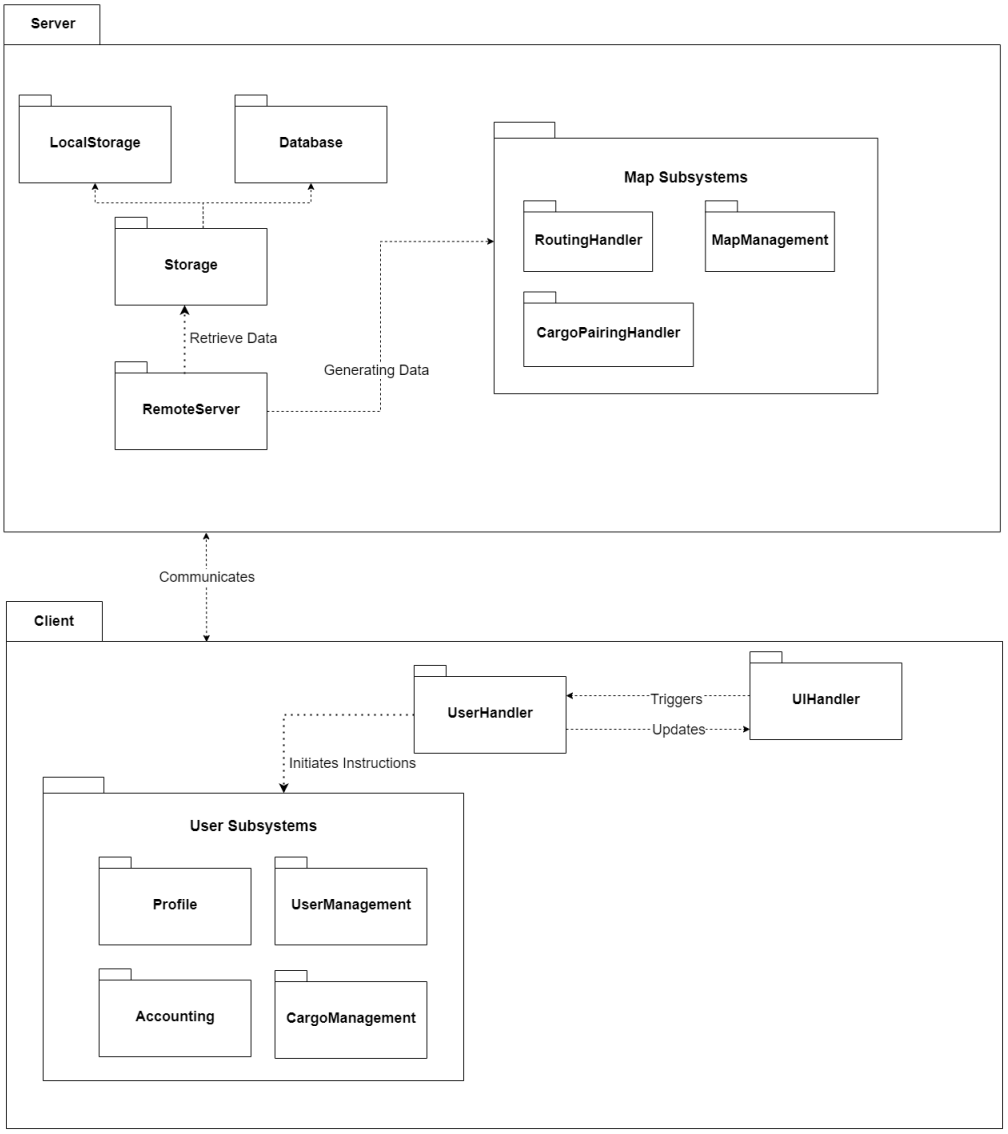


Fig. 1 Subsystem Decomposition Diagram

3.2 Class Interfaces

3.1.1 Courier
3.1.2 Courier UI

CourierLoginUI
The page where the courierwill login to the application
Attributes
+ box: Box - A storage that holds user sensitive data. + emailTextController: TextController - A controller for the text field of email. + password: TextController - A controller for the text field of password.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + login(email:String, password:String): Future - The function that calls the corresponding API function for login. Future is a class in Dart language that is used for asynchronous computation. It means that the return value of the function will be calculated or an error will be returned. + showAlertDialog(context:BuildContext): Widget - A function that returns an alert dialog widget to the login screen if login fails.

CourierSignUpUI
The page where the courierwill signup to the application
Attributes
+ box: Box - A storage that holds user sensitive data. + emailTextController: TextController - A controller for the text field of email. + passwordTextController: TextController - A controller for the text field of password. + nameTextController: TextController - A controller for text field of name. + surnameTextController: TextController - A controller for the text field of surname. + tcNumTextController: TextController - A controller for text field of. TC identity no
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + signup(email: String, password: String, name: String, surname: String, phoneNumber: String): Future - The function that calls the corresponding API function for signup. + showAlertDialog(context: BuildContext): Widget - A function that returns an alert dialog widget to the signup screen if login fails.

CourierMainMenuUI
The page where the courier will see his/her transportations.
Attributes
+ box:Box - A storage that holds user sensitive data.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getAllTransortations(token: String): Future - The function that calls the Wheelancer API which fetches all of the transportation datas to corresponding to the user token.

ProfileUI
The page where couriers can see the comments, upload driver license.
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getCourierComments(token:String): Future, This function returns the comments made by customers on specified courier. + getFromCamera(): returns photo image from camera. + uploadCourierLicense(token:String, file:ImageFile): Future, This function uploads the image file on specified courier.

AddVehicleUI
Couriers can add the informations vehiclerelated.
Attributes
+ box: Box - A storage that holds user sensitive data. + brandCont: TextController: Controller for taking the brand. + brandCont: TextController: Controller for taking the model of the vehicle. + lengthCont: TextController: Controller for taking the length of the vehicle. + widthCont: TextController: Controller for taking the width of the vehicle. + heightCont: TextController: Controller for taking the height of the vehicle. + weightCont: TextController: Controller for taking the weight of the vehicle. + horsepowerCont: TextController: Controller for taking the horsepower of the vehicle.

+ registrationplateCont: TextController: Controller for taking the registration of the vehicle.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + addVehicle(token:String, vehicleType: String, brand: String, model: String, length: double, width: double, height: double, weight: double, horsepower:int, registrationplate: String): Future, This function adds vehicle to the specified courier. + showAlertDialog(context: BuildContext, error: String): Shows corresponding message.

TransportationSelectVehicle
Couriers must select vehicle which were added before for selecting transportation.
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getAllVehicles(token:String): Future, This function returns all the informations related the vehicles of the specified courier.

AddTransportationInfosUI
Couriers must select the departure date and time and arrival date and time for adding transportation.
Attributes
+ box: Box - A storage that holds user sensitive data. + departureTime: TimeOfDay - Departure time for this transportation. + arrivalTime: TimeOfDay - Arrival time for this transportation. + departureDate: DateTime - Departure date for this transportation. + arrivalDate: DateTime - Arrival date for this transportation. + pos: Position - Position for courier.
Methods

+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + addTransportation(token:String, currentVehicleId: int, lat: double, lng: double, departureDateTime: String, arrivalDateTime: String): Future, This function adds transportation to specified courier. + showAlertDialog(context: BuildContext, error: String): Shows corresponding message. + determinePosition(): Future, gets the current position of the user. + selectTime(context: BuildContext): Future, provide the ui for selecting time. + selectDate(context: BuildContext): Future, provide the ui for selecting date.

PackagesOffersCourierUI
Couriers can see the offers and current packages of the user and control the functionalities of the packages based on their status.
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getCourierTransportationPackages(token:String, transportationID: int): Future, This function gets all the packages for specified courier in selected transportation. + returnTrailing(status: String, pid: int, transportationID: int, courierID: int, token: String, context: BuildContext, box: Box, s_long: double, s_lat: double, d_long: double, d_lat: double):Widget - If the status of the cargo is negotiated, there will be two button which are reject and showing route for taking the cargo. If the status is picked up, then there will be two buttons for showing the route to destination and submitting the delivery.

GetNearPackagesUI
Couriers must select the city and radius in selected transportation to see the packages that are created on offer.
Attributes
+ box: Box - A storage that holds user sensitive data. + selectedCity: String - Dropdown value for selecting the city. + pos: Position - Position of the values. + radiusController: TextController - Takes the radius value from the courier.
Methods

+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked.
+ getNearPackages(token:String, selectedCity: String, lat: double, long:double, radius: String): Future, takes the cargos that is matched with courier preferences.
+ showAlertDialog(context: BuildContext, error: String): Shows corresponding message.
+ determinePosition(): Future, gets the current position of the user.

CourierNearPackagesUI
Couriers can see the packages near to the courier and by selecting the cargo can see details about package and make offer.
Attributes
+ box: Box - A storage that holds user sensitive data. + packages: Map<String, dynamic>: Packages that are near to the courier.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + showAlertDialog(context: BuildContext, error: String): Shows corresponding message.

MakeOfferUI
Couriers can see the details about packages. There are two buttons. One of them shows the package starting point and destination point on the map. Other can make offer with specified price.
Attributes
+ box: Box - A storage that holds user sensitive data. + package: Map<String, dynamic>: All the informations related to the cargos. + amountController: TextController: The amount field that courier will offer the specified cargo.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + showAlertDialog(context: BuildContext, error: String): Shows corresponding message. + getFirstPhoto(token:String, package_id: int): Future, This function gets the first photo which is first uploaded by the customer on a specified package. + returnRemainingWidgets(context: BuildContext, status: String, packageID: int, token:String, box: Box, transport_id: int): Widget, This function returns the buttons

of checking source and destination of package in map and making offer to the package. + makeOffer(token: String, package_id: int, transport_id: int, amount: double): Future, This function makes the offer with specified amount to the specified cargo.

3.1.2 Client

3.1.1.1 Customer UI

HomePageUI
HomePage is a class where customers or couriers are able to choose their role as either courier or customer.
Attributes
+ box:Box - A storage that holds user sensitive data.
Methods
+ build(context:BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. BuildContext context is a handle to the location of the widget in the widget tree.

CustomerLoginUI
The page where the customer will login to the application
Attributes
+ box: Box - A storage that holds user sensitive data. + emailTextController: TextController - A controller for the text field of email. + password: TextController - A controller for the text field of password.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + login(email:String, password:String): Future - The function that calls the corresponding API function for login. Future is a class in Dart language that is used for asynchronous computation. It means that the return value of the function will be calculated or an error will be returned. + showAlertDialog(context:BuildContext): Widget - A function that returns an alert dialog widget to the login screen if login fails.

CustomerSignUpUI
The page where the customer will signup to the application
Attributes
+ box: Box - A storage that holds user sensitive data. + emailTextController: TextController - A controller for the text field of email. + passwordTextController: TextController - A controller for the text field of password. + nameTextController: TextController - A controller for text field of name. + surnameTextController: TextController - A controller for the text field of surname. + tcNumTextController: TextController - A controller for text field of. TC identity no
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + signup(email: String, password: String, name: String, surname: String, phoneNumber: String): Future - The function that calls the corresponding API function for signup. + showAlertDialog(context: BuildContext): Widget - A function that returns an alert dialog widget to the signup screen if login fails.

CustomerMainMenuUI
The page where the customer will see his/her cargo
Attributes
+ box:Box - A storage that holds user sensitive data.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getAllPackages(token: String): Future - The function that calls the Wheelancer API which fetches the cargo information corresponding to the user token.

CreateCargoMenuUI
The page where a user will create a new cargo order.
Attributes
+ box: Box - A storage that holds user sensitive data. + startPointTextController: TextController - A controller for the text field of starting point. + destinationTextController: TextController - A controller for the text field of destination. + startCityController: TextController - A controller for the text field of starting city + destCityController: TextController - A controller for the text field of starting city + widthTextController: TextController - A controller for the text field of width. + heightTextController: TextController - A controller for the text field of height. + lengthTextController: TextController - A controller for the text field of length. + weightTextController: TextController - A controller for the text field of weight. + recTextController: TextController - A controller for the text field of receiver. + base64Image:String - A variable for holding the base64 encoding of the image taken by the user while creating the cargo + picker:ImagePicker - A variable that is used for opening the camera
Methods
+ build(context: BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + addCargo(token: String, startPoint: String, dest: String, width: float, height: float, length: float, weight:float, description:String, startCity:String, destCity: String, base64Image:Image): Future - Calls the API function that adds the specified cargo to the database + showAlertDialog(context:BuildContext): Widget, A function that returns an alert dialog widget to the signup screen if cargo submission fails.

CargoDetailUI
The page where customers can find detailed information about the cargo.
Attributes
+ box:Box - A storage that holds user sensitive data. + data_about_package: Map<String,dynamic> - The variable used to store information passed from customer main menu regarding a specific package
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getCargoDetail(token:String, cargoID:int):Future, Gets the details of the cargo corresponding to the specified ID via calling the API function. + getRemainingButton(status:String):Widget, Gets the remaining buttons for the screen depending on the status of the package

ProfileUI
The page where customers can see and change their profile information.
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context:BuildContext):Widget, This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getProfileInfo(token:String): Future, This function returns the information about the user

HelpUI
The page where customers can access the user manual and credits.
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked.

OffersUI
The page where the customer can see the offers made to a package owned by him/her
Attributes
+ box: Box - A storage that holds user sensitive data.
Methods
+ build(context: BuildContext): Widget - This function returns a widget that will be attached to the Widget subtree where this method is invoked. + getOffers(token:String): Future - This function returns all the offers that are made to the user specified with the token

DetailedOfferScreen
The screen where customer can see the details of a specific offer.
Attributes
+ box:Box - A storage that holds user sensitive data.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method + affectOffer(token: String, offerID: int, date: String,): Future - the function that calls the API function to accept an offer + rejectOffer(token: String, offerID: int, date: String,): Future - the function that calls the API function to reject an offer

MapUI
The page where the customer will be able to see the current location of the cargo which is being delivered.
Attributes
+ box: Box - A storage that holds user sensitive data. + timer:Timer - A timer that is used for calling the getLocation method periodically. + center:LatLng - The locations of the coordinates where map is focused + int milisec - The value used for starting the timer
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method + getLocation(orderID: int): Future - Calls the API function to fetch the last known location of the cargo

3.1.1.2 Courier UI

HomePage
HomePage is a class where customers or couriers are able to choose their role as either courier or customer.
Attributes
+ box: Box - A storage that holds user sensitive data

Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method

SignUpPageCourier
SignUpPageCourier is a class where couriers can sign up from this page through their email, phone number, password, license, name and surname.
Attributes
+ box: Box - A storage that holds user sensitive data + emailTextController: TextController - A controller for the text field of email. + passwordTextController: TextController - A controller for the text field of password. + nameTextController: TextController - A controller for text field of name. + surnameTextController: TextController - A controller for the text field of surname. + tcNumTextController: TextController - A controller for text field of. TC identity no
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + TextField({label, obscurity}): Widget - It returns a text field widget. Obscurity is differentiated between password text fields and other text fields. + signup(email: String, password: String, name: String, surname: String, phoneNumber: String): Future - The function that calls the corresponding API function for signup. + showAlertDialog(context: BuildContext): Widget - A function that returns an alert dialog widget to the signup screen if login fails.

LoginPageCourier
LoginPageCourier class where couriers can login with their email and password to their accounts.
Attributes
+ box: Box - A storage that holds user sensitive data + emailTextController: TextController - A controller for the text field of email. + password: TextController - A controller for the text field of password.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + login(email:String, password:String): Future - The function that calls the corresponding API function for login. Future is a class in Dart language that is used for asynchronous computation. It means that the return value of the function will be calculated or an error will be returned.

+ showAlertDialog(context:BuildContext): Widget - A function that returns an alert dialog widget to the login screen if login fails.

CourierMainMenu
SignUpPageCourier is a class where couriers can see their main screen. In the main screen couriers can see the matched cargos and can connect with the customer.
Attributes
+ box: Box - A storage for user information + cards: List<Card> - It contains fetched cargos which match with preferences of courier from API.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + ProgressStatefull(): Widget - It returns a widget for couriers to connect with the owner of the cargos. Widget also shows the status with the connection with the owner of the cargo with different colors. + buildAppBar(context: BuildContext): void - Builds the basic app bar for the courier and from the appbar courier can navigate to other pages. + ListView.builder(context: BuildContext, index: var): void - It displays cards of the cargo on the screen.

CourierProfileUI
CourierProfileUI is a class where couriers can see their profile page. It contains the badgets, scores from the customers. Couriers also can change their information on this page.
Attributes
+ box: Box - A storage for user information + profileImage: Image - Image file that is uploaded by the courier. The file that is displayed is fetched from the controller class and stored at local while the user opens the courier profile.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + createCheckboxElements(): void - It creates elements for the checkbox by calling checkbox.createElement() method. Courier can indicate which materials s/he can take on his/her vehicle.

DeliveryMenu
DeliveryMenu is a class where couriers can validate cargo deliveration by uploading photo.
Attributes
+ box: Box - A storage for user information + delivery: Image - An image file that is to verify the delivery of the cargo to the destination location. This file is kept in local storage before asserting the fully delivery.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + StatefulFilePicker(): Widget - It returns a widget for couriers to pick the delivery photo.

MapRouteUI
MapRouteUI is a class where couriers can see their location and route to destination location.
Attributes
+ box: Box - A storage for user information + accessToken: String - Access token to access map API. + mapcontroller: MapController - Stateful controller for flutter map to manage markers, lines and polygons.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method. + initState(): void - Initializes the state variables of the class. + getCurrentLocation() async: Marker - It returns a marker by retrieving information of current location. + setState(): void - User can move the map and center changes by the movement.

3.2.2 Server

3.2.1.1 REST API

3.2.1.1.1 MapManager

Transport
A session of a delivery maintained by backend
Attributes
- transport: Transport - Transport that is willing to take packages. - courier: Courier - Courier that is responsible for a single transport operation. - vehicle: Vehicle - Vehicle responsible for transport operation. - long: double - Longitude of the current location of the courier. - lat: double - Latitude of the current location of the courier. - last_update_date: Date - The date and time of the last modification in the delivery record. - remaining_weight: double - Remaining amount of weight that the vehicle can take for the delivery. - remaining_volume: double - Remaining amount of space that the vehicle can take for the delivery. - currentPosition: Node - Current position of courier carrying the cargo. - departureDate: Date - Departure date of the transport. - arrivalDate: Date - Arrival date of the transport. - status: Enum - Status of transport (ready, ongoing, finished).
Methods
+ updateCourierPosition(courier: Courier, long : decimal, lat: decimal): JSON- Updates the courier coordinates for the package owners. + getAllPackagesInCity(city: String): JSON - Shows a list of packages in the departure city. + getCustomerOffers(customer: Customer): JSON - Show list of attributes for the pending offers, visible features for the customers. + getCourierOffers(courier: Courier): JSON - Show list of attributes for the pending offers, visible features for the couriers. + removeUnwantedOffersExceptGiven(package: Package, courier: Courier, transport: Transport): JSON - Deletes all other offers for the package after an offer is accepted. + upIncrRemainingValue(transport: Transport, volume: double, weight: double): JSON - Updates remaining space and weight of the vehicle after a package is added. + cancelOffer(package: Package, courier: Courier, transport: Transport): JSON - Removes the pending offer from the system. + getTransportFromCustomerPackage(customer: Customer, package: Package): JSON - Shows the delivery information that the customer's package is in. + getOfferDetails(package: Package, courier: Courier, transport: Transport): JSON - Shows all information about the offer made on the package. + addOffer(package: Package, courier: Courier, transport: Transport, price: double): JSON - Creates an offer made for a specific package and a delivery in the system. + addTransport(courier: Courier, vehicle: Vehicle, long: double, lat: double, last_update_date: Date, remaining_volume: double, remaining_weight: double, departure_date: Date, arrival_date: Date, status: String): JSON - Creates a record for a potential delivery.

+ getCourierTransports(courier: Courier): JSON - Shows a list of deliveries of a courier with the vehicle information.

3.2.1.1.2 AccountingManager

Customer
Class for representing cargo owners.
Attributes
+ customer: Customer - The cargo owner.
Methods

Admin
Class for representing administrators.
Attributes
+ admin: Admin - The administrator.
Methods

User <<Interface>>
User interface for Customers, Couriers and Admin classes.
Attributes
+ user: User - The instance of the user. + name: String - Name of the user. + surname: String - Surname of the user. + birthday: Date - Birthday of the user. + idNum: String - The identification number of the user. + balance: double - The money balance of the user inside the app. + email - String - the e-mail address of the user. + password: String - Password of the user (encrypted) + email_verified: int - Indicator of the verification of the user. + registration_date: Date - The record for the date that the user first registered.

+ type: int - Type of the user (Admin, Customer, Courier)

Methods

+ addUser(name: String, surname: String, birthday: Date, idNum: String, email: String, password: String): JSON - Creates a user in the system.
+ addAdmin(user: User): JSON - Creates an administrator account.
+ addCourier(user: User): JSON - Creates a courier account.
+ addCustomer(user: User): JSON - Creates a customer account.
+ addVehicle(user: User, model: String, brand: String, max_length: double, max_width: double, max_height: double, max_weight: double, horsepower: int, registration_plate: Date, vehicle_type: String): JSON - Creates a vehicle.
+ getVehicleFromTransport(user: User, transport: Transport): JSON - Shows the vehicle used for the delivery.
+ getVehicle(user: User, vehicle: Vehicle): JSON - Shows the vehicle information.
+ getVehicles(user: User): JSON - Shows all vehicles of the courier.
+ addDocument(user: User, document: image): Adds the courier's drivers license.
+ giveFeedback(customer: Customer, courier: Courier, package: Package, message: String, rate: double): JSON - Saves the rating and the feedback message for the courier.
+ getMyFeedbacks(courier: Courier): JSON - Displays all of the feedback for a specific courier.
+ autoUpdateAverageScore(courier: Courier): JSON - Automatically updates the average rating of the courier.
+ getMyDocument(user: User): JSON - Displays all of the documents for a specified user.
+ verifyCourier(user: User, courier: Courier): JSON - Allows the admin to verify the courier based on its document and information.
+ getUnverifiedCouriers():JSON - Displays all of the couriers who are nor verified.
+ getVerifiedCouriers(): JSON - Shows the list of verified couriers.
+ setBiography(user: User, biography: String): JSON - person can create a biography of himself.
+ setEmail(user: User, email: String): JSON - Updates the e-mail address of the user.
+ setName(user: User, name: String): JSON - Updates the name of the user.
+ setPassword(user: User, password: String): JSON - user can change his/ her password
+ updateBalance(user: User, amount: double): JSON - Updates the balance of the user.
+ setUsername(user: User, username: String): JSON - Updates the username of the user.
+ createVerifiationCode(user: User, expireDate: Date, code: String): JSON - Creates a verification code for the user.
+ listVerificationCodes(user: User):JSON - Displays all of the verifications of a specified user
+ verifyEmail(user: User): JSON - change the email status to make it verified for a single user.
+ loginUser(email: String, password: String): JSON - Lets the user login to the

system.
+ getUser(user: User): JSON - Shows the user information.
+ checkUserType(user: User): JSON - display the type of the user for a single user.
+ checkAuth(authCode: String): JSON - Checks authentication token validity.
+ getUserInfo(user: User): JSON - displays information about the user according to specified user id.
+ checkAuthAdmin(authCode: String, user: User): JSON - Checks authentication token for the admin.
+ checkAuthCourier(authCode: String, user: User): JSON - Checks authentication token for the courier.
+ addAuth(authCode: String, user: User): JSON - Creates authentication token.

3.2.1.1.3 CargoPairingHandler

Courier
Courier class that represent users responsible for deliveries.
Attributes
+ deliveries: ArrayList<Transport> - List of deliveries that are assigned to a courier. + feedbacks: ArrayList<Feedback> - List of given feedbacks to the courier. + score: float - Total score of the courier about prior operations. + isVerified: bool - Verification status of courier.
Methods
+ addUser(name: String, surname: String, birthday: Date, idNum: String,): double - Return current balance of a user.

Vehicle
Class representation of a vehicle used in transports.
Attributes
+ model: String - Model of a car. + brand: String - Brand of a car. + cargoSize: double - Total cargo size that vehicle is capable of. + horsePower: double - HP of a vehicle. + fuelConsumptionRate: float - Fuel consumption rate of the vehicle for calculation cost.
Methods
+ calculateCost(routes: ArrayList<Route>): void - Calculates recommended cost.

DeliveryGood
Class represenation goods that are transported by Couriers.
Attributes
+ size: double - Size of total volume the good. + weight: double - Weight of the good. + destinationPoint: Node - Destination point for the good. + departurePoint: Node - Departure point for the good. + photoUrl: String - Photo url that is taken of the good. + status: String - Status of the good.
Methods
+ getRoute(): Route - Returns routes that are assigned to the good.

Report
Class representation of the reports provided by users.
Attributes
+ type: String - Type of the document (Driver License). + description: String - Description of the document. + reportDate: Date - Submission date of the document. + photoUrls: ArrayList<String> - Pack of photos about the document.
Methods
+ addPhoto(url: String): void - Adds document to the report.

3.2.1.1.4 RoutingHandler

DirectionsService
Service for getting directions for couriers.
Attributes
Methods

+ route(request: DirectionsRequest, callback: Function): void - Asynchronous function that forms a proper route for given DirectionRequest instance and calls callback function

DistanceMatrixService
Distance matrix service for calculating routes.
Attributes
Methods
+ getDistanceMatrix(request: DirectionsRequest, callback: Function): void - Asynchronous function that calculated road aligned for given DirectionRequest instance and calls callback function

DirectionsRequest
Request class for parameterizing the services.
Attributes
+ destination: String - Destination point for the route. + origin: String - Origin point for the route. + travelMode: String - Travel mode for route (onfoot, car, public).
Methods

3.2.1.2 Storage

3.2.1.2.1 DataStorage

MySQL_Database
Holds all the data other than compromise requests. Such as user records.
Attributes
+ host: String - Name of the host + username: String - Name of the user + password: String - Name of the password

+ database: String - Name of the database + options: Object - JSON via express
Methods
+ createPool(options: Object): Promise - Creates a pool to hold requests + query(table: String, params: List): Promise Hold request when the pool is full.

3.2.1.2.2 I/O

RequestHandler
Transfers the client requests to RouteManager with a REST API.
Attributes
+ port: int - number of the port + userType: int -type of the user + userID: int - key for the user + password: String - password of the user
Methods
+ ServerRequestHandler(port: int, userType: int, userID: int, password: String): + authentication(userID: int, userType: int, password: String): boolean - Check credentials and return a boolean according to credentials matching or not. + requestManager(port: int, requestType: String, request: List): Organize the request and give them to the manager.

3.2.1.2.3 RouteManager

CompromiseManager
Deals with matching courier and customer accounts.
Attributes
+ userID: int - ID of the user + userType: int - Type of the user + compromiseID: int - Key for the compromise
Methods
+ CompromiseManager(userType: int, userID: int, compromiseID: int): void - Contractor for the compromise manager. + createCompromiseRequest(userType: int, userID: int, password: String): void - Creates a request to make a compromise. + createCompromise(int userType, int userID, int compromiseID, int addType): void - Creates a compromise.

MainManager
Gets the request from the I/O layer and organizes the processor layer to provide the data to the clients.
Attributes
+ mysqlConnection: MySQL_Database - Mysql database. + restApi: RequestHandler - Starts request handler. + database: String - Name of the database. + sqlUserName: String - Username for MySQL. + sqlPassword: Spring - Password for MySQL.
Methods
+ MainManager(): void - Constructor for the main manager. + start():void - Starts the connection. + stop():void - Terminal the connection. + requestProcessor(requestType: String, request:List):void - Organize the requests.

Authentication
User authentication during server access.
Attributes
+ userID: int - User id. + userType: int - Type of the user. + password: int - Password of the user. + connectionPas: String - connection cod.
Methods
+ Authentication(userType: int, userID: int, password: String): void - Check credentials. + getConnectionPas(userType: int, userID: int, password: int): bool - Approves the connection return a boolean according to credentials matching or not. + connection(): boolean - Establish connection if the credentials are matching

4. Development/Implementation Details

4.1 Client Side

The UI was created with Flutter Framework. For version control GitHub was used so that the UI team could get the latest version of the UI codes. In order to write

Flutter code, Android Studio is used as the development environment. Android Studio provides emulators which we can see how the widget looks on a virtual phone. This way we could detect the errors and correct them. Apart from technical details, the UI team has internal meetings for discussing the looks and functionalities of the screens.

4.2 Server Side

The back-end system was created with Node, Express frameworks integrated with MySQL. The back-end crew worked on a different branch than the UI team in the GitHub repository in order to maintain concurrency and prevent conflicts. We used Insomnia REST Client for testing our API and Swagger to provide documentation for the frontend team. For authentication, we used the User table on our database. On the other hand, for authorization, we choosed to store tokens on client-side. In order to achieve that we needed JSON Web Tokens, JWT allows us to sign our token before handing over the client with an expiration date and user identification for further authorization [3]. For storing user information we utilized Amazon Relational Database Service (RDS) MySQL database system [4]. Images related to users and packages like driver license and package proofs are stored in Amazon Simple Storage Service (S3) [5]. All file and information relation record and retrieval operations are maintained by Node.js backend application.

4.2.1 Verification Implementation

We use mail verification with a free outlook mail account to send a 4 digit pin and use government identity check API to check validity of user information. Since the receiver of a delivered package is not registered in our system we send a mail delivery confirmation link to the receiver which is provided when the package is added to our system. Given confirmation link holds encrypted information about the package and courier in order to process and authorize the confirmation request.

4.2.2 Package Volume Requirement Implementation

In the case of a courier carrying more than one cargo in a single transport operation, we needed to check volume eligibility of new cargo insertion to the vehicle. Since a single body of a package can deny more than its actual volume after

many insertions. We used a 3D Rectangular Box Packing algorithm to check and find an arrangeable combination of packages so that the courier's vehicle can fit as many packages as physically possible to maximize profit.

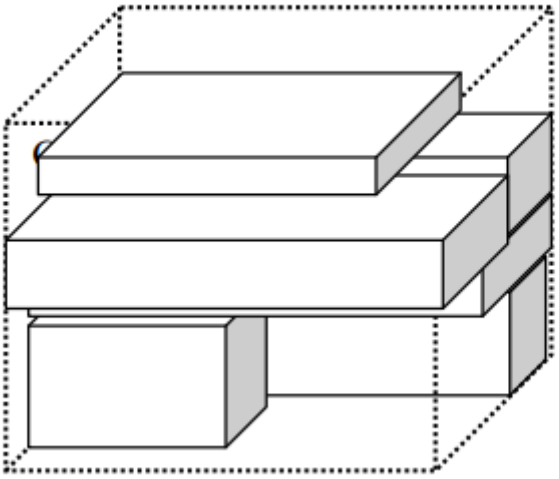


Fig. 2 A possible solution for box packing [6].

5. Testing Details

5.1 Testing the UI

Testing the user interface was done by utilizing the emulators inside Android Studio. The application was run on the emulator and the behavior of widgets was tracked. Any misbehavior of the widgets were noted and corrected in the iteration.

The visualization of the UI also enabled us to test different color combinations for ease of readability. For a specific widget, different positions were also tested to enhance user experience while keeping UI simple as possible.

5.2 Integration Tests

As the front-end and back-end team developed pieces of product that can be integrated into one entity, we did integration tests to see the behavior of the end product. By doing these, we were able to see bugs that cannot be seen in separate testing. After updating the product to resolve the bug, we run the integration test to

see if the issue persists or not. In each step, we carefully integrated pieces together to decrease the number of bugs as much as possible.

5.3 Testing the API with Insomnia

The communication of the client-server pair is an important point in our application. While building endpoints for different functionalities, we wanted to test these endpoints without requiring a front-end that was built for that functionality. For this purpose, we used Insomnia and tested our endpoints. Using Insomnia was useful for our application for a couple of reasons. First, we saw the time that it takes to send a request and get a response. Second, we saw the HTTP response codes and implemented front-end behavior depending on it. Lastly, the front end team could see the structure of JSON files so that they can parse the JSON responses accordingly without any issues.

6. Maintenance Plan and Details

Maintenance is a crucial part of our application due to the fact that the application relies on a server, Flutter packages can get deprecated due to updates and bug fixes.

6.1 Flutter Packages

Flutter packages can get updated depending on the developers, new upgrades to the Flutter Framework itself or bug fixes. Getting the last version of a package is crucial in our system because old versions may contain deprecated functions, parameters, variables etc. If any package contains one of these, the UI may behave in an unintended way.

6.2 Bug Fixes

Due to the nature of programming, bugs and other issues can occur while Wheelancer is used. An info section will be put to the profile pages so that users can contact the developers for reporting bugs and issues. Then, the developers will strive to fix the issues so that they will not happen in the future.

6.3 Server Maintenance

Currently, the app relies on a free Amazon Web Services such as EC2 and S3 in order to operate. Depending on the use of our product by increase in user base, a single EC2 instance might not handle all the traffic coming through clients. However, EC2 allows us to scale with increasing EC2 instances and distribute load with elastic load balancer. Moreover, the status of the server will be checked regularly for major issues. If a problem happens, the person responsible for the server needs to be notified quickly so that the server can be restarted quickly in order to make the app functional again. In case of dependency vulnerability, Wheelancer utilizes npm for managing packages. Therefore, used packages are always up-to-date.

7. Other Project Elements

7.1.Consideration of Various Factors in Engineering Design

Wheelancer is an application that matches the senders and couriers and aims to decrease the cost of sending cargo, assist financial support to any person with a vehicle by creating a network of senders and couriers. The first main consideration of this project is to create an efficient algorithm to match the couriers and senders. The aim of creating an efficient matching system is to minimize the errors of matched senders and couriers so that in the path that is chosen by sender must be chosen correct couriers even provides more than one courier for the path. Users of this application must not encounter matching problems.

Second consideration of the Wheelancer is to make the user insightful to the application functionalities. Hence, the process of sending a cargo must be explicitly simple as sending cargo must not affect the sender's life.

Third consideration is security. For the security of the cargo, budget systems will be implemented, the courier's security number will be taken by application and the courier photo of the cargo must be sent by the time the cargo is taken and delivered to its destination.

	Effect Level	Effect
Public Health	8	Due to COVID, people started to worry about their health. Our application will alleviate their problems by helping them to

		carry their cargo without the need of going to the cargo branch.
Public Safety	7	Since we are using personal data, we need to ensure that personal information is stored and secured properly.
Public Welfare	0	This factor has no effect on the application.
Global Factors	2	We need to develop the application in different languages if necessary.
Cultural Factors	0	This factor has no effect on the application.
Social Factors	4	The effects of COVID related to customer preferences affects the application.

Table 1: Factors that affected implementation

7.2.Ethics and Professional Responsibilities

One of the issues regarding our application is that as the developers we cannot see the contents of the packages of our customer due to privacy. Nevertheless, this also means that Wheelancer can be used to transfer illegal substances. Another issue is that the courier can steal the package, which will cause problems for the customers and decrease the reputation of our app.

In order to prevent these types of incidents, we will use a User Agreement to ensure any misconduct will be penalized by the authorities. In order to show the current location of the package, we will require the couriers to keep their GPS on while delivering the package. This can lead to problems that can endanger the safety of the courier because of unsafe roads or routes. Therefore, we will give the courier to choose whether he/she wants to deliver a particular package or not.

We will provide an agreement about the access permissions when a user downloads Wheelancer. This way every user will know how we process their data and decide whether or not they want to use Wheelancer or not.

In order to ensure that the couriers have licenses for their car type, we will require the photo of their license which will be stored in our database. Therefore, any data leak will be a violation of privacy rights. We will ensure that we will not share this data and no one can reach this information using our app apart from the people who need this information.

Last but not least, we should be careful about the APIs, libraries and other things because of copyright issues. Therefore, we will use free alternatives to avoid any legal trouble. We will also reference the sources if we utilize their technologies in any way.

7.3.Judgements and Impacts to Various Contexts

Judgment Description	Impact Level	Impact Description
Impact in Global Context	2	We might develop the application in different languages if necessary.
Impact in Economic Context	7	The application will provide an income source for couriers.
Impact in Environmental Context	6	The couriers will utilize their vehicle more effectively. For example, truck owners do not need to return with empty trailers.
Impact in Social Context	4	Apart from the issues created by COVID, there is not any societal impact.

Table 2: Judgements and Impacts table

7.4 Teamwork Details

The broad distribution of workload among team members are as follows:

- Client side: Ümit Çivi, Muhammed Maruf Şatır
- Server side: Onat Korkmaz, Muharrem Berk Yıldız, Erdem Ege Eroğlu

7.4.1 Contributing and functioning effectively on the team

We divided the application into work packages as described in the analysis report. This way we determined our goals beforehand and achieved a work balance between CS491-2 and other courses. Each team member learned his own part individually and started to implement it right away. By reporting our progress in weekly meetings, we managed to establish a steady development curve. Moreover, we helped each other while brainstorming so that we can build a more complex

application. The most crucial part was the weekly team meetings because we discussed various things in these meetings while developing the application.

7.4.2 Helping creating a collaborative and inclusive environment

Weekly meetings played a key role in our application. These meetings were held on Zoom or Discord depending on our preference. This way we communicated in an effective manner and generated unique ideas while developing the application. For storing reports or other documents related to implementation, we used a Google Drive folder. In order to monitor our progress, we used a Trello board. Each work package was divided into smaller tasks and when a task was completed, the person responsible for it marked the task as completed. This way we were able to track the progress smoothly. For the project repository, we used Github which enabled us to have a neat organization.

Apart from the applications we use, another issue was resolving conflicts among team members. When two members gave two different ideas for the same problem, the remaining team members helped them to reach a state of compromise.

7.4.3 Taking lead role and sharing leadership on the team

As described in other sections, the project was divided into work packages and each work package has a different leader. This way we made every team member demonstrate his leadership skills to organize the work package progress flow. We thought that by making someone a leader of a specific work package, he can dedicate himself better to the project which in turn might increase his motivation.

7.4.4 Meeting objectives

In order to meet objectives, deadlines are set at each step of the development. By doing so, we created a steady work flow and built upon the progress that is made by other team members. Using a Trello board, we notified each other regarding the completed, on-going or newly created tasks. If any team member could not finish the task in time,which can happen due to our busy schedules and COVID, we strived to extend the deadline without affecting the whole structure.

7.5 New Knowledge Acquired and Applied

After determining the project idea, we started to talk about different tools that we can use for the development of our app. We focused on the development of a mobile application that will satisfy our requirements. After brainstorming sessions, we established a foundation to build our project. Then, each team member started to learn about his own part.

7.5.1 Flutter Framework

For the UI design of our project, Flutter Framework has been used. None of the members responsible for the front-end had knowledge about Flutter Framework. The first goal of the UI team was to learn how widgets and other components in Flutter works.

For testing the widgets while building them, the UI team used emulators inside the Android Studio development environment. These emulators enabled us to check the behavior and view of the user interface. This helped us to see our errors and allowed us to create better user interfaces for our users.

Apart from the widgets in the Flutter Framework, we also used packages for various purposes such as storing crucial info, map view, getting user location etc. For sending requests to the server, we used the HTTP package of Dart language. These different packages helped us to perform different operations without writing the whole process from scratch.

In order to use Flutter effectively, we learned Dart language, specifically the asynchronous function calls in Dart language that returns Future type objects. Since we are getting information from the server side, a delay is expected while getting the data. Until data arrives, we stopped the building process of the screen so that we can display the information from the server.

7.5.2 Amazon Web Services

In order to build our REST API, we needed an environment to build it on. At first, we thought using a virtual machine hosted on our machine would suffice for hosting both database and REST API. However, we decided on utilizing Amazon Web Services (AWS) since it is popular in the industry. We gave it a try and challenged ourselves to learn the ecosystem. After learning about services that

Amazon provides. We decided on using EC2 (Elastic Compute Cloud) for hosting our API in order to scale our product with ease in the case of a need with Amazon's own balance loader service, RDS (Relational Database Service) for hosting our MySQL Server and S3 (Simple Storage Service) as a file storage to provide both security, speed and reliability for storing pictures of delivered good and driver license of courier. Ultimately, this architecture communicates with the world through Amazon Virtual Private Cloud (VPC) which allows only network traffic in specified ports and ensure security.

7.5.3 JSON Web Tokens

At first our main intuition about authorization in our system was built upon the idea of having a separate table on our MySQL server for handling authorization. However, after looking for best practices. We learned about JWT and utilized it in our project.

7.5.3 MySQL

After taking the CS353 course, we were all introduced to basic SQL, and with this project we had the chance to work with various MySQL queries in order to present desired data for the front-end instead of testing use cases outside a real-time project development environment. We learned to cooperate on integrating database administration with Express framework in order to build a REST API. During the building process, we learned about being cautious about any security vulnerabilities. Basically, this project process has been a great environment for us to practice and master building real-time MySQL databases.

7.5.4 Websocket

We wanted to implement chat as a feature in our project as a means of communication. However, we could not integrate the chat into our application. In order to implement the chat we used Websocket to listen to incoming messages from the other users. We believe that REST api is beneficial when there is a scenario. However, in chat implementation we need a real time scalable application,

thus, we prefer to make use of websocket. Indeed, that is a stateful protocol which we need. Although we generally used REST, we acknowledged websocket for the chat part of our application.

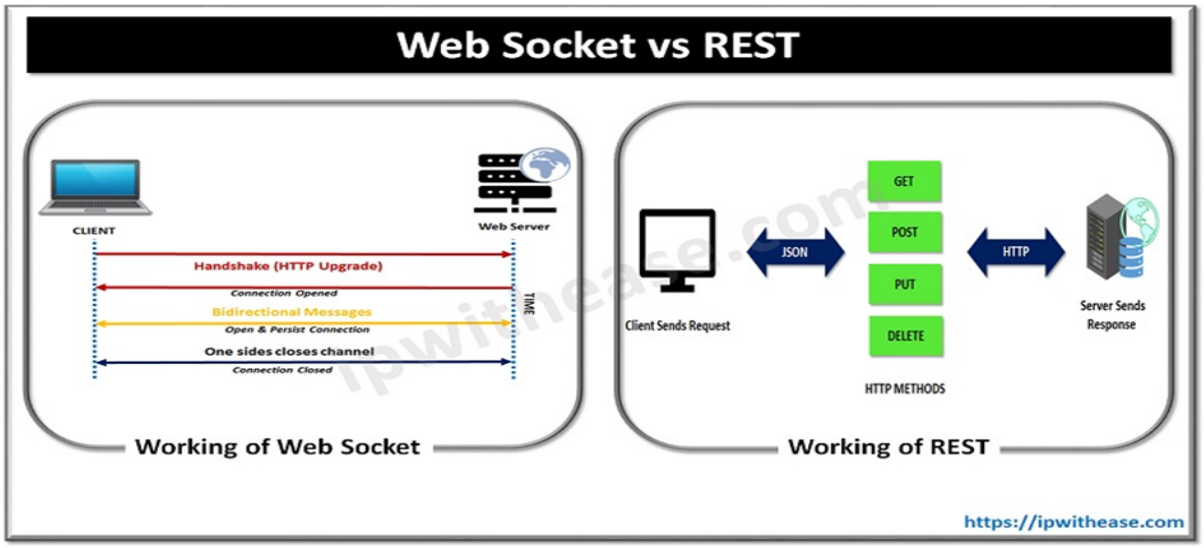


Fig. 3 Difference between websocket and http [7].

8. Conclusion and Future Work

8.1 Conclusion

In the end, we managed to build an application that is similar to the idea we have had at the start of the fall semester. We hope that Wheelancer will be a useful platform for cargo transportation. In terms of experience, this project enabled us to gain valuable experience in different areas such as front-end and back-end design. We learned how to collaborate as an effective team, helping each other while someone has an issue and the importance of leadership. All of these enabled us to develop a complex, functional end product. We also realized the importance of testing in a complex project so that we can understand our errors and not make them again.

8.2 Future Work

Depending on the feedback of our supervisor, jury members and users, we will strive to update the app to meet their requirements and make a better application. We also plan to deploy the application on Google Play Store to see the

potential user count. This will also increase the number of feedback we will get from the users, which will enable us to develop a better application than the current version.

9. User Manual

Installation Details

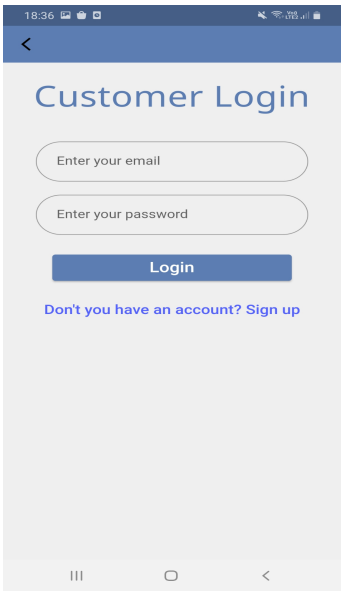
Run the apk file and give necessary permission if the UI asks.

Welcome Page

The user selects his/her account type.

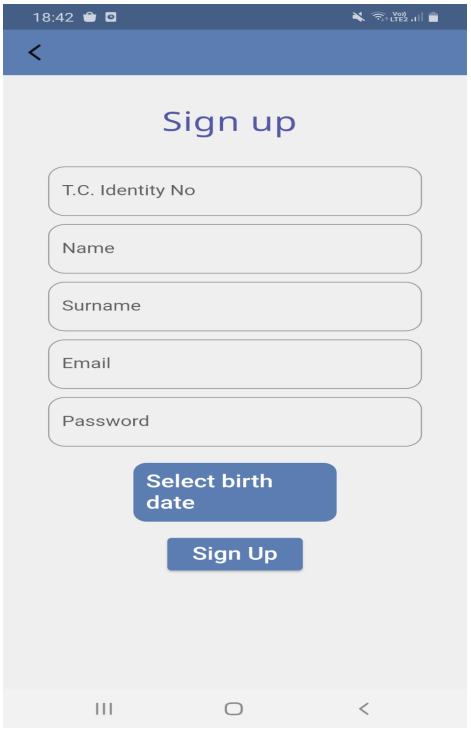
Customer Login

Customer has to choose an email and password to enter his/her account. If a customer does not have an account, the customer can tab the Sign Up button as indicated.



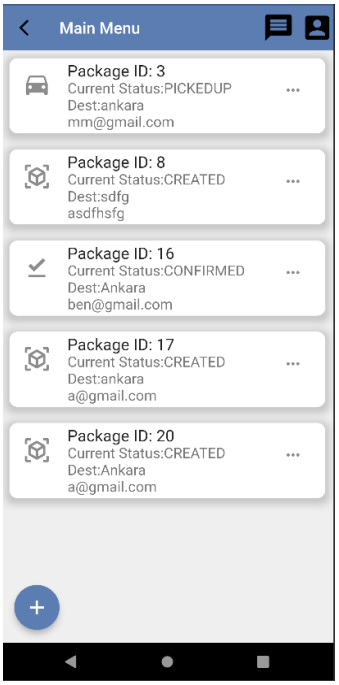
Customer/Courier Sign Up

Users can sign up from this page, by writing TC identity, name, surname, email, password and choosing birth date.



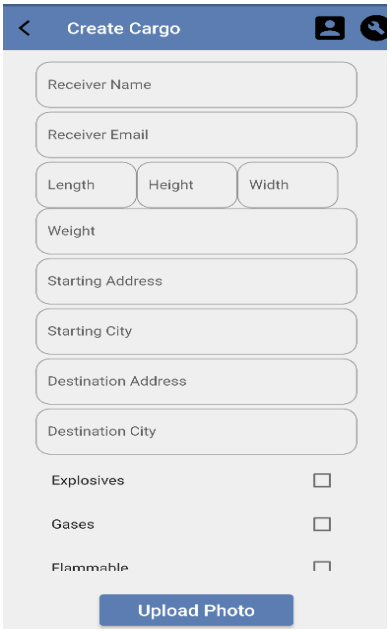
Customer Main Menu

Customers can see their current packages and their current state from this page. To add a new package click the button at the left-bottom corner. To see the offers made to the packages click the “message” icon at the app bar. Clicking the “human” icon will direct the customer to the profile page.



Create Cargo Menu

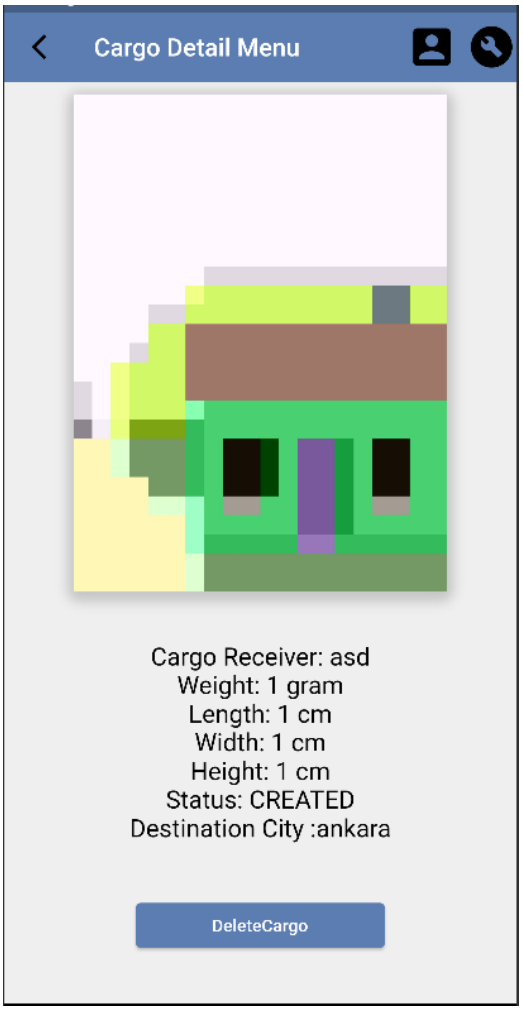
The customer can create a new cargo to be delivered. The customer needs to add the necessary information in the below screenshot. A photo of the package is also expected for verification. After entering the necessary information and uploading a photo, the user can upload the package.



Detailed Cargo Menu

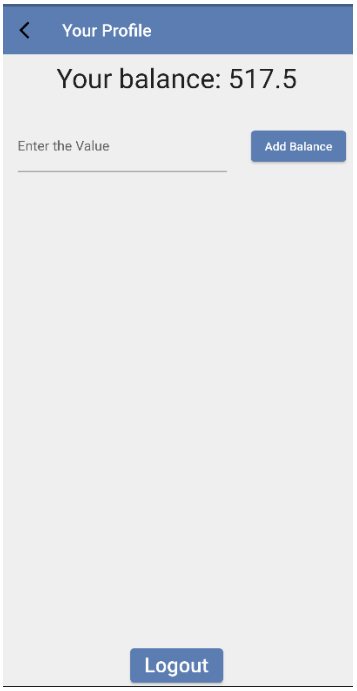
After clicking the button at the right of a specific cargo, the customer will be redirected to this page. In this page, the customer will see the details of the package and the picture uploaded to the server. Depending on the current status of the package different buttons will appear.

Status: CREATED - delete cargo button
Status: NEGOTIATED - contact the courier button for pickup
Status: PICKEDUP - live location button
Status: COMPLETED - no buttons
Status: FINISHED - review courier button

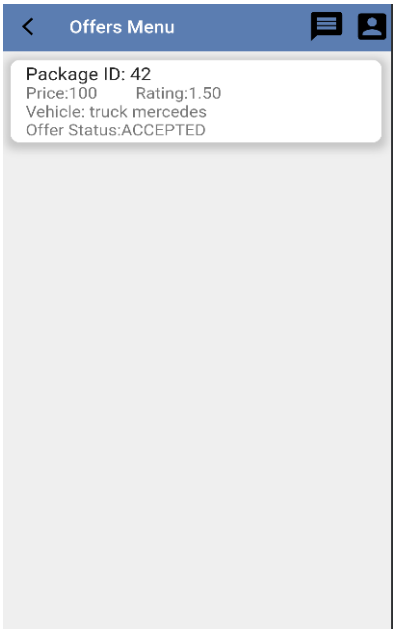


Profile Menu

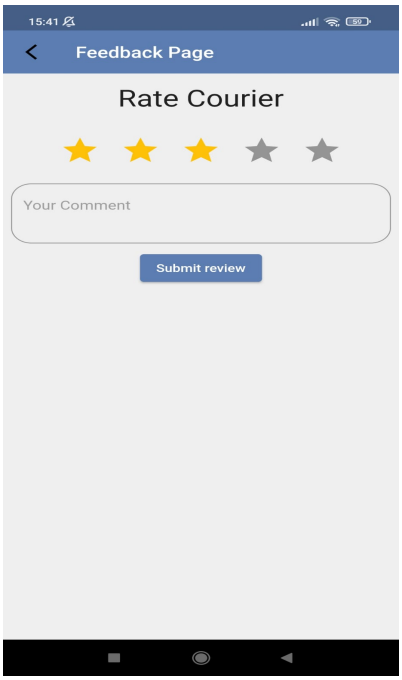
The user can add balance to his/her account from this page. The balance is just a visual feature for this version. Depending on the requirements of the project, different payment methods will be considered. Moreover, the user can logout from this page.



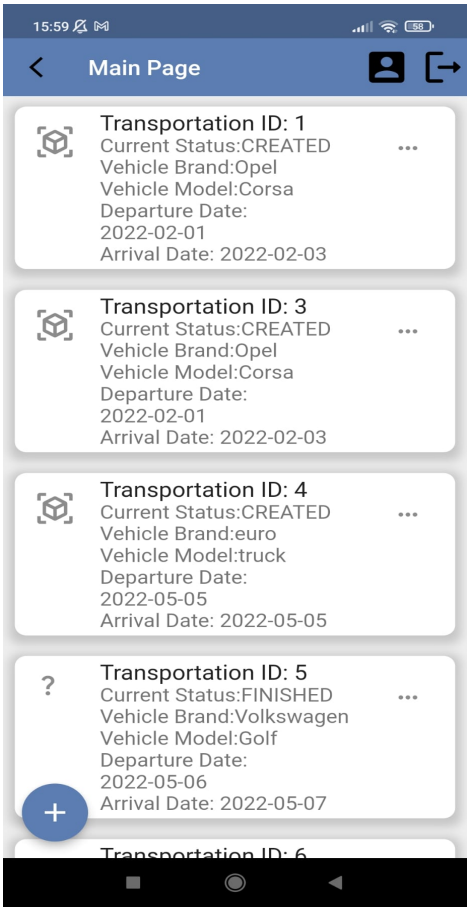
Offers Page: Users can see the offers made by couriers. Users can accept or reject the offers.



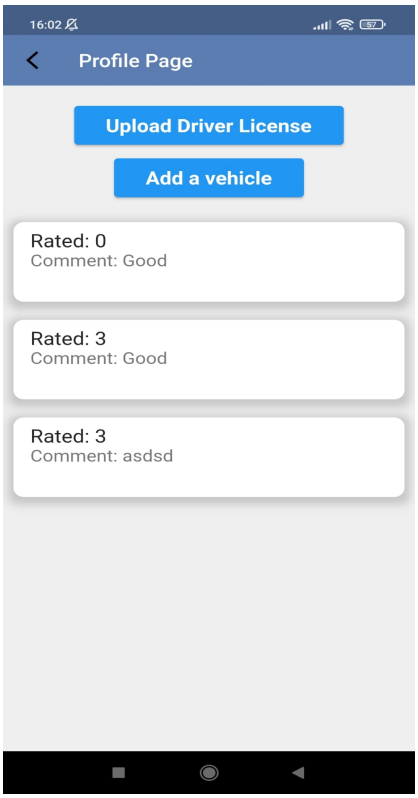
Rating Courier: When the process of delivering the cargo finishes, the user can review the courier from this menu.



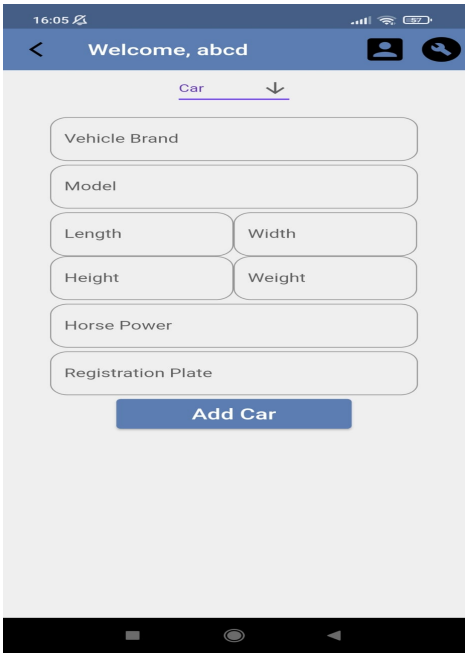
Courier Main Page: Courier can see and select the transportations. The plus button opens the menu for creating new transportation.



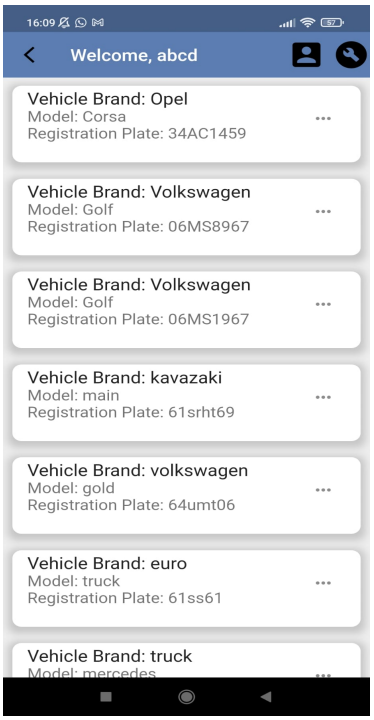
Courier Profile Page: Courier can upload driver license and add a vehicle in a new page. Moreover, couriers can see anonymous rates and comments on the delivery process.



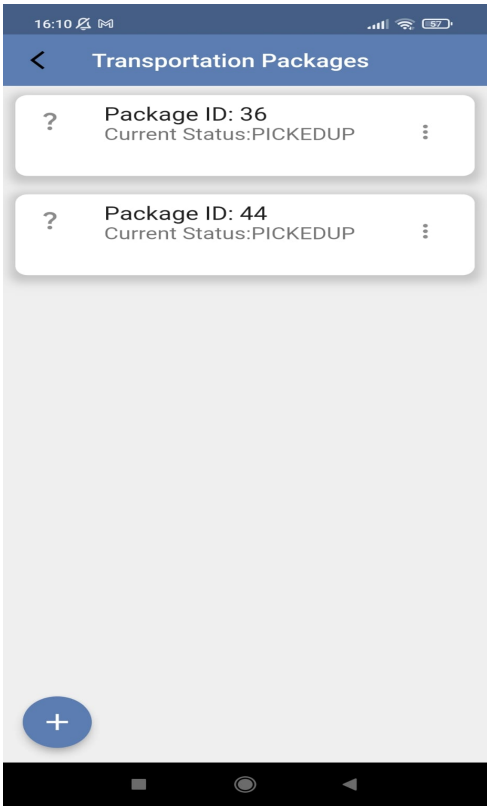
Courier Adding Vehicle Page: Courier specifies vehicle type and by selecting the required information about the vehicle can add his/her vehicle on his/her account.



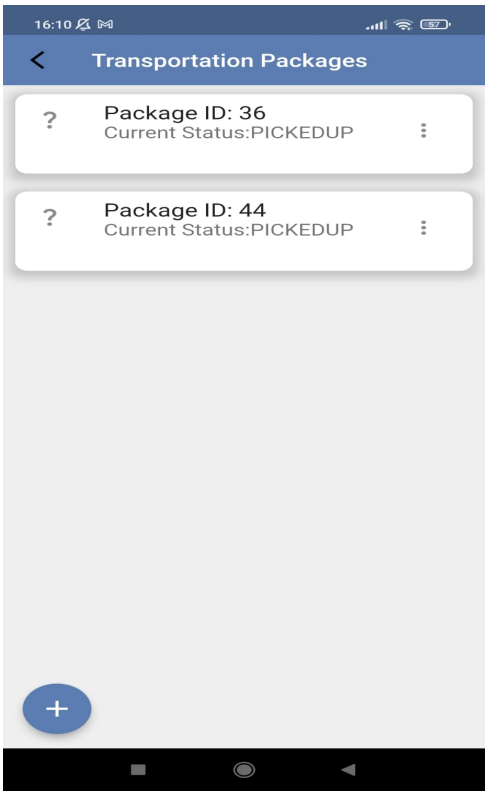
Courier Transportation Vehicle Selection: Courier must select the vehicle to continue the process of creating transportation.



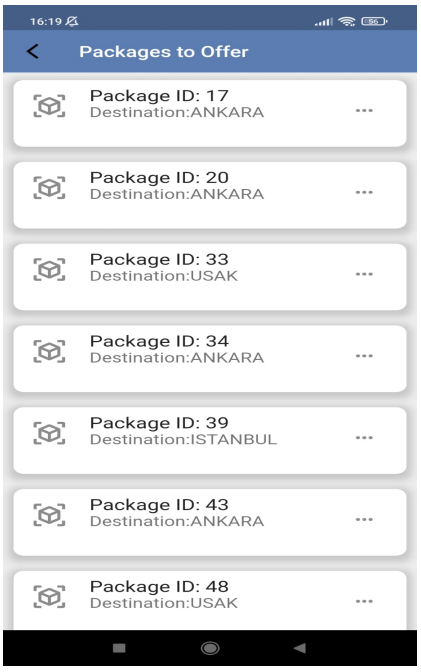
Courier Transportation Page: Courier can see packages that are added to current transportation. In this page, by clicking the plus sign, searching for new cargos and by clicking on the cargo, different buttons are uploaded depending on the status of the cargo.



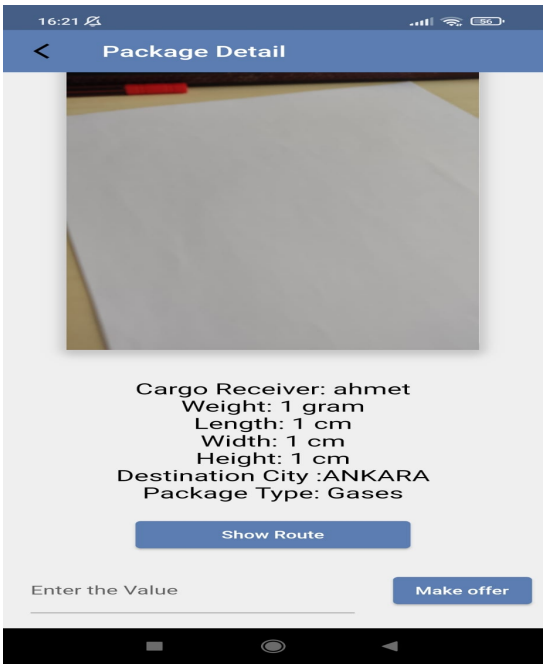
Find Near Packages Page: Courier must select the city and indicate radius for searching cargos near to courier.



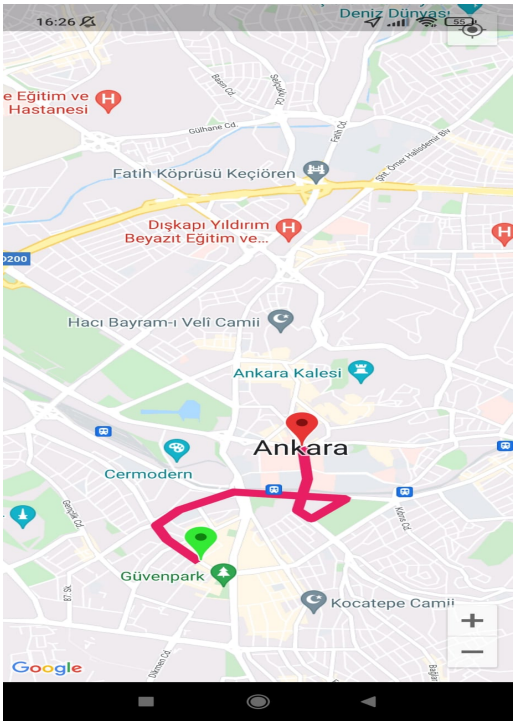
Near Packages Menu: Courier can see near packages and by clicking the button next to them see cargo details.



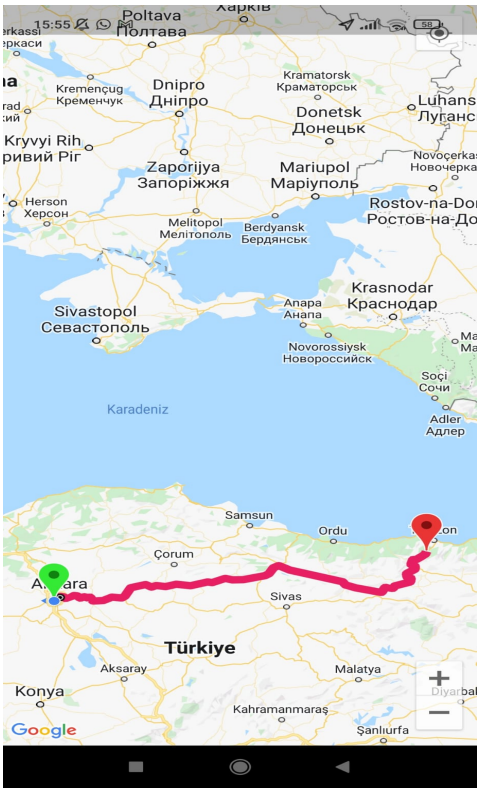
Cargo Detail Menu: Courier can see the information related to the package. Make an offer with a specified value. By clicking the show route, the courier can see the cargo source location and destination location.



Cargo Destinations: Courier by clicking the show route button on the cargo detail menu can see the source location and destination location.



Picked Up Show Destination: Courier can click the pickup package and see his/her location and path to the destination location of the package.



10. Glossary

- **Wheelancer:** Name of our mobile application.
- **API:** Application programming interface
- **HTTP:** HyperText transfer protocol
- **UI:** User interface
- **Flutter:** Open source framework for building multi-platform applications
- **Widget:** Fundamental building blocks in Flutter
- **Box:** A type of safe storage in Flutter to store user sensitive data such as authorization tokens

11. References

[1] C. Smith, “Interesting ups statistics and facts,” DMR, 27-May-2021. [Online]. Available: https://expandedramblings.com/index.php/ups-statistics-and-facts/. [Accessed: 06-Oct-2021].

[2] “What is a rest api?,” Red Hat - We make open source technologies for the enterprise. [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Accessed: 03-Nov-2021].

[3] “JSON web tokens introduction,” JSON Web Token Introduction. [Online]. Available: https://jwt.io/introduction. [Accessed: 04-May-2022].

[4] “RDS: Understanding animal research in medicine,” *Amazon*, 2007. [Online]. Available: https://aws.amazon.com/rds/?nc1=h_ls. [Accessed: 04-May-2022].

[5] “S3,” *Amazon*, 2002. [Online]. Available: https://aws.amazon.com/s3/?nc1=h_ls. [Accessed: 03-May-2022].

[6] “An efficient algorithm for 3D rectangular box packing - park beach system's.” [Online]. Available: https://www.parkbeachsystems.com/images/usps/An_Efficient_Algorithm_for_3D_Rectangular_Box_Packing.pdf. [Accessed: 02-May-2022].

[7] R. Bhardwaj and Rashmi BhardwajMore From This AuthorI am here to share my knowledge and experience in the field of networking with the goal being - "The more you share, “Web socket vs rest,” *IP With Ease*, 18-Nov-2021. [Online]. Available: https://ipwithease.com/web-socket-vs-rest/. [Accessed: 06-May-2022].