Bilkent University

Department of Computer Engineering

Senior Design Project

Project name: Wheelancer

Low-Level Design 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

February 28, 2022

Contents

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 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. 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 transportation and help people by enabling them to have another source of income in the pandemic.

In this low-level design report, we will provide a low-level description of the system. Firstly, we will explain the object design trade-offs. Then, we will elaborate the system architecture that we introduced in the high-level design report. After that, we will analyze the class interfaces inside these packages deeply.

1.1 Object design trade-offs

1.1.1 Performance vs Security

Two of them are essential for Wheelancer application. Since mapping and matching must be done rapidly and private information must not be accessed by other people.

While hashing passwords and encrypting credit card information security is crucial in those cases, we will not take into account performance largely. Moreover, those algorithms do not deplete resources of new mobile phones. Since Wheelancer uses client server design paradigm, the performance will not be a problem for phones for the matching algorithms. However, this will cause a decrease in performance in servers but this will lead applications to be more secure.

1.1.2 Reliability vs Scalability

Reliability of Wheelancer is more important than the scalability of our application. Since we are dealing with sensitive data such as user passwords, license information etc. we cannot take risks to cause data losses while expanding our application. We will strive to scale our application while protecting user data as much as we can. Moreover, we will implement optimization algorithms for routing so that our application does not make the user wait a significant amount of time.

1.1.3 Functionality vs Usability

Wheelancer will select usability over functionality. We will avoid unnecessary and complex UI functionalities in order to build an app that can be used easily by the users. Nevertheless, maintaining a simple UI while building complex functionalities such as drawing routes will be a challenge for our developers.

1.2 Interface documentation guidelines

Class Name
Class description
Attributes
Attribute list and descriptions
Methods
Method list and descriptions

Table 1: Interface Documentation Guidelines

1.3 Engineering standards

While writing reports we are following IEEE standards [1] for citation and UML standards for drawing diagrams [2].

1.4 Definitions, acronyms, and abbreviations

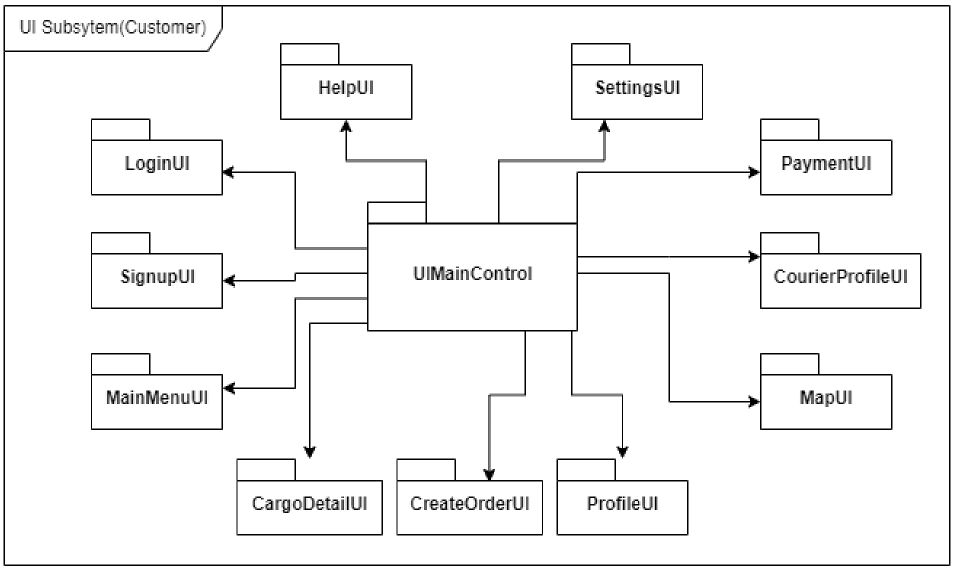
- **Wheelancer:** Name of our mobile application.
- **API:** Application programming interface
- **MongoDB:** A NoSQL database system [3]
- **HTTP:** HyperText transfer protocol
- **UI:** User interface
- **Flutter:** Open source framework for building multi-platform applications
- **Widget:** Fundamental building blocks in Flutter
- **Box:** A type of safe storage in Flutter to store user sensitive data such as authorization tokens

2. Packages

Wheelancer is built using the Client-Server architecture. Client side includes the views and controllers for customers and couriers. Server side contains map, storage and database subsystems. The server side is responsible for storing data and processing requests made by the users. The client side is responsible for presenting the data fetched from the server side in a readable way. In order to establish a connection between these subsystems we use HTTP requests. By using Client-Server architecture, we aim to make the server side handle the hard work, while the client side can easily update itself when a request response comes from the server.

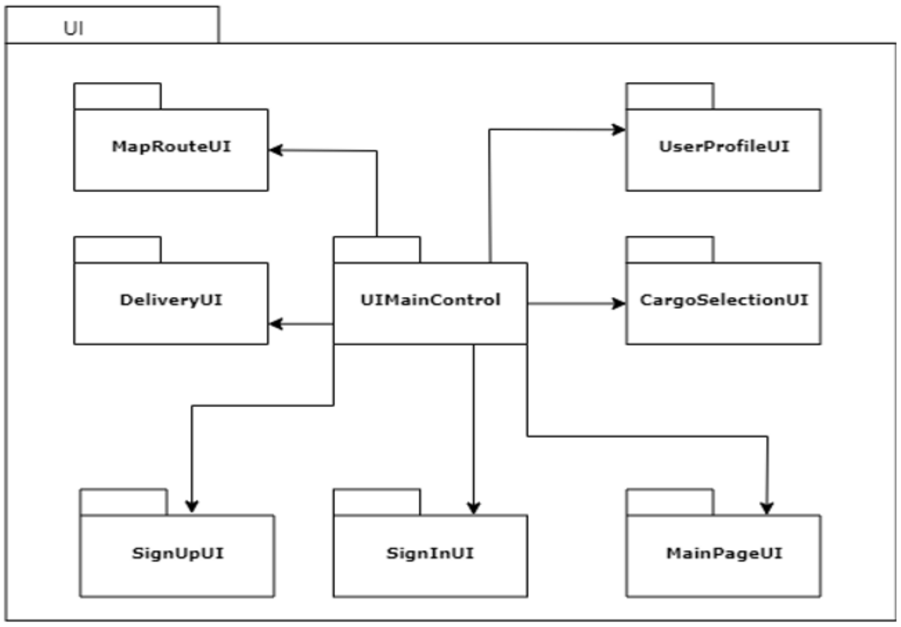
2.1 Client

2.1.1 Customer View



UIMainControl: The controller of the customer user interface. It manages and changes the user interfaces based on user input and client response.
LoginUI: User interface for the login.
SignupUI: User interface for the signup.
MainMenuUI: User interface for the main menu which the user can access many features such as cargo details, cargo creation, profile etc.
CargoDetailUI: User interface for cargo detail which is chosen by the user.
CreaterOrderUI: User interface for creating a new order
ProfileUI: User interface for the customer profile
SettingsUI: User interface for the application settings for the customer
HelpUI: User interface for the application help screen for the customer
PaymentUI: User interface for payment view after delivery is complete
CourierProfileUI: User interface for customer to inspect a couriers so that the customer can accept or decline the customer
MapUI: User interface to track a cargo that is being delivered by a courier.

2.1.2 Courier View



***UIMainControl*:** The controller of the courier user interface. It manages and changes the user interfaces based on user input and client response.
***SignInUI*:** User interface for the login.
***SignupUI*:** User interface for the signup.
***MainPageUI*:** Main interface of the courier. Couriers can access functionalities of cargo selection, profile, selecting route for cargo selection etc.
***DeliveryUI*:** Delivery page will be used by courier to upload a photo of the cargo delivering.
***MapRouteUI*:** Route page will be used to see the best route from current location to delivery location. Delivery location can be changed if there is more than one courier on the route.

2.2 Server

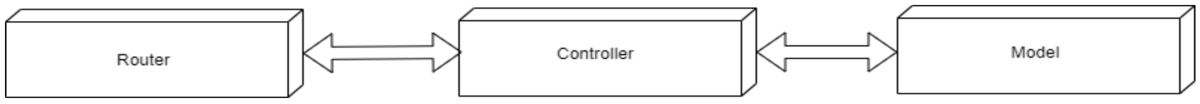
2.2.1 REST API

REST API is built upon Controller, Router and Modal Structure. Each block is responsible for a specific set of actions during the lifetime of a HTTP request.

Router ensures incoming request has proper body, header and query formation. If the request structure is not in feasible format without further actions it directly answers requests with proper status codes. Otherwise it redirects incoming requests to the controller.

Controller fetches required data and executes functions related to the received endpoints, in our project most of the controller functions prepare data to perform database queries in the Model.

Model forms a SQL query for mySQL pool or calls collection fetch for MongoDB collection then executes query and finalizing request lifetime by sending counter response according to the data gathered from databases.

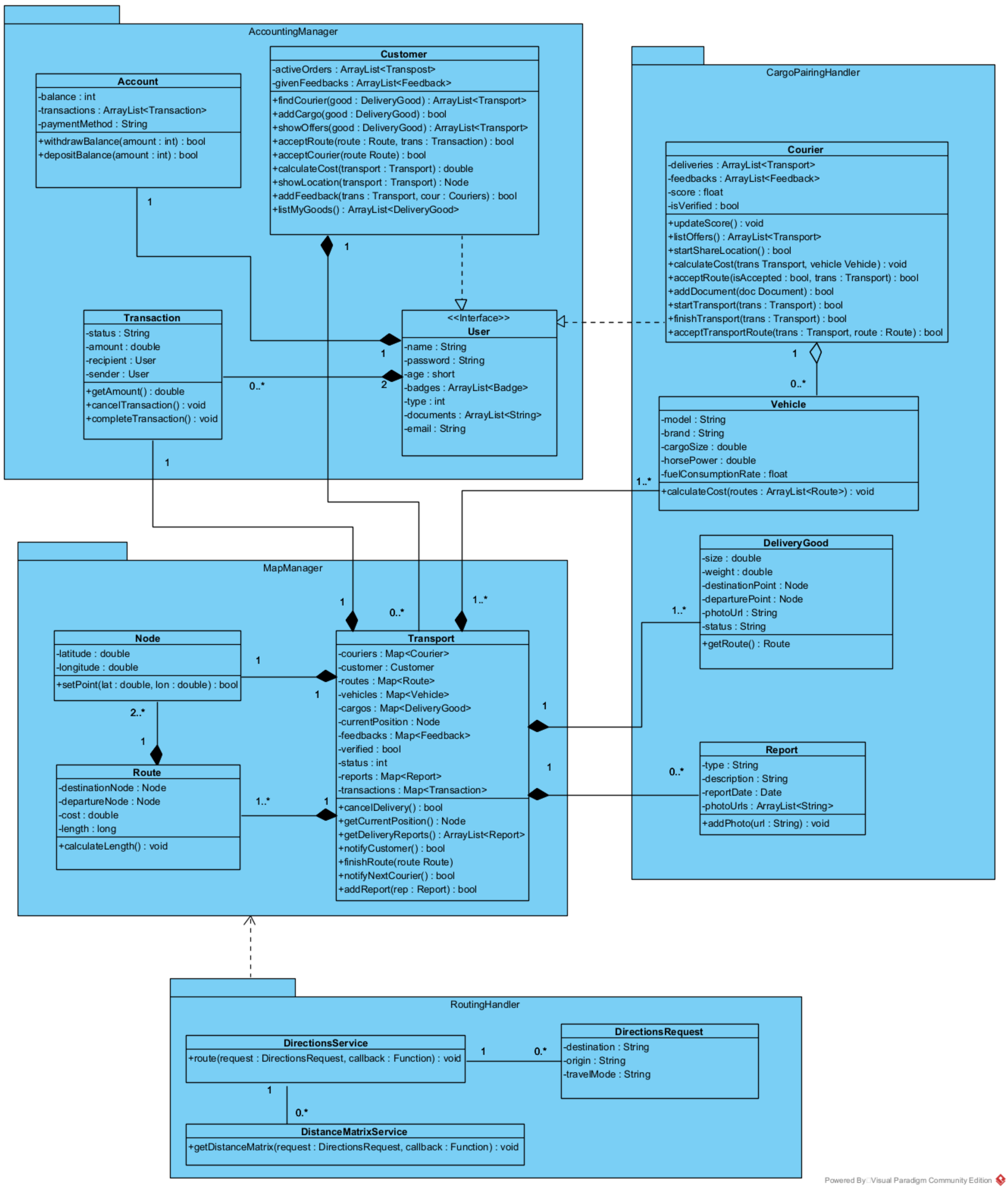


AccountingManager: Keeps track of the user activity involving accounting for transferring balances, transaction verification.

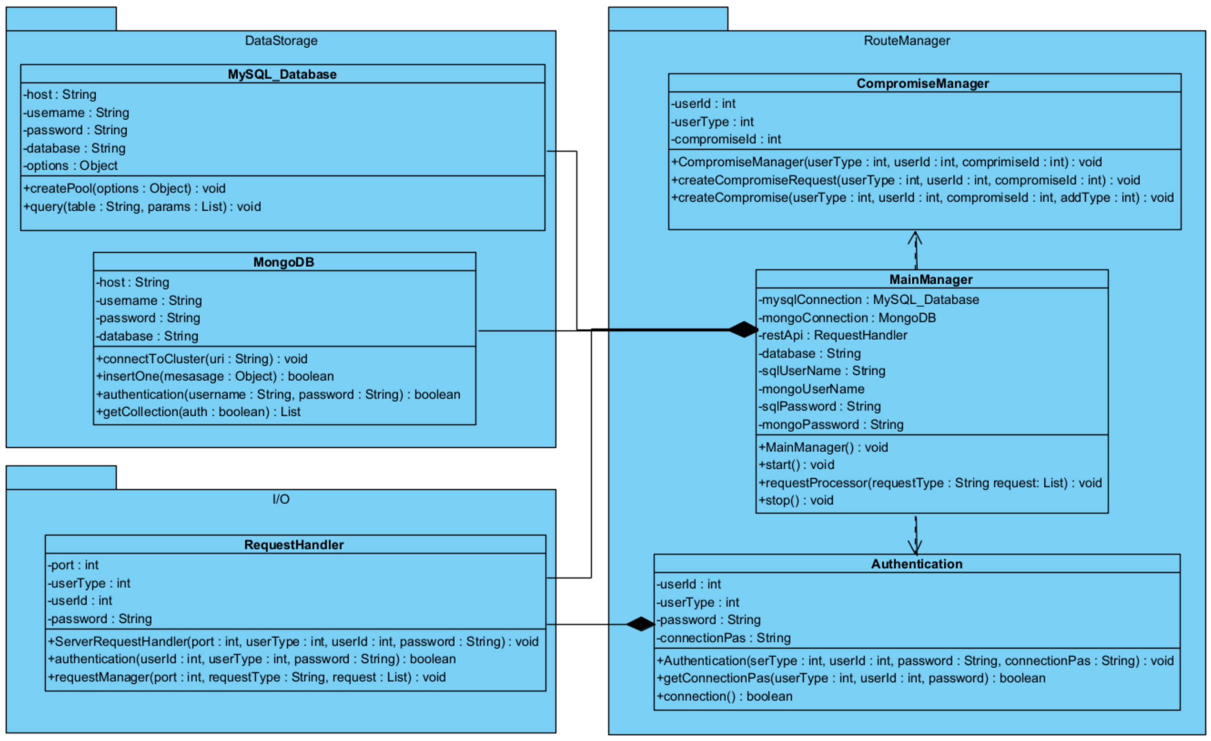
CargoPairingHandler: Handles weight and volume constraint while process of pairing for each Transport object.

MapManager: Manages unclaimed, claimed and started transport routes providing routing functionality for Courier and keeps track of live location and process of transportation.

RoutingHandler: Utilizes routing API to determine distance and road matrix while peering process to assign offers to Customers.


AccountingManager
Account
-balance : int
-transactions : ArrayList<Transaction>
-paymentMethod : String
+withdrawBalance(amount : int) : bool
+depositBalance(amount : int) : bool
Customer
-activeOrders : ArrayList<Transpost>
-givenFeedbacks : ArrayList<Feedback>
+findCourier(good : DeliveryGood) : ArrayList<Transport>
+addCargo(good : DeliveryGood) : bool
+showOffers(good : DeliveryGood) : ArrayList<Transport>
+acceptRoute(route : Route, trans : Transaction) : bool
+acceptCourier(route Route) : bool
+calculateCost(transport : Transport) : double
+showLocation(transport : Transport) : Node
+addFeedback(trans : Transport, cour : Couriers) : bool
+listMyGoods() : ArrayList<DeliveryGood>
Transaction
-status : String
-amount : double
-recipient : User
-sender : User
+getAmount() : double
+cancelTransaction() : void
+completeTransaction() : void
<<Interface>>
User
-name : String
-password : String
-age : short
-badges : ArrayList<Badge>
-type : int
-documents : ArrayList<String>
-email : String
1
1
1
0..*
2
1
CargoPairingHandler
Courier
-deliveries : ArrayList<Transport>
-feedbacks : ArrayList<Feedback>
-score : float
-isVerified : bool
+updateScore() : void
+listOffers() : ArrayList<Transport>
+startShareLocation() : bool
+calculateCost(trans Transport, vehicle Vehicle) : void
+acceptRoute(isAccepted : bool, trans : Transport) : bool
+addDocument(doc Document) : bool
+startTransport(trans : Transport) : bool
+finishTransport(trans : Transport) : bool
+acceptTransportRoute(trans : Transport, route : Route) : bool
1
0..*
Vehicle
-model : String
-brand : String
-cargoSize : double
-horsePower : double
-fuelConsumptionRate : float
+calculateCost(routes : ArrayList<Route>) : void
1..*
DeliveryGood
-size : double
-weight : double
-destinationPoint : Node
-departurePoint : Node
-photoUrl : String
-status : String
+getRoute() : Route
1..*
Report
-type : String
-description : String
-reportDate : Date
-photoUrls : ArrayList<String>
+addPhoto(url : String) : void
0..*
1
MapManager
1
0..*
1..*
Node
-latitude : double
-longitude : double
+setPoint(lat : double, lon : double) : bool
1
1
2..*
1
Route
-destinationNode : Node
-departureNode : Node
-cost : double
-length : long
+calculateLength() : void
1..*
1
Transport
-couriers : Map<Courier>
-customer : Customer
-routes : Map<Route>
-vehicles : Map<Vehicle>
-cargos : Map<DeliveryGood>
-currentPosition : Node
-feedbacks : Map<Feedback>
-verified : bool
-status : int
-reports : Map<Report>
-transactions : Map<Transaction>
+cancelDelivery() : bool
+getCurrentPosition() : Node
+getDeliveryReports() : ArrayList<Report>
+notifyCustomer() : bool
+finishRoute(route Route)
+notifyNextCourier() : bool
+addReport(rep : Report) : bool
1
RoutingHandler
DirectionsService
+route(request : DirectionsRequest, callback : Function) : void
1
0.*
DirectionsRequest
-destination : String
-origin : String
-travelMode : String
1
0.*
DistanceMatrixService
+getDistanceMatrix(request : DirectionsRequest, callback : Function) : void

2.2.2 Storage


DataStorage
MySQL_Database
-host : String
-username : String
-password : String
-database : String
-options : Object
+createPool(options : Object) : void
+query(table : String, params : List) : void
MongoDB
-host : String
-username : String
-password : String
-database : String
+connectToCluster(uri : String) : void
+insertOne(mesasage : Object) : boolean
+authentication(username : String, password : String) : boolean
+getCollection(auth : boolean) : List
I/O
RequestHandler
-port : int
-userType : int
-userId : int
-password : String
+ServerRequestHandler(port : int, userType : int, userId : int, password : String) : void
+authentication(userId : int, userType : int, password : String) : boolean
+requestManager(port : int, requestType : String, request : List) : void
RouteManager
CompromiseManager
-userId : int
-userType : int
-compromiseId : int
+CompromiseManager(userType : int, userId : int, comprimiseId : int) : void
+createCompromiseRequest(userType : int, userId : int, compromiseId : int) : void
+createCompromise(userType : int, userId : int, compromiseId : int, addType : int) : void
MainManager
-mysqlConnection : MySQL_Database
-mongoConnection : MongoDB
-restApi : RequestHandler
-database : String
-sqlUserName : String
-mongoUserName
-sqlPassword : String
-mongoPassword : String
+MainManager() : void
+start() : void
+requestProcessor(requestType : String request: List) : void
+stop() : void
Authentication
-userId : int
-userType : int
-password : String
-connectionPas : String
+Authentication(serType : int, userId : int, password : String, connectionPas : String) : void
+getConnectionPas(userType : int, userId : int, password) : boolean
+connection() : boolean

2.2.2.1 DataStorage

MongoDB: Holds data about compromise requests for both drivers and package owners.

MySQL_Database: Holds all the data other than compromise requests. Such as user records.

2.2.2.2 I/O

RequestHandler: Transfers the client requests to RouteManager with a REST API.

2.2.2.3 RouteManager

CompromiseManager: Deals with matching courier and customer accounts.

MainManager: Gets the request from the I/O layer and organizes the processor layer to provide the data to the clients.

Authentication: User authentication during server access.

3. Class Interfaces

3.1 Client

3.1.1 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 text field of email. + password: TextController - A controller for 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.
+ phoneNumberTextController: TextController - A controller for text field of. phonenumber

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.
+ getCargo(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.

+ 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.
+ descTextController: TextController - A controller for the text field of description.

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): 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.

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.

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.
+ getCargoDetail(token:String):Future, Gets the details of the profile corresponding to the specified token via calling the API function.

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.

SettingsUI
The page where the user can change the settings of the application.
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.

PaymentUI
The screen where payment information will be taken from the customer
Attributes
+ box:Box - A storage that holds user sensitive data. + cardTextController:TextController - A controller for the text field of card number. + dateTextController:TextController - A controller for the text field of card expiration date. + cvvTextController:TextController - A controller for the text field of card cvv number.
Methods
+ build(context: BuildContext): Widget - The framework replaces the subtree below this widget with the widget returned by this method + confirmPayment(amount: float, cardNo: String, date: String, cvv: String): Future - the function that calls the API function to perform payment for the cargo

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.
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

CourierProfileUI
The page where the customer will be able to see the detailed profile of the courier who wants carry the customer's cargo
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 + getCourierInfo(courierID: int): Future - Calls the API function to fetch the information of the courier who wants to carry the cargo

3.1.2 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
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. + StatefulDrowDown(): Widget - It returns a widget for couriers to choose their vehicle type. + StatefulFilePicker(): Widget - It returns a widget for couriers to upload their driver's license.

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

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 Server

3.2.1 REST API

3.2.1.1 MapManager

Node
Representation of a singular coordinate.
Attributes
- latitude: double - Latitude of a coordinate. - longitude: double - Longitude of a coordinate.
Methods
+ setPoint(lat, lon): bool - Sets the node to a specific position. + calculateLength(): void - Calculates road-alligned distance between nodes.

Route

Representation of a single route for a delivery trip.

Attributes

- destinationNode: Node - Node for destination point of a route.
- departuteNode: Node - Node for departure of a route.
- cost: double - Calculated recommended cost for delivery trip.
- length: long - Road aligned length of the route.

Methods

+ calculateLength(): void - Calculates road-alligned distance between nodes.

Transport

A session of a delivery maintained by backend

Attributes

- couriers: Map<Courier> - List of couriers that is responsible for a single transport operation.
- customer: Customer - Owner of the goods.
- cost: double - Calculated cost for transport.
- routes: Map<Routes> - List of routes for representing merged routes.
- vehicle: Vehicle - Vehicle responsible for transport operation.
- cargos: Map<DeliveryGood> - List of cargos.
- currentPosition: Node - Current position of courier carrying the cargo.
- feedbacks: Map<Feedback> - List of feedbacks given to single transport.
- verified: bool - Status of verification.
- status: int - Status of transport (ready, ongoing, unmatched, matched, offered).
- report: Report - Report object after finalizing the transport.
- transaction: Transaction - Accounting information about transport.

Methods

+ cancelDelivery(): bool - Cancels delivery that is not started yet.
+ getCurrencyPosition(type: String): Node - Returns current position in relative route position or absolute position.
+ getDeliveryReport(): Report - Returns delivery report for the transport.
+ notifyCustomer(): bool - Pushes notification to user’s notification queue.
+ addReport(rep: Report): bool, Adds report to transportation.

3.2.1.2 AccountingManager

Account

Single account class for representing user accounting information.

Attributes

+ balance: int - Current balance of a user.
+ transactions: ArrayList<Transaction> - List of recent transaction of a user.
+ paymentMethod: String - Stored payment methods for user.

Methods

+ withdrawBalance(amount: int): bool - Withdraws specific amount from user balance.
+ depositBalance(amount: int): bool - Deposits specific amount to user balance.

Customer

Class for representing cargo owners.

Attributes

+ activeOrders: ArrayList<Transport> - Orders that currently ongoing.
+ givenFeedbacks: ArrayList<Feedback> - List of feedbacks that is given by customer.

Methods

+ findCourier(good: DeliveryGood): ArrayList<Transport> - Returns lists of transport offerings.
+ addCargo(good: DeliveryGood): bool - Adds a cargo offering to the system.
+ showOffers(good: DeliveryGood): ArrayList<Transport> - Returns offerings calculated for customer.
+ acceptRoute(route: Route, trans: Transaction): bool - Accepts specific route for the customer.
+ acceptCourier(route: Route): bool - Accept route from an offered transport.
+ calculateCost(transport: Transport): double - Calculates cost of selected transport.
+ showLocation(transport: Transport): Node - Show location of specific ongoing transport operation.
+ addFeedback(trans: Transport, cour: Couriers): bool - Adds feedback to a courier.
+ listMyGoods(): ArrayList<DeliveryGood> - Lists all enlisted goods.

Transaction

Class for transactions for transport operations.

Attributes

+ status: String - Whether it is an revenue or a cost.

+ amount: double - Amount of balance it involves.
+ recipient: User - Receiver of the transaction.
+ sender: User - Actor of the transaction.

Methods

+ getAmount(): double - Return current balance of a user.
+ cancelTransaction(): void - Cancels any unconfirmed transaction.
+ completeTransaction(): void - Confirms an unconfirmed transaction.

User <<Interface>>

User interface for Customers, Couriers and Admin classes.

Attributes

+ name: String - Name of the user.
+ password:String - Password of the user (encrypted)
+ age: short - Age of the user.
+ badges: ArrayList<Badge> - Badges that user collected so far.
+ type: int - Type of the user (Admin, Customer, Courier)
+ documents: ArrayList<String> - email: String - List of documents that used for registration.

Methods

3.2.1.3 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

+ updateScore(): void - Updates score of the courier.
+ listOffers(): ArrayList<Transport> - List of transportation offers.
+ startShareLocation(): bool - Starts emitting current position for the courier.

+ calculateCost(trans: Transport, vehicle: Vehicle): void - Calculated recommended trip cost.
+ acceptRoute(isAccepted: bool, trans: Transport): bool - Accept provided transport.
+ addDocument(doc: Document): bool - Adds document about courier.
+ startTransport(trans: Transport): bool - Starts a transport process.
+ finishTransport(trans: Transport): bool - Finalizes a transport.
+ acceptTransportRoute(trans: Transport, route: Route): bool - Accept given route.

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 is 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 is 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.

3.2.1.4 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 alligned for given DirectionRequest instance and calls callback function

DirectionsRequest

Request class for parametizing 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

3.2.2 Storage

3.2.2.1 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.

MongoDB
Holds data about compromise requests for both drivers and package owners.
Attributes
+ host: String - Name of the host + username: String - Name of the user + password: String - Password of the user + database: String - Name of the database
Methods
+ connectToCluster(uri: String): void - Establish connection. + insertOne(message: Object): boolean - Insert new request and return a boolean on the basis of success of the method.

+ authentication(username:String, password: String): boolean - Check credentials and return a boolean according to credentials matching or not. + getCollection(auth: boolean): List - Returns the list of the requests.

3.2.2.2 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.

3.2.2.3 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.
+ mongoConnection: MongoDB - Mongodb database.
+ restApi: RequestHandler - Starts request handler.
+ database: String - Name of the database.
+ sqlUserName: String - Username for MySQL.
+ mongoUserName: String - Username for MongoDB.
+ sqlPassword: Spring - Password for MySQL.
+ mongoPassword: String - Password for MongoDB.

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. References

[1] “IEEE Reference Guide - IEEE Author Center.” [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. [Accessed: 26-Feb-2022].

[2] “What is UML,” *What is UML | Unified Modeling Language*. [Online]. Available: https://www.uml.org/what-is-uml.htm. [Accessed: 26-Feb-2022].

[3] “The Application Data Platform,” *MongoDB*. [Online]. Available: https://www.mongodb.com/. [Accessed: 26-Feb-2022].