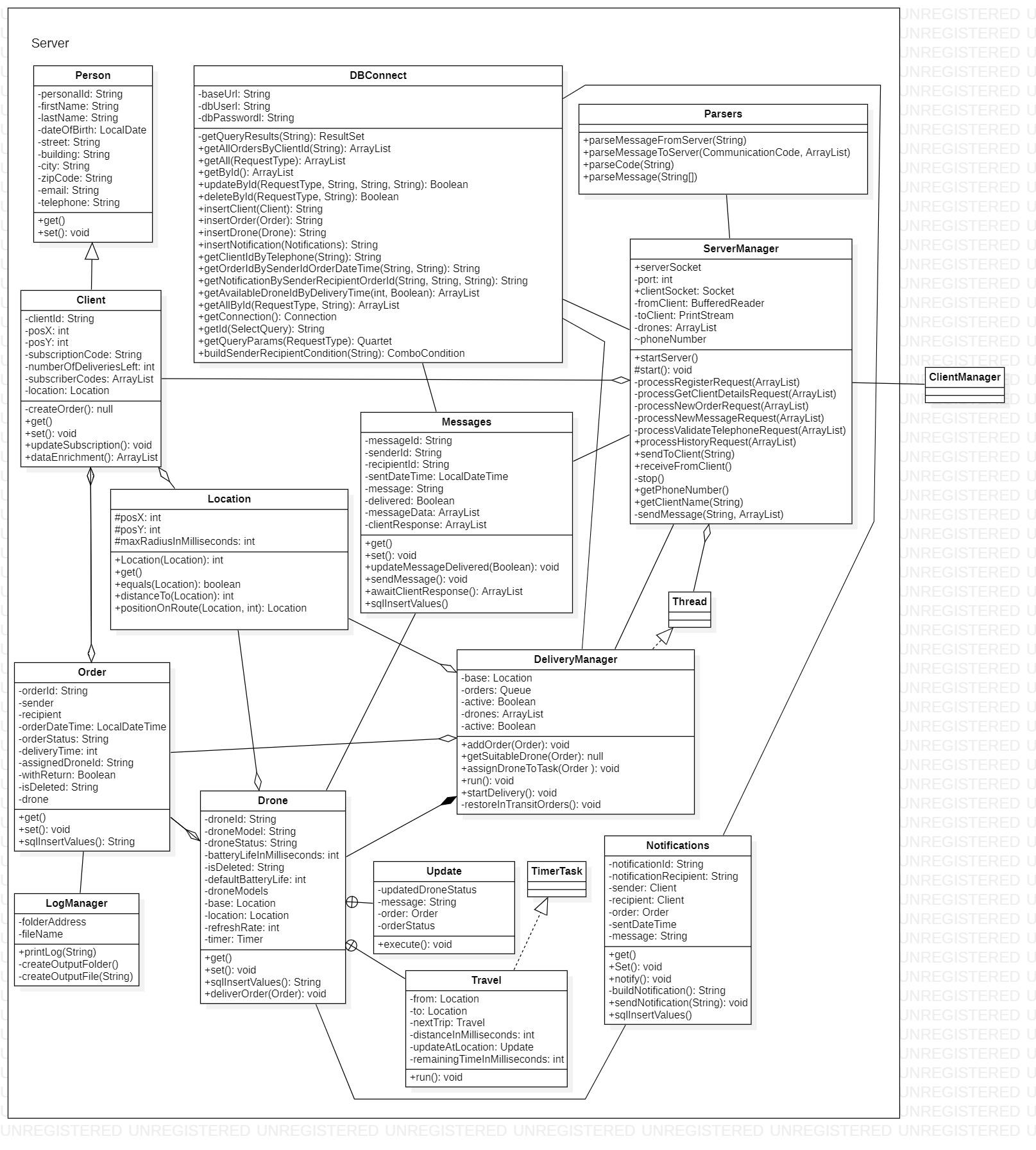
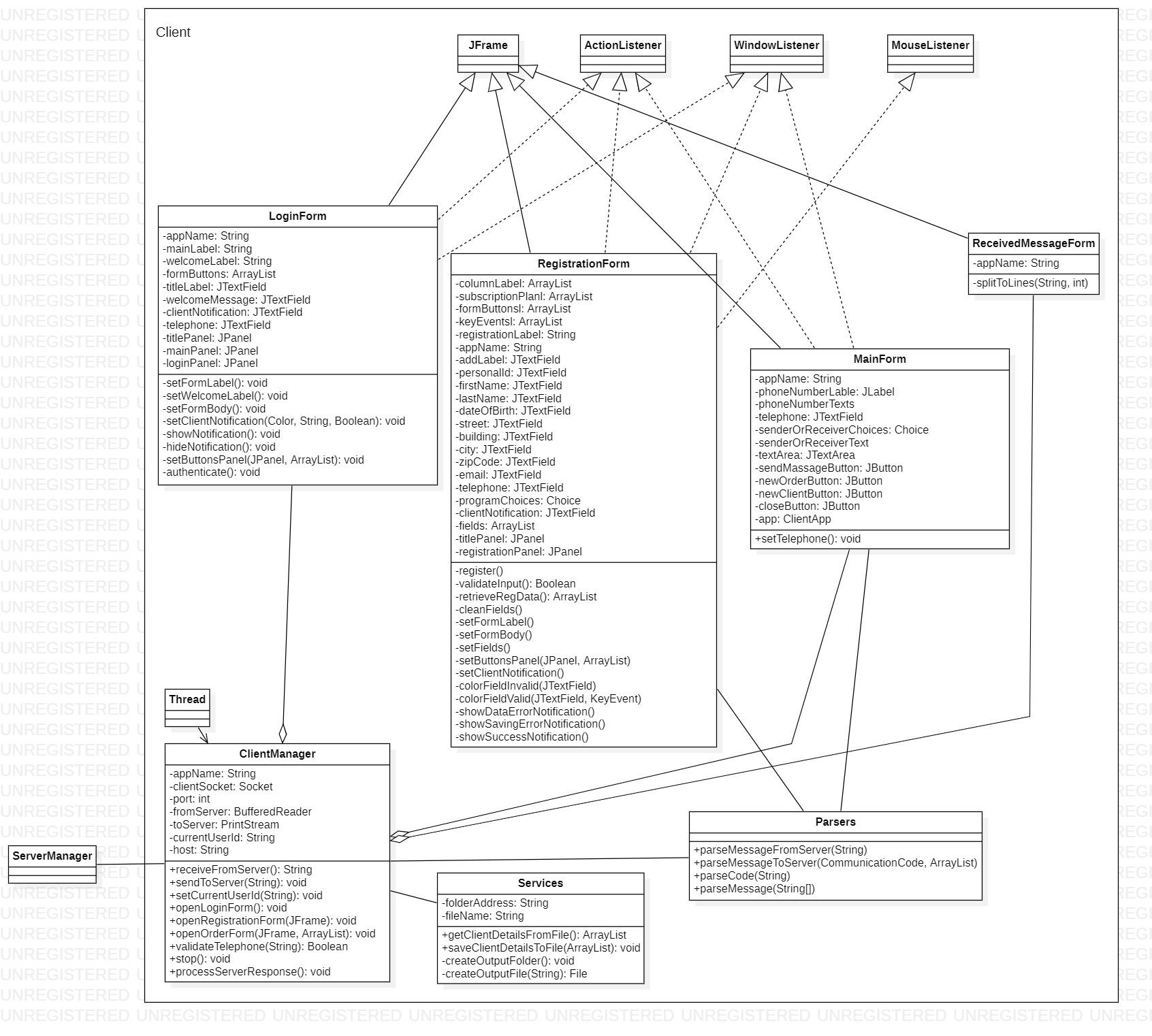
DronePost project OOP class

* The system is built from two projects: DronePost, which contains the server code, and DronePostUI, which contains the user code.
* The SQL code is located in the TableScript.sql file in the DataBaseScript folder in the DronePost project and in this file.
* Delivery times are calculated with the assumption that every millisecond simulates a second (for example, the battery time on the computer is 10,800 milliseconds instead of 10,800 seconds).
* The server initializes 20 drones and saves them in the database. If drones already exist in the database, it uses those.
* The status of orders (when the delivery starts, when it is delivered) is saved in the serverLog.txt file in the data folder in the DronePost project.
* Data validation from the client is performed both in the client application and on the server side.
* i used the intelliJ IDE

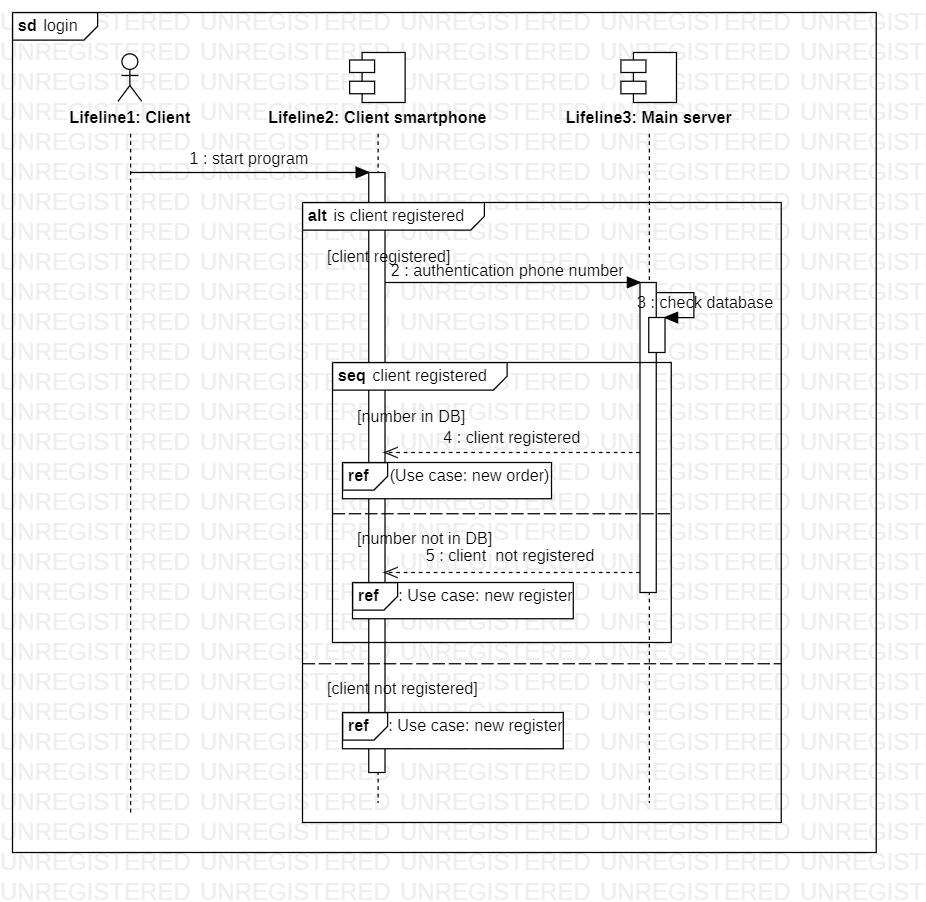
Class Diagram Server



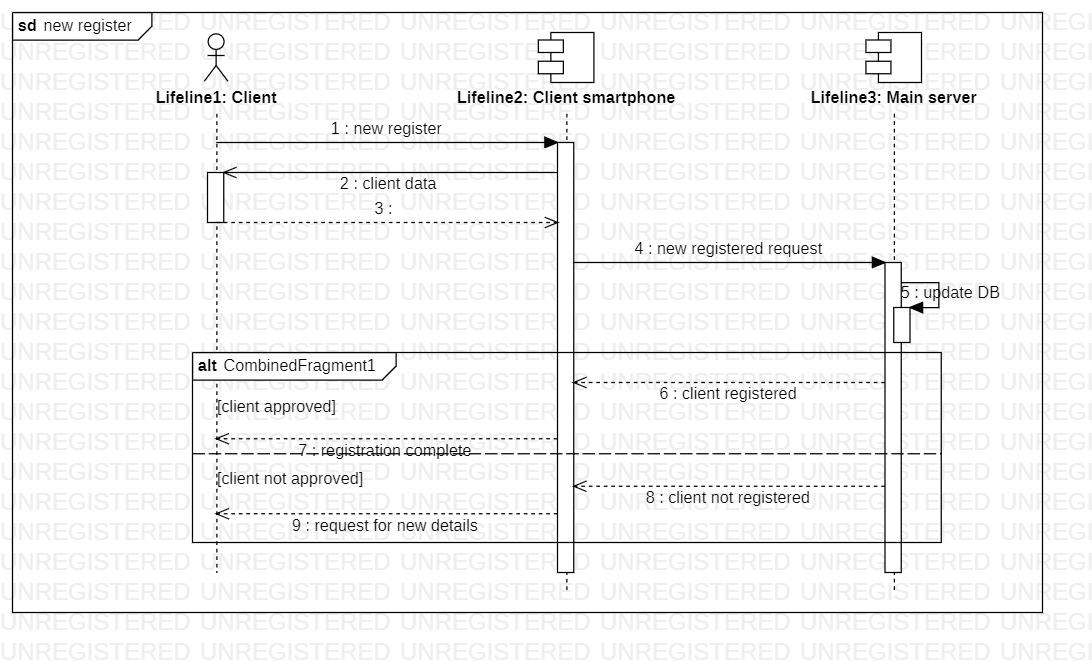
Class Diagram Client



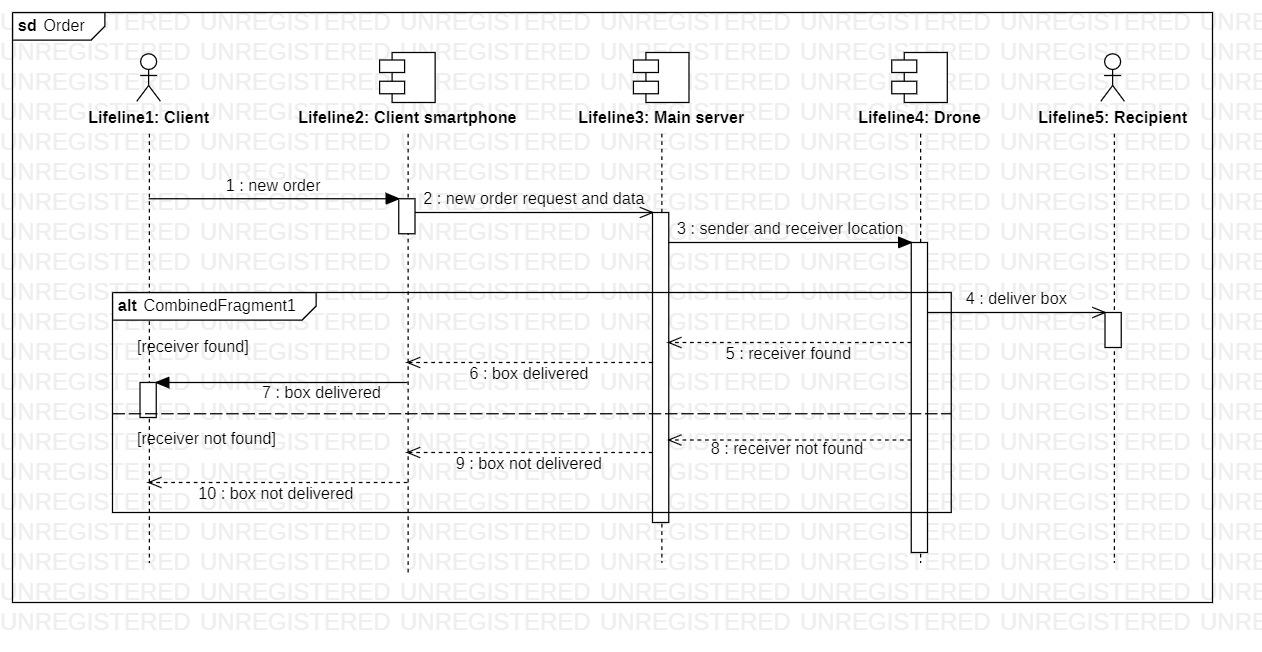
Sequence Diagram 1. LogIn



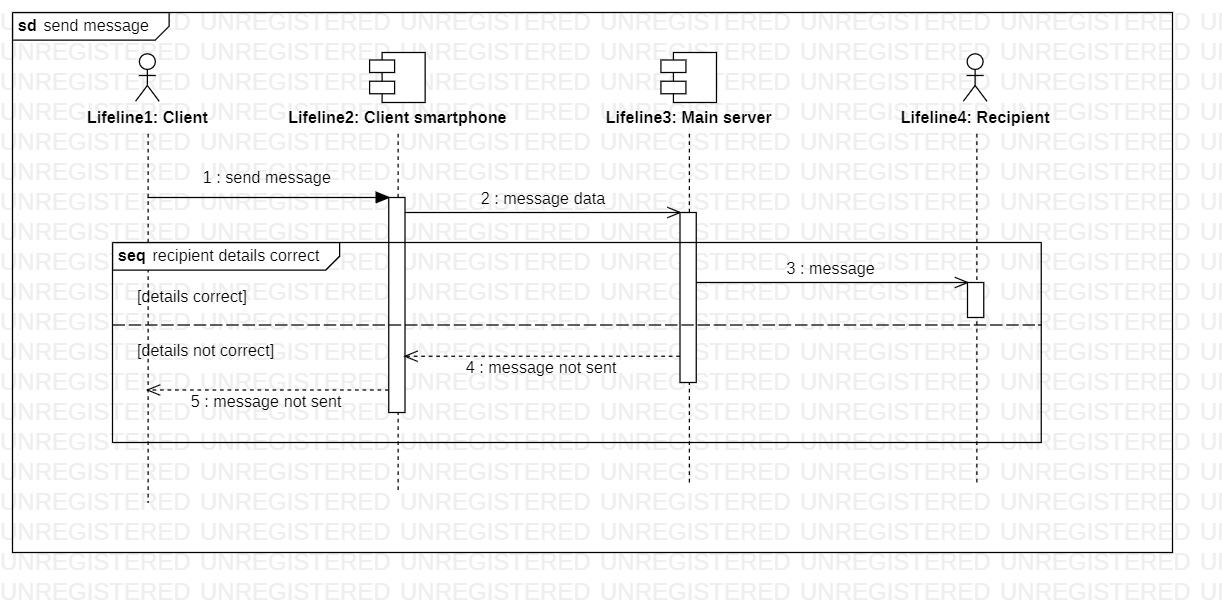
Sequence Diagram 2. New register



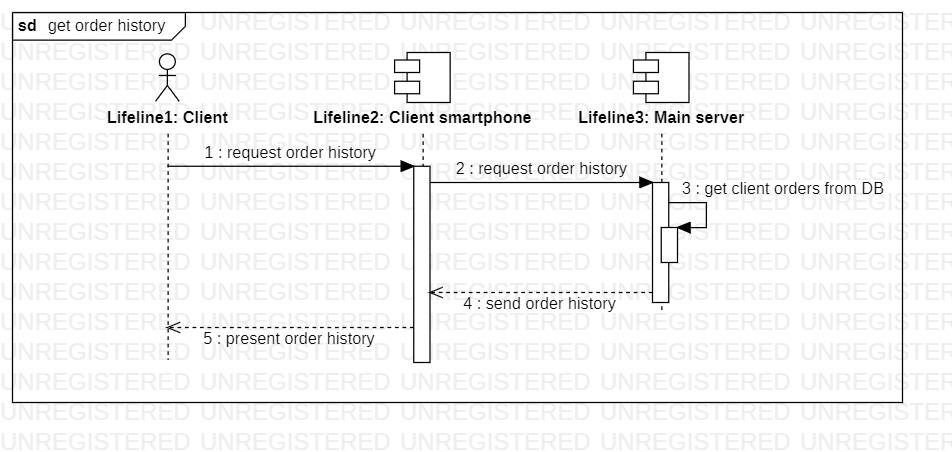
Sequence Diagram 3. Order



Sequence Diagram 4. Send message



Sequence Diagram 5. Get order history



DronePost (server ) classes:

package Enums.Fields;

import java.util.ArrayList;

import java.util.Arrays;

public enum ClientFields {

ClientId("clientId"),

PersonalId("personalId"),

FirstName("firstName"),

LastName("lastName"),

DateOfBirth("dateOfBirth"),

Street("street"),

Building("building"),

City("city"),

ZipCode("zipCode"),

Email("email"),

Telephone("telephone"),

SubscriptionCode("subscriptionCode"),

PosX("posX"),

PosY("posY"),

NumberOfDeliveriesLeft("numberOfDeliveriesLeft"),

IsDeleted("isDeleted");

private String name;

ClientFields(String name) { this.name = name; }

public static ArrayList<String> personColumns(){

ArrayList<String> columns = new ArrayList<>();

ClientFields[] person = Arrays.copyOfRange(values(),1, ClientFields.values().length-2);

for (ClientFields cf : person){

columns.add(cf.getValue());

}

return columns;//Arrays.asList(person.).forEach((item) -> item.getValue());

}

public static ClientFields[] sqlInsertColumns(){

return Arrays.copyOfRange(values(),1, ClientFields.values().length-1);

}

public String getValue(){

return this.name;

}

}

package Enums.Fields;

import java.util.Arrays;

public enum DroneFields {

DroneId("droneId"),

DroneModel("droneModel"),

DroneStatus("droneStatus"),

IsDeleted("isDeleted");

private String name;

DroneFields(String name) { this.name = name; }

public static DroneFields[] sqlInsertColumns(){

return Arrays.copyOfRange(values(),0, DroneFields.values().length-1);

}

public String getValue(){

return this.name;

}

}

package Enums.Fields;

import java.util.Arrays;

public enum MessageFields {

MessageId("messageId"),

SenderId("senderId"),

RecipientId("recipientId"),

SentDateTime("sentDateTime"),

Message("message"),

Delivered("delivered");

private String name;

MessageFields(String name) { this.name = name; }

public static MessageFields[] sqlInsertColumns(){

return Arrays.copyOfRange(values(),1, MessageFields.values().length-1);

}

public String getValue(){

return this.name;

}

}

package Enums.Fields;

import java.util.Arrays;

public enum NotificationFields {

NotificationId("notificationId"),

RecipientId("recipientId"),

OrderId("orderId"),

SentDateTime("sentDateTime"),

Message("message"),

Delivered("delivered");

private String name;

NotificationFields(String name) { this.name = name; }

public static NotificationFields[] sqlInsertColumns(){

return Arrays.copyOfRange(values(),1, NotificationFields.values().length-1);

}

public String getValue(){

return this.name;

}

}

package Enums.Fields;

import java.util.Arrays;

public enum OrderFields {

OrderId("orderId"),

SenderId("senderId"),

RecipientId("recipientId"),

OrderDateTime("orderDateTime"),

OrderStatus("orderStatus"),

AssignedDroneId("assignedDroneId"),

WithReturn("withReturn"),

IsDeleted("isDeleted");

private String name;

OrderFields(String value) { this.name = value; }

public static OrderFields[] sqlInsertColumns(){

return Arrays.copyOfRange(values(),1, OrderFields.values().length-1);

}

public String getValue(){

return this.name;

}

}

package Enums.Status;

public enum DroneStatus {

Available("available"),

InTransit("inTransit");

private final String name;

DroneStatus(String name) { this.name = name; }

public String getValue(){

return this.name;

}

public static DroneStatus getStatus(String name){

for(DroneStatus value : values()){

if (value.getValue().equals(name))

return value;

}

return null;

}

}

package Enums.Status;

public enum OrderStatus {

Paid("paid"),

InTransit("inTransit"),

Delivered("delivered");

private final String name;

OrderStatus(String name) { this.name = name; }

public String getValue(){

return this.name;

}

public static OrderStatus getStatus(String name){

for(OrderStatus value : values()){

if (value.getValue().equals(name))

return value;

}

return null;

}

}

package Enums.Types;

public enum NotificationType {

Delivered(" has been delivered."),

Sent(" is on the way");

private String name;

NotificationType(String name){ this.name = name; }

public String getValue(){

return this.name;

}

public static NotificationType getNotification(String name){

for(NotificationType value : values()){

if (value.getValue().equals(name))

return value;

}

return null;

}

}

package Enums.Types;

public enum RequestType {

CLIENTS("dronepost.clients"),

ORDERS("dronepost.orders"),

DRONES("dronepost.drones"),

NOTIFICATIONS("dronepost.notifications"),

MESSAGES("dronepost.messages");

private String tableName;

RequestType(String tableName) {this.tableName = tableName; }

public String getTableName() {

return tableName;

}

}

package Enums;

import java.util.\*;

public enum CommunicationCode {

RegisterClient("1"),

ClientRegistered("2"),

ClientAlreadyExists("3"),

ClientSavingError("4"),

ValidateTelephone("5"),//Server to ClientApp

GetClientDetails("6"),

NewOrder("7"),//ClientApp to Server

NewMessage("8"),//ClientApp to Server

ReceiverNotConnectedToServer("9"),//Server to ClientApp

MessageDelivered("10"),//Server to ClientApp(Receiver ClientApp)

OrderDelivered("11"),//Server to ClientApp (Sender)

OrderSent("12"),

ServerProcessingError("13"),

MessageReceived("14"),

HistoryRequest("15"),

HistoryReport("16");

private String codeId;

CommunicationCode (String codeId) {

this.codeId = codeId;

}

private static final Map<String,CommunicationCode> lookup = new HashMap<>();

static {

for(CommunicationCode cc : EnumSet.allOf(CommunicationCode.class))

lookup.put(cc.getCodeId(), cc);

}

public String getCodeId(){

return this.codeId;

}

public static ArrayList<String> codes(){

var codes = new ArrayList<String>();

for (CommunicationCode cc : CommunicationCode.values()) {

codes.add(cc.getCodeId());

}

return codes;

}

public static CommunicationCode getCode(String codeId){

return lookup.get(codeId);

}

}

package Enums;

public enum NotificationAddressee {

SENDER,

RECIPIENT

}

package OrderManagement;

import Enums.Fields.DroneFields;

import Enums.NotificationAddressee;

import Enums.Status.OrderStatus;

import Enums.Types.NotificationType;

import Services.DBConnect;

import Enums.Status.DroneStatus;

import Enums.Types.RequestType;

import Services.LogManager;

import org.javatuples.Pair;

import java.util.\*;

public class Drone {

private String droneId;

private String droneModel;

private String droneStatus;

private Order currentOrder;

private int batteryLifeInMilliseconds;

private String isDeleted = "0";

final int defaultBatteryLife = 3 \* 60 \* 60;

private final ArrayList<String> droneModels = new ArrayList<>(Arrays.asList("ModelA", "ModelB", "ModelC", "ModelD"));

private Location base;

private Location location;

private int refreshRate = 100;

private Timer timer;

public Drone() {

}

public Drone(String droneId, DroneStatus status) {

setDroneId(droneId);

setModel(null);

this.droneStatus = status.getValue();

this.location = this.base = new Location(0, 0);

this.batteryLifeInMilliseconds = 1000 \* defaultBatteryLife;

}

public Drone(ArrayList<String> droneDetails) {

try {

setDroneId(droneDetails.get(0));

setModel(droneDetails.get(1));

setDroneStatus(DroneStatus.getStatus(droneDetails.get(2)), false);

this.location = this.base = new Location(0, 0);

this.batteryLifeInMilliseconds = 1000 \* defaultBatteryLife;

setIsDeleted(droneDetails.get(3));

} catch (IllegalArgumentException e) {

e.printStackTrace();

}

}

public String getDroneId() {

return droneId;

}

public String getDroneModel() {

return droneModel;

}

public String getDroneStatus() {

return droneStatus;

}

public Location getDroneLocation() {

return location;

}

public int getBatteryLifeInMilliseconds() {

return batteryLifeInMilliseconds;

}

public String getIsDeleted() {

return isDeleted;

}

public void setDroneId(String droneId) {

this.droneId = droneId;

}

public void setModel(String droneModel) throws IllegalArgumentException {

if (droneModel == null || droneModel.isEmpty())

this.droneModel = droneModels.get(new Random().nextInt(4));

else if (!droneModels.contains(droneModel))

throw new IllegalArgumentException("Unknown drone model");

else this.droneModel = droneModel;

}

public void setDroneStatus(DroneStatus droneStatus, Boolean updateDB) throws IllegalArgumentException {

if (droneStatus == null)

throw new IllegalArgumentException("Non-existing drone status provided.");

else {

this.droneStatus = droneStatus.getValue();

if (updateDB) {

var param = new Pair<>(DroneFields.DroneStatus.getValue(), this.droneStatus);

DBConnect.updateById(RequestType.DRONES, this.getDroneId(), new ArrayList<>(Arrays.asList(param)));

}

}

}

private void setIsDeleted(String isDeleted) throws IllegalArgumentException {

if (!Arrays.asList("0", "1").contains(isDeleted))

throw new IllegalArgumentException("Incorrect isDeleted status.");

this.isDeleted = isDeleted;

}

public String[] sqlInsertValues() {

return Arrays.asList(

getDroneId(), getDroneModel(), getDroneStatus()).toArray(new String[0]);

}

public ArrayList<Drone> getAllDrones() {

ArrayList<ArrayList<String>> dronesData = DBConnect.getAll(RequestType.DRONES);

ArrayList<Drone> drones = new ArrayList<>();

int numberOfDrones = dronesData.size();

if (numberOfDrones >= 20)

dronesData.forEach(data -> drones.add(new Drone(data)));

int retryCounter = 0;

while (numberOfDrones < 21 && retryCounter < 3) {

Drone drone = new Drone(String.valueOf(++numberOfDrones), DroneStatus.Available);

String insertResult = DBConnect.insertDrone(drone);

switch (insertResult) {

case "1" -> drones.add(drone);

case "0" -> {

System.out.println("Writing to database error.");

++retryCounter;

}

case "-1" -> {

System.out.println("Connection to database error.");

++retryCounter;

}

}

}

if (retryCounter == 3)

System.out.println("Error creating drones.");

return drones;

}

public void deliverOrder(Order order) {

timer = new Timer();

currentOrder = order;

Location sender = order.getSender().getLocation();

Location recipient = order.getRecipient().getLocation();

String message0 = ": Drone (id " + currentOrder.getDrone().getDroneId() + ") is on it's way to pick up the order #"

+ currentOrder.getOrderId() + " from " + currentOrder.getSender().getFullName();

String message1 = ": Drone (id " + currentOrder.getDrone().getDroneId() + ") has picked up the order #"

+ currentOrder.getOrderId() + " from " + currentOrder.getSender().getFullName();

String message2 = ": Drone (id " + currentOrder.getDrone().getDroneId() + ") has delivered the order #"

+ currentOrder.getOrderId() + " to " + currentOrder.getRecipient().getFullName();

String message3 = ": Drone (id " + currentOrder.getDrone().getDroneId() + ") has arrived at base and is fully charged";

var onWayToSender = new Update(DroneStatus.InTransit, message0, currentOrder, OrderStatus.Paid);

var packagePickedUp = new Update(DroneStatus.InTransit, message1, currentOrder, OrderStatus.InTransit);

var packageDelivered = new Update(DroneStatus.Available, message2, currentOrder, OrderStatus.Delivered);

var backAtBase = new Update(DroneStatus.Available, message3, currentOrder, null);

var toBase = new Travel(recipient, this.base, backAtBase, null);

var toRecipient = new Travel(sender, recipient, packageDelivered, toBase);

var toSender = new Travel(this.location, sender, packagePickedUp, toRecipient);

onWayToSender.execute();

timer.schedule(toSender, 0, refreshRate);

}

class Update {

private final DroneStatus updatedDroneStatus;

private final String message;

private final Order order;

private final OrderStatus updatedOrderStatus;

public Update(DroneStatus updatedDroneStatus, String message, Order order, OrderStatus updatedOrderStatus) {

this.updatedDroneStatus = updatedDroneStatus;

this.message = message;

this.order = order;

this.updatedOrderStatus = updatedOrderStatus;

}

public void execute() {

if (!getDroneStatus().equals(this.updatedDroneStatus.getValue()))

setDroneStatus(this.updatedDroneStatus, true);

if (location.equals(base))

batteryLifeInMilliseconds = 1000 \* defaultBatteryLife;

if (updatedOrderStatus != null && !order.getOrderStatus().equals(updatedOrderStatus)) {

order.setOrderStatus(updatedOrderStatus, true);

switch (updatedOrderStatus) {

case InTransit -> {

new Thread(() -> new Notifications(order).notify(NotificationAddressee.SENDER, NotificationType.Sent)).start();

new Thread(() -> new Notifications(order).notify(NotificationAddressee.RECIPIENT, NotificationType.Sent)).start();

}

case Delivered -> {

new Thread(() -> new Notifications(order).notify(NotificationAddressee.SENDER, NotificationType.Delivered)).start();

new Thread(() -> new Notifications(order).notify(NotificationAddressee.RECIPIENT, NotificationType.Delivered)).start();

}

}

}

LogManager.printLog(new Date() + message + " ... battery left = " + batteryLifeInMilliseconds);

}

}

class Travel extends TimerTask {

private final Location from;

private final Location to;

private final Travel nextTrip;

private final int distanceInMilliseconds;

private final Update updateAtLocation;

private int remainingTimeInMilliseconds;

public Travel(Location from, Location to, Update updateAtLocation, Travel nextTrip) {

this.from = from;

this.to = to;

this.remainingTimeInMilliseconds = this.distanceInMilliseconds = from.distanceTo(to);

this.updateAtLocation = updateAtLocation;

this.nextTrip = nextTrip;

}

public void run() {

if (remainingTimeInMilliseconds > 0) {

location = from.positionOnRoute(to, distanceInMilliseconds - remainingTimeInMilliseconds);

remainingTimeInMilliseconds -= refreshRate;

} else if (nextTrip != null) {

this.cancel();

location = to;

this.updateAtLocation.execute();

timer.schedule(this.nextTrip, 0, refreshRate);

} else {

this.cancel();

if (droneStatus.equals(DroneStatus.Available.getValue())) {

location = to;

this.updateAtLocation.execute();

timer.cancel();

timer.purge();

}

}

batteryLifeInMilliseconds -= refreshRate;

}

}

}

package OrderManagement;

import java.util.Random;

public class Location {

protected int posX;

protected int posY;

public Location() {

int maxRadiusInMilliseconds = 7200;

this.posX = new Random().nextInt(2 \* maxRadiusInMilliseconds) - maxRadiusInMilliseconds; //random in range

int maxY = (int)Math.sqrt(Math.pow(maxRadiusInMilliseconds,2)-Math.pow(this.posX,2));

this.posY = new Random().nextInt(2 \* maxY ) - maxY ; //random in range

}

public Location(int posX, int posY) {

this.posX = posX;

this.posY = posY;

}

public int getPosX(){

return this.posX;

}

public int getPosY(){

return this.posY;

}

public boolean equals(Location position) {

return (posX == position.posX && posY == position.posY);

}

public int distanceTo(Location p) {

double x0 = posX;

double x1 = p.posX;

double y0 = posY;

double y1 = p.posY;

return (int)(Math.sqrt(Math.pow(x0 - x1, 2) + Math.pow(y0 - y1, 2)));

}

public Location positionOnRoute(Location p, int distancePassed) {

double distanceTotal = distanceTo(p);

double x0 = posX;

double x1 = p.posX;

double y0 = posY;

double y1 = p.posY;

int x2 = (int)( ((double)distancePassed/distanceTotal) \* (x1 - x0) + x0);

int y2 = (int)( ((double)distancePassed/distanceTotal) \* (y1 - y0) + y0);

return new Location(x2, y2);

}

}

package OrderManagement;

import Enums.Fields.NotificationFields;

import Enums.Types.\*;

import Services.Parsers;

import Services.DBConnect;

import Enums.\*;

import Services.ServerManager;

import UserManagement.Client;

import org.javatuples.Pair;

import java.time.LocalDateTime;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Locale;

public class Notifications {

private String notificationId;

private String notificationRecipient;

private Client sender;

private Client recipient;

private Order order;

private LocalDateTime sentDateTime;

private String message;

public Notifications(Order order) {

setSender(order.getSender());

setRecipient(order.getRecipient());

setOrder(order);

}

public String getNotificationId() {

return this.notificationId;

}

public String getNotificationRecipient() {

return notificationRecipient;

}

public String getSenderId() {

return this.sender.getClientId();

}

public String getRecipientId() {

return this.recipient.getClientId();

}

public String getOrderId() {

return this.order.getOrderId();

}

public LocalDateTime getSentDateTime() {

return this.sentDateTime;

}

public String getMessage() {

return this.message;

}

public void setNotificationId(String notificationId) {

this.notificationId = notificationId;

}

public void setNotificationRecipient(String notificationRecipient) {

this.notificationRecipient = notificationRecipient;

}

public void setSender(Client sender) throws IllegalArgumentException {

if (Integer.parseInt(sender.getClientId()) < 1)

throw new IllegalArgumentException("Non-existing sender, order ID is invalid.");

this.sender = sender;

}

public void setRecipient(Client recipient) throws IllegalArgumentException {

if (Integer.parseInt(recipient.getClientId()) < 1)

throw new IllegalArgumentException("Non-existing recipient, order ID is invalid.");

this.recipient = recipient;

}

public void setOrder(Order order) throws IllegalArgumentException {

if (Integer.parseInt(order.getOrderId()) < 1)

throw new IllegalArgumentException("Non-existing order, order ID is invalid.");

this.order = order;

}

public void setSentDateTime(LocalDateTime sentDateTime) throws IllegalArgumentException {

if (sentDateTime.isAfter(LocalDateTime.now()))

throw new IllegalArgumentException("Incorrect order Date/Time provided.");

this.sentDateTime = sentDateTime.truncatedTo(ChronoUnit.SECONDS);

}

public void setMessage(String message) throws IllegalArgumentException {

if (message.length() > 500)

throw new IllegalArgumentException("Message length is over 500 symbols.");

this.message = message;

}

/\*\*

\* Send notification to the sender

\* @param addressee value from NotificationAddressee enum

\* @param type value from NotificationType enum

\*/

public void notify(NotificationAddressee addressee, NotificationType type) {

try {

String message = buildNotification(addressee, type);

switch (addressee){

case SENDER -> setNotificationRecipient(getSenderId());

case RECIPIENT -> setNotificationRecipient(getRecipientId());

}

CommunicationCode code = null;

switch (type) {

case Sent -> code = CommunicationCode.OrderSent;

case Delivered -> code = CommunicationCode.OrderDelivered;

}

setSentDateTime(LocalDateTime.now());

setMessage(message);

String id = DBConnect.insertNotification(this);

System.out.println(id);

setNotificationId(id);

if (code != null)

message = Parsers.parseMessageToClient(code, new ArrayList<>(Arrays.asList(getNotificationRecipient(), message)));

sendNotification(message);

} catch (IllegalArgumentException e) {

e.printStackTrace();

}

}

/\*\*

\* Builds the notification message

\* @param addressee a value from NotificationAddressee enum

\* @param type value from NotificationType enum

\* @return notification message as String

\* @throws IllegalArgumentException is thrown in case we try to send the recipient notifications on events other that Sent or Delivered

\*/

private String buildNotification(NotificationAddressee addressee, Enums.Types.NotificationType type) throws IllegalArgumentException {

if (addressee.equals(NotificationAddressee.RECIPIENT) && !Arrays.asList(NotificationType.Delivered, NotificationType.Sent).contains(type))

throw new IllegalArgumentException("Sending this type of notification to order recipient is not allowed: " + type.toString());

Client messageAddressee = null;

switch (addressee) {

case SENDER -> messageAddressee = this.sender;

case RECIPIENT -> messageAddressee = this.recipient;

}

String message = messageAddressee.getFullName().toUpperCase(Locale.ROOT) + " hello, Order #" + order.getOrderId() + type.getValue();

if (addressee.equals(NotificationAddressee.SENDER) && type.equals(NotificationType.Sent))

message = message.concat(" to " + recipient.getFullName());

if (type.equals(NotificationType.Sent)) {

LocalDateTime deliveryDate = LocalDateTime.now().plusSeconds(order.getDeliveryTime()/1000 + 1);

message = message.concat(". Estimated delivery: " + Parsers.dateTimeConverter(deliveryDate, true) + ".");

}

return message;

}

/\*\*

\* Sends the notification via configured channel (SMS, Email)

\* @param message test message that needs to be sent

\*/

private void sendNotification(String message) {

ServerManager.sendToClient(message); //send notification via SMS API

var param = new Pair<>(NotificationFields.Delivered.getValue(), "1");

DBConnect.updateById(RequestType.NOTIFICATIONS, getNotificationId(), new ArrayList<>(Arrays.asList(param)));

}

public String[] sqlInsertValues() {

return Arrays.asList(

getNotificationRecipient(), getOrderId(), getSentDateTime().toString(), getMessage()

).toArray(new String[0]);

}

}

package OrderManagement;

import Enums.Fields.OrderFields;

import Services.Parsers;

import Services.DBConnect;

import Enums.Status.OrderStatus;

import Enums.Types.RequestType;

import UserManagement.Client;

import org.javatuples.Pair;

import java.sql.SQLException;

import java.time.LocalDateTime;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

import java.util.Arrays;

public class Order {

private String orderId;

private Client sender;

private Client recipient;

private LocalDateTime orderDateTime;

private OrderStatus orderStatus;

private int deliveryTime;

private String assignedDroneId = "-1";

private Boolean withReturn;

private String isDeleted;

private Drone drone = null;

public Order(Client sender, Client recipient, LocalDateTime orderDateTime, OrderStatus orderStatus, Boolean withReturn) {

try {

setSenderRecipient(sender, recipient);

setOrderDateTime(orderDateTime);

setOrderStatus(orderStatus,false);

} catch (IllegalArgumentException e){

e.printStackTrace();

}

this.withReturn = withReturn;

this.isDeleted = "0";

}

public Order (ArrayList<String> order){

try{

setOrderId(order.get(0));

Client sender = new Client(DBConnect.getById(RequestType.CLIENTS, order.get(1)), false);

Client recipient = new Client(DBConnect.getById(RequestType.CLIENTS, order.get(2)), false);

setSenderRecipient(sender, recipient);

setOrderDateTime(order.get(3));

setOrderStatus(OrderStatus.getStatus(order.get(4)), false);

setAssignedDroneId(order.get(5), false);

setWithReturn(order.get(6));

setIsDeleted(order.get(7));

} catch (IllegalArgumentException e){

e.printStackTrace();

}

}

public String getOrderId(){ return orderId; }

public String getSenderId() { return sender.getClientId(); }

public String getRecipientId() { return recipient.getClientId(); }

public Drone getDrone(){

return this.drone;

}

public Client getSender(){ return this.sender;}

public Client getRecipient(){ return this.recipient;}

public LocalDateTime getOrderDateTime() { return orderDateTime; }

public OrderStatus getOrderStatus() { return orderStatus; }

public int getDeliveryTime() { return deliveryTime; }

public String getAssignedDroneId() {

return assignedDroneId;

}

public Boolean getWithReturn() { return withReturn; }

public void setDrone(Drone drone){

this.drone = drone;

try {

setAssignedDroneId(this.drone.getDroneId(), true);

} catch (IllegalArgumentException iae) {System.out.println(iae.getMessage());}

}

public void setOrderId(String orderId){

this.orderId = orderId;

}

public void setSenderRecipient(Client sender, Client recipient) throws IllegalArgumentException {

if (sender.getClientId().equals("-1")

|| recipient.getClientId().equals("-1")

|| sender.getClientId().equals(recipient.getClientId())

)

throw new IllegalArgumentException("Incorrect sender and/or recipient provided.");

else { this.sender = sender; this.recipient = recipient; }

}

public void setOrderDateTime (LocalDateTime orderDateTime) throws IllegalArgumentException{

if (orderDateTime.isAfter(LocalDateTime.now()))

throw new IllegalArgumentException("Incorrect order Date/Time provided.");

this.orderDateTime = orderDateTime.truncatedTo(ChronoUnit.SECONDS);

}

public void setOrderDateTime (String orderDateTime) {

setOrderDateTime(Parsers.dateTimeConverter(orderDateTime));

}

public void setOrderStatus(OrderStatus orderStatus, Boolean updateDB) throws IllegalArgumentException {

if (orderStatus == null)

throw new IllegalArgumentException("Non-existing order status provided.");

else {

this.orderStatus = orderStatus;

if (updateDB) {

var param = new Pair<>(OrderFields.OrderStatus.getValue(), getOrderStatus().getValue());

DBConnect.updateById(RequestType.ORDERS, this.getOrderId(), new ArrayList<>(Arrays.asList(param)));

}

}

}

public void setDeliveryTime(int deliveryTime) throws IllegalArgumentException {

if (deliveryTime > 0)

this.deliveryTime = deliveryTime;

else throw new IllegalArgumentException("Incorrect delivery time.");

}

public void setAssignedDroneId(String assignedDroneId, Boolean updateDB) throws IllegalArgumentException {

int id = Integer.parseInt(assignedDroneId);

if (id == -1 || (id > 0 && id < 21)) {

this.assignedDroneId = assignedDroneId;

if (updateDB) {

try {

var param = new Pair<>(OrderFields.AssignedDroneId.getValue(), assignedDroneId);

Boolean flag = DBConnect.updateById(RequestType.ORDERS, this.getOrderId(), new ArrayList<>(Arrays.asList(param)));

if (!flag)

throw new SQLException("Unable to update drone in database.");

} catch (SQLException e) {e.printStackTrace();}

}

}

else throw new IllegalArgumentException("Incorrect drone ID.");

}

public void setWithReturn(String withReturn) throws IllegalArgumentException {

if (withReturn.equals("0") || withReturn.equals("1"))

this.withReturn = withReturn.equals("1");

else

throw new IllegalArgumentException("Wrong withReturn value.");

}

public void setIsDeleted(String isDeleted) throws IllegalArgumentException {

if (!Arrays.asList("0","1").contains(isDeleted))

throw new IllegalArgumentException("Incorrect isDeleted status.");

this.isDeleted = isDeleted;

}

public String[] sqlInsertValues(){

return Arrays.asList(

getSenderId(), getRecipientId(), getOrderDateTime().toString(),

getOrderStatus().getValue(), getAssignedDroneId(), getWithReturn() ? "1" : "0"

).toArray(new String[0]);

}

}

package Services;

import Enums.Fields.\*;

import Enums.Types.RequestType;

import Enums.Status.OrderStatus;

import OrderManagement.\*;

import UserManagement.\*;

import com.healthmarketscience.sqlbuilder.\*;

import org.javatuples.Pair;

import org.javatuples.Quartet;

import java.sql.\*;

import java.util.ArrayList;

public class DBConnect {

private static final String baseUrl = "jdbc:mysql://localhost:3306/DronePost";

private static final String dbUser = "root";

private static final String dbPassword = "";

private static ResultSet getQueryResults(String query) throws SQLException {

try {

Connection connection = getConnection();

Statement statement = connection.createStatement();

return statement.executeQuery(query);

} catch (ClassNotFoundException e) {

e.printStackTrace();

return null;

}

}

public static ArrayList<ArrayList<String>> getAllOrdersByClientId(String clientId) {

return getAllById(RequestType.ORDERS, clientId, null);

}

public static ArrayList<ArrayList<String>> getAll(RequestType type) {

return getAllById(type, null, null);

}

/\*\*

\* DBConnect API that returns data on the object by its ID.

\* @param type defines the table of enquiry, optional values come from RequestType enum

\* @param id string value that is the object's ID in the relevant table in the DB

\* @return ArrayList<String> with data of the object

\*/

public static ArrayList<String> getById(RequestType type, String id) {

ArrayList<String> parsedResults = new ArrayList<>(0);

Quartet<String, Integer, String, String> params = getQueryParams(type);

SelectQuery selectQuery = new SelectQuery().addAllColumns()

.addCustomFromTable(params.getValue0())

.addCondition(Converter.toConditionObject(params.getValue2() + id).setDisableParens(true));

if (!type.equals(RequestType.NOTIFICATIONS))

selectQuery.addCondition(Converter.toConditionObject(params.getValue3() + " = 0").setDisableParens(true));

try (ResultSet resultSet = getQueryResults(selectQuery.toString())) {

if (resultSet == null)

throw new NullPointerException("No results received from the DB.");

int index = 1;

resultSet.next();

while (index <= params.getValue1())

parsedResults.add(resultSet.getString(index++));

} catch (SQLException | NullPointerException e) {

e.printStackTrace();

}

return parsedResults;

}

public static Boolean updateById(RequestType type, String id, ArrayList<Pair<String, String>> parameters) {

Quartet<String, Integer, String, String> params = getQueryParams(type);

int updated = -1;

UpdateQuery updateQuery = new UpdateQuery(params.getValue0());

try {

if (parameters.isEmpty())

throw new IllegalArgumentException("Property and/or value parameters are invalid.");

parameters.forEach( item -> updateQuery.addCustomSetClause(item.getValue0(), item.getValue1()));

updateQuery.addCondition(Converter.toConditionObject(params.getValue2() + id).setDisableParens(true));

if (!type.equals(RequestType.NOTIFICATIONS) && !type.equals(RequestType.MESSAGES))

updateQuery.addCondition(Converter.toConditionObject(params.getValue3() + " = 0").setDisableParens(true));

} catch (IllegalArgumentException e) {

e.printStackTrace();

return false;

}

try (Connection connection = getConnection()) {

System.out.println(updateQuery);

updated = connection.prepareStatement(updateQuery.toString()).executeUpdate();

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return updated == 1;

}

public static Boolean deleteById(RequestType type, String id) throws ClassNotFoundException {

Quartet<String, Integer, String, String> params = getQueryParams(type);

UpdateQuery updateQuery = (new UpdateQuery(params.getValue0())).addCustomSetClause(params.getValue3(), 1)

.addCondition(Converter.toConditionObject(params.getValue2() + id).setDisableParens(true));

int updated = -1;

try (Connection connection = getConnection()) {

updated = connection.prepareStatement(updateQuery.toString()).executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

return updated == 1;

}

public static String insertClient(Client client) {

InsertQuery insertQuery = (new InsertQuery(RequestType.CLIENTS.getTableName()))

.addCustomColumns(ClientFields.sqlInsertColumns(), client.sqlInsertValues());

String insertedId = "-1";

try (Connection connection = getConnection()) {

if (getClientIdByTelephone(client.getTelephone()).isBlank()) {

connection.prepareStatement(insertQuery.toString()).executeUpdate();

insertedId = getClientIdByTelephone(client.getTelephone());

} else insertedId = "-2"; //code for "Provided phone number already exists"

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return insertedId;

}

public static String insertOrder(Order order) {

InsertQuery insertQuery = (new InsertQuery(RequestType.ORDERS.getTableName()))

.addCustomColumns(OrderFields.sqlInsertColumns(), order.sqlInsertValues());

String insertedId = "-1";

try (Connection connection = getConnection()) {

connection.prepareStatement(insertQuery.toString()).executeUpdate();

insertedId = getOrderIdBySenderIdOrderDateTime(order.getSenderId(), order.getOrderDateTime().toString());

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return insertedId;

}

public static String insertDrone(Drone drone) {

InsertQuery insertQuery = (new InsertQuery(RequestType.DRONES.getTableName()))

.addCustomColumns(DroneFields.sqlInsertColumns(), drone.sqlInsertValues());

String insertedId = "-1";

try (Connection connection = getConnection()) {

insertedId = String.valueOf(connection.prepareStatement(insertQuery.toString()).executeUpdate());

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return insertedId;

}

public static String insertNotification(Notifications notification) {

InsertQuery insertQuery = (new InsertQuery(RequestType.NOTIFICATIONS.getTableName()))

.addCustomColumns(NotificationFields.sqlInsertColumns(), notification.sqlInsertValues());

String insertedId = "-1";

try (Connection connection = getConnection()) {

connection.prepareStatement(insertQuery.toString()).executeUpdate();

insertedId = getNotificationByRecipientOrderId(notification.getRecipientId(), notification.getOrderId(), notification.getSentDateTime().toString());

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return insertedId;

}

public static String insertMessage(Messages message) {

InsertQuery insertQuery = (new InsertQuery(RequestType.MESSAGES.getTableName()))

.addCustomColumns(MessageFields.sqlInsertColumns(), message.sqlInsertValues());

String insertedId = "-1";

try (Connection connection = getConnection()) {

connection.prepareStatement(insertQuery.toString()).executeUpdate();

insertedId = getMessageBySenderRecipientOrderId(message.getSenderId(), message.getRecipientId(), message.getSentDateTime().toString());

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

}

return insertedId;

}

public static String getClientIdByTelephone(String telephone) throws ClassNotFoundException {

SelectQuery selectQuery = new SelectQuery().addCustomColumns(Converter.toCustomColumnSqlObject(ClientFields.ClientId.toString()))

.addCustomFromTable(RequestType.CLIENTS.getTableName())

.addCondition(Converter.toConditionObject("telephone = '" + telephone + "'").setDisableParens(true))

.addCondition(Converter.toConditionObject(ClientFields.IsDeleted.getValue() + " = 0").setDisableParens(true));

return getId(selectQuery);

}

public static ArrayList<ArrayList<String>> getPaidOrders() throws ClassNotFoundException {

return getAllById(RequestType.ORDERS, null, OrderStatus.Paid);

}

public static ArrayList<ArrayList<String>> getInTransitOrders() throws ClassNotFoundException {

return getAllById(RequestType.ORDERS, null, OrderStatus.InTransit);

}

private static String getOrderIdBySenderIdOrderDateTime(String senderId, String orderDateTime) throws ClassNotFoundException {

SelectQuery selectQuery = new SelectQuery().addCustomColumns(Converter.toCustomColumnSqlObject(OrderFields.OrderId.toString()))

.addCustomFromTable(RequestType.ORDERS.getTableName())

.addCondition(Converter.toConditionObject("senderId = " + senderId).setDisableParens(true))

.addCondition(Converter.toConditionObject("orderDateTime = '" + orderDateTime + "'").setDisableParens(true))

.addCondition(Converter.toConditionObject(OrderFields.IsDeleted.getValue() + " = 0").setDisableParens(true));

return getId(selectQuery);

}

private static String getNotificationByRecipientOrderId(String recipientId, String orderId, String sentDateTime) throws ClassNotFoundException {

SelectQuery selectQuery = new SelectQuery()

.addCustomColumns(Converter.toCustomColumnSqlObject(NotificationFields.NotificationId.toString()))

.addCustomFromTable(RequestType.NOTIFICATIONS.getTableName())

.addCondition(Converter.toConditionObject("recipientId = " + recipientId).setDisableParens(true))

.addCondition(Converter.toConditionObject("orderId = " + orderId).setDisableParens(true))

.addCondition(Converter.toConditionObject("sentDateTime = '" + sentDateTime + "'").setDisableParens(true));

return getId(selectQuery);

}

public static String getMessageBySenderRecipientOrderId(String senderId, String recipientId, String sentDateTime) throws ClassNotFoundException {

SelectQuery selectQuery = new SelectQuery()

.addCustomColumns(Converter.toCustomColumnSqlObject(MessageFields.MessageId.getValue()))

.addCustomFromTable(RequestType.MESSAGES.getTableName())

.addCondition(Converter.toConditionObject("senderId = " + senderId).setDisableParens(true))

.addCondition(Converter.toConditionObject("recipientId = " + recipientId).setDisableParens(true))

.addCondition(Converter.toConditionObject("sentDateTime = '" + sentDateTime + "'").setDisableParens(true));

return getId(selectQuery);

}

/\*\*

\*

\* @param type of RequestType list

\* @param clientId of the client to be queried. If the value is null, it will be disregarded

\* @param orderStatus of order to be queried. If value is null, it will be disregarded

\* @return the list of items, each one of which represents the list of details as per Request type

\*/

private static ArrayList<ArrayList<String>> getAllById(RequestType type, String clientId, OrderStatus orderStatus) {

ArrayList<ArrayList<String>> parsedResults = new ArrayList<>(0);

Quartet<String, Integer, String, String> params = getQueryParams(type);

String query;

SelectQuery selectQuery = new SelectQuery().addAllColumns()

.addCustomFromTable(params.getValue0());

//Adding condition to query for ALL orders

boolean getAllOrders = type.equals(RequestType.ORDERS) && clientId != null && orderStatus == null;

if (getAllOrders) {

selectQuery.addCondition(buildSenderRecipientCondition(clientId).setDisableParens(true));

}

//Adding condition to query for ANY UNPROCESSED orders

boolean getUnprocessedOrders = type.equals(RequestType.ORDERS) && clientId == null && orderStatus != null;

if (getUnprocessedOrders) {

selectQuery.addCondition(Converter.toConditionObject("orderStatus = '" + orderStatus.getValue() + "'").setDisableParens(true));

}

//adding isDeleted condition for queries except Notifications

if (!type.equals(RequestType.NOTIFICATIONS))

selectQuery.addCondition(Converter.toConditionObject(params.getValue3() + " = 0").setDisableParens(true));

//removing brackets for all queries except get ANY UNPROCESSED orders

query = orderStatus != null ? selectQuery.toString() : selectQuery.toString().replace("'", "");

try (ResultSet resultSet = getQueryResults(query)) {

if (resultSet == null)

throw new NullPointerException("No results received from the DB.");

while (resultSet.next()) {

int index = 1;

ArrayList<String> row = new ArrayList<>(0);

while (index <= params.getValue1())

row.add(resultSet.getString(index++));

parsedResults.add(row);

}

} catch (SQLException | NullPointerException e) {

e.printStackTrace();

}

return getAllOrders ? Parsers.parseSentReceivedOrders(parsedResults, clientId) : parsedResults;

}

private static Connection getConnection() throws ClassNotFoundException {

int i = 1;

Connection connection = null;

while (connection == null) {

try {

Class.forName("com.mysql.jdbc.Driver");

connection = DriverManager.getConnection(baseUrl, dbUser, dbPassword);

} catch (SQLException e) {

e.printStackTrace();

}

if (++i == 4) break;

}

return connection;

}

private static String getId(SelectQuery selectQuery) {

String parsedResults = "";

try (ResultSet resultSet = getQueryResults(selectQuery.toString())) {

if (!resultSet.next())

throw new NullPointerException("No results received from the DB.");

else parsedResults = resultSet.getString(1);

} catch (SQLException e) {

e.printStackTrace();

} catch (NullPointerException e) {

System.out.println(e.getLocalizedMessage());

return "";

}

return parsedResults;

}

private static Quartet<String, Integer, String, String> getQueryParams(RequestType type) {

String table = type.getTableName();

int colCount = 0;

String condition = "";

String deleteColumn = "";

switch (type) {

case CLIENTS -> {

colCount = ClientFields.values().length;

condition = "clientId = ";

deleteColumn = ClientFields.IsDeleted.getValue();

}

case ORDERS -> {

colCount = OrderFields.values().length;

condition = "orderId = ";

deleteColumn = OrderFields.IsDeleted.getValue();

}

case DRONES -> {

colCount = DroneFields.values().length;

condition = "droneId = ";

deleteColumn = DroneFields.IsDeleted.getValue();

}

case NOTIFICATIONS -> {

colCount = NotificationFields.values().length;

condition = "notificationId = ";

}

case MESSAGES -> {

colCount = MessageFields.values().length;

condition = "messageId = ";

}

}

return new Quartet<>(table, colCount, condition, deleteColumn);

}

private static ComboCondition buildSenderRecipientCondition(String clientId) {

return ComboCondition.or(

BinaryCondition.equalTo(OrderFields.SenderId.getValue(), clientId).setDisableParens(true),

BinaryCondition.equalTo(OrderFields.RecipientId.getValue(), clientId).setDisableParens(true));

}

}

package Services;

import Enums.Status.DroneStatus;

import Enums.Status.OrderStatus;

import OrderManagement.\*;

import java.util.\*;

public class DeliveryManager extends Thread {

private final Location base = new Location(0, 0);

private final Queue<Order> orders;

private Boolean active = false;

private Order currentOrder;

private Drone deliveryDrone;

private ArrayList<Drone> drones;

public DeliveryManager(ArrayList<Drone> drones) {

restoreInTransitOrders();

ArrayList<Order> paidOrders = new ArrayList<>();

try {

DBConnect.getPaidOrders().forEach(orderDetails -> paidOrders.add(new Order(orderDetails)) );

}

catch (ClassNotFoundException e) { e.printStackTrace(); }

this.orders = new LinkedList<>();

orders.addAll(paidOrders);

if (drones.isEmpty())

throw new NullPointerException("Drones' list is empty.");

else this.drones = drones;

startDelivery();

}

public void addOrder(Order order) {

this.orders.add(order);

startDelivery();

}

private Drone getSuitableDrone(Order order) {

Location senderLocation = order.getSender().getLocation();

Location recipientLocation = order.getRecipient().getLocation();

int distance = senderLocation.distanceTo(recipientLocation) + recipientLocation.distanceTo(base);

order.setDeliveryTime(distance);

for (Drone drone : drones) {

if (drone.getDroneStatus().equals(DroneStatus.Available.getValue())) {

int totalDistanceInMilliseconds = distance + drone.getDroneLocation().distanceTo(senderLocation);

if (totalDistanceInMilliseconds < drone.getBatteryLifeInMilliseconds()) {

drone.setDroneStatus(DroneStatus.InTransit, true);

return drone;

}

}

}

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

return getSuitableDrone(order);

}

private Drone assignDroneToTask(Order order) {

Drone drone = getSuitableDrone(order);

if (!drone.getDroneId().isBlank()) {

order.setDrone(drone);

orders.remove();

return drone;

}

return null;

}

public void startDelivery() {

active = true;

try {

new Thread( () -> {

while (active) {

if (!orders.isEmpty())

new Thread( () -> {

deliveryDrone = null;

currentOrder = orders.peek();

while (deliveryDrone == null){

deliveryDrone = assignDroneToTask(currentOrder);

}

deliveryDrone.deliverOrder(currentOrder);

}).run();

else active = false;

}

}).run();

} catch (NullPointerException e) {

e.printStackTrace();

}

}

private void restoreInTransitOrders(){

ArrayList<Order> inTransitOrders = new ArrayList<>();

try {

DBConnect.getInTransitOrders().forEach(orderDetails -> inTransitOrders.add(new Order(orderDetails)) );

inTransitOrders.forEach( order -> order.setOrderStatus(OrderStatus.Paid, true));

}

catch (ClassNotFoundException e) { e.printStackTrace(); }

}

}

package Services;

import java.io.\*;

public class LogManager {

private static final String folderAddress = "data\\";

private static final String fileName = "serverLog.txt";

public LogManager() {}

public static void printLog(String log) {

try {

File outputFile = createOutputFile(folderAddress + fileName);

if (outputFile == null)

throw new NullPointerException("Failed to create the output file.");

FileWriter myWriter = new FileWriter(outputFile, true);

PrintWriter out = new PrintWriter(new BufferedWriter(myWriter));

out.println(log + "\n");

out.close();

} catch (IOException e){

System.out.println("Error");

System.out.println(e.getMessage());

} catch (NullPointerException npe) { npe.printStackTrace(); }

}

/\*\*

\* Checks if the requested output folder exists and creates it if absent

\*/

private static void createOutputFolder(){

try {

File folder = new File(folderAddress);

if (!folder.isDirectory())

if (folder.mkdir())

System.out.println("Folder created: " + folder.getName());

else throw new IOException();

}

catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

/\*\*

\* Checks if the requested output file exists and creates it if absent

\* @param outputFile String containing the full name of the destination file

\* @return The requested output file

\*/

private static File createOutputFile(String outputFile){

try {

createOutputFolder();

File file = new File(outputFile);

if (!file.exists())

if (file.createNewFile())

System.out.println("File created: " + file.getName());

else throw new IOException();

return file;

}

catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

return null;

}

}

package Services;

import Enums.CommunicationCode;

import org.javatuples.Pair;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

import java.util.Arrays;

public class Parsers {

public static LocalDateTime dateTimeConverter(String dateTime) {

if (dateTime.contains("."))

dateTime = dateTime.split("\\.")[0];

if (dateTime.contains(" "))

dateTime = dateTime.replace(" ","T");

return LocalDateTime.parse(dateTime, DateTimeFormatter.ISO\_LOCAL\_DATE\_TIME);

}

public static String dateTimeConverter(LocalDateTime dateTime, Boolean forNotification) {

if (!forNotification)

return dateTime.truncatedTo(ChronoUnit.SECONDS).toString().replace("T", " ");

return dateTime.truncatedTo(ChronoUnit.MINUTES).toString().replace("T", " at ");

}

public static ArrayList<ArrayList<String>> parseSentReceivedOrders (ArrayList<ArrayList<String>> orders, String clientId){

orders.forEach(order ->{

if (order.get(1).equals(clientId))

order.add("Sender");

if (order.get(2).equals(clientId))

order.add("Recipient");

});

return orders;

}

public static Pair<CommunicationCode, ArrayList<String>> parseMessageFromClient (String message) {

CommunicationCode code = null;

ArrayList<String> messageContent = new ArrayList<>();

try {

if (!message.contains(";")) {

throw new IllegalArgumentException("Received invalid message from the client.");

}

String[] messageSplit = message.split(";");

code = parseCode(messageSplit[0]);

messageContent = parseMessage(messageSplit);

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (NullPointerException npe) {

npe.printStackTrace();

ServerManager.sendToClient(Parsers.parseMessageToClient(CommunicationCode.ServerProcessingError, null));

}

return new Pair<>(code, messageContent);

}

public static String parseMessageToClient (CommunicationCode code, ArrayList<String> data){

StringBuilder response = new StringBuilder().append(code.getCodeId()).append(";");

if (data != null)

response.append(String.join(";", data));

return response.toString();

}

private static CommunicationCode parseCode(String code){

CommunicationCode cc;

if (!CommunicationCode.codes().contains(code))

throw new IllegalArgumentException("Invalid message code");

else cc = CommunicationCode.getCode(code);

return cc;

}

private static ArrayList<String> parseMessage (String[] messageSplit) throws IllegalArgumentException {

ArrayList<String> messageContent = new ArrayList<>();

for (int i = 1; i < messageSplit.length; ++i) {

if (messageSplit[i].isBlank())

throw new IllegalArgumentException("Invalid message");

else messageContent.add(messageSplit[i]);

}

return messageContent;

}

}

package Services;

import Enums.CommunicationCode;

import Enums.Types.RequestType;

import OrderManagement.Drone;

import OrderManagement.Order;

import UserManagement.Client;

import UserManagement.Messages;

import org.javatuples.Pair;

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Arrays;

public class ServerManager {

public static ServerSocket serverSocket = null;

private static final int port = 9000;

public static Socket clientSocket;

private static BufferedReader fromClient;

private static PrintStream toClient;

private static DeliveryManager deliveryManager;

private static ArrayList<Drone> drones = new ArrayList<>();

private static String phoneNumber = "";

public static void startServer() {

try {

drones = new Drone().getAllDrones();

deliveryManager = new DeliveryManager(drones);

serverSocket = new ServerSocket(port);//or 127.0.0.1

System.out.println("Connected to server..." + serverSocket.getInetAddress() + " " + serverSocket.getLocalPort());

start();

} catch (IOException e) {

e.printStackTrace();

}

}

protected static void start() throws IOException {

while (true) {

clientSocket = serverSocket.accept();

new Thread(() -> {

while (!clientSocket.isClosed()) {

try {

fromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

toClient = new PrintStream(clientSocket.getOutputStream(), true);

Pair<CommunicationCode, ArrayList<String>> parsedMessage = Parsers.parseMessageFromClient(fromClient.readLine());

System.out.println(parsedMessage);

switch (parsedMessage.getValue0()) {

case RegisterClient -> processRegisterRequest(parsedMessage.getValue1());

case GetClientDetails -> processGetClientDetailsRequest(parsedMessage.getValue1());

case NewOrder -> processNewOrderRequest(parsedMessage.getValue1());

case NewMessage -> processNewMessageRequest(parsedMessage.getValue1());

case ValidateTelephone -> processValidateTelephoneRequest(parsedMessage.getValue1());

case HistoryRequest -> processHistoryRequest(parsedMessage.getValue1());

}

} catch (Exception e) {

e.printStackTrace();

break;

}

}

}).run();

}

}

private static void processRegisterRequest(ArrayList<String> requestData) {

requestData.add(0, "-1");

Client client = new Client(requestData, true);

String dbResponse = DBConnect.insertClient(client);

switch (dbResponse) {

case "-2" -> sendToClient(Parsers.parseMessageToClient(CommunicationCode.ClientAlreadyExists, null));

case "-1" -> sendToClient(Parsers.parseMessageToClient(CommunicationCode.ClientSavingError, null));

default -> {

client.setClientId(dbResponse);

sendToClient(Parsers.parseMessageToClient(CommunicationCode.ClientRegistered, null));

phoneNumber = requestData.get(10);

}

}

}

private static void processGetClientDetailsRequest(ArrayList<String> requestData) {

try {

String id = DBConnect.getClientIdByTelephone(requestData.get(0));

ArrayList<String> allDetails = null;

if (id != null && !id.isBlank())

allDetails = DBConnect.getById(RequestType.CLIENTS, id);

if (allDetails.size() > 12)

ServerManager.sendToClient(Parsers.parseMessageToClient(CommunicationCode.GetClientDetails, new ArrayList<>(allDetails.subList(0, 12))));

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

private static void processNewOrderRequest(ArrayList<String> requestData) {

new Thread( () -> {

try {

String senderId = DBConnect.getClientIdByTelephone(requestData.get(0));

String recipientId = DBConnect.getClientIdByTelephone(requestData.get(1));

Client sender = new Client(DBConnect.getById(RequestType.CLIENTS, senderId), false);

Client recipient = new Client(DBConnect.getById(RequestType.CLIENTS, recipientId), false);

Order newOrder = sender.createOrder(recipient);

deliveryManager.addOrder(newOrder);

} catch (ClassNotFoundException cnf) {

cnf.printStackTrace();

}

}).start();

}

private static void processNewMessageRequest(ArrayList<String> requestData) {

new Thread( () -> {

Messages message = new Messages(requestData);

message.setMessageId(DBConnect.insertMessage(message));

message.sendMessage();

message.awaitClientResponse();

message.updateMessageDelivered(true);

}).start();

}

private static void processValidateTelephoneRequest(ArrayList<String> requestData) {

new Thread ( () -> {

try {

String response = DBConnect.getClientIdByTelephone(requestData.get(0));

if (!response.isBlank())

sendToClient(Parsers.parseMessageToClient(CommunicationCode.ValidateTelephone, new ArrayList<>(Arrays.asList("valid"))));

else

sendToClient(Parsers.parseMessageToClient(CommunicationCode.ValidateTelephone, new ArrayList<>(Arrays.asList("invalid"))));

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}).start();

}

public static void processHistoryRequest(ArrayList<String> phone) {

new Thread(() -> {

try {

String clientPhone = phone.get(0);

String clientID = DBConnect.getClientIdByTelephone(clientPhone);

ArrayList<ArrayList<String>> orders = DBConnect.getAllOrdersByClientId(clientID);

StringBuilder order = new StringBuilder();

for (ArrayList<String> o : orders) {

order.append("order ID: ").append(o.get(0)).append(", ");

order.append("sender ID: ").append(o.get(1)).append(", ");

order.append("recipient ID: ").append(o.get(2)).append(", ");

order.append("time of order: ").append(o.get(3)).append(", ");

if (o.get(4).equals("delivered")) {

order.append("order delivered");

} else {

order.append("order not yet delivered");

}

order.append(";");

}

sendToClient(Parsers.parseMessageToClient(CommunicationCode.HistoryReport,

new ArrayList<>(Arrays.asList(order.toString()))));

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}).start();

}

public static void sendToClient(String message) {

try {

System.out.println(message);

if (toClient != null)

toClient.println(message);

} catch (NullPointerException e) {

e.printStackTrace();

}

}

public static String receiveFromClient() {

try {

return fromClient.readLine();

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

private static void stop() {

try {

fromClient.close();

toClient.close();

clientSocket.close();

serverSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

public static String getPhoneNumber() {

return phoneNumber;

}

}

package UserManagement;

import Enums.Fields.ClientFields;

import Enums.Types.RequestType;

import Services.DBConnect;

import Enums.Status.OrderStatus;

import OrderManagement.\*;

import java.sql.SQLException;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.Arrays;

import OrderManagement.Location;

import org.javatuples.Pair;

public class Client extends Person {

private String clientId;

private int posX;

private int posY;

private String subscriptionCode;

private int numberOfDeliveriesLeft;

private String isDeleted = "0";

private final ArrayList<String> subscriberCodes = new ArrayList<>(Arrays.asList("1", "2"));

private Location location;

/\*\*

\* Create the instance of Users.Client, that extends Users.Person.

\* @param clientDetails - ordered list of data: clientId, personalId, firstName, lastName, dateOfBirth,

\* street, building, city, zipCode, email, telephone, subscriptionCode.

\* In case the client is being restored from the DB, additional data shall appear:

\* posX, posY, numberOfDeliveriesLeft, isDeleted

\* @param fromUI - If true the constructor will set posX & posY randomly, numberOfDeliveriesLeft (based on the subscriptionCode),

\* isDeleted (default value >> 0)

\*/

public Client(ArrayList<String> clientDetails, Boolean fromUI){

super(new ArrayList<>(clientDetails.subList(1,11)));

if (fromUI) {

this.clientId = clientDetails.get(0);

updateSubscription(clientDetails.get(11));

this.location = new Location();

this.posX = this.location.getPosX();

this.posY = this.location.getPosY();

} else {

this.clientId = clientDetails.get(0);

this.subscriptionCode = clientDetails.get(11);

this.posX = Integer.parseInt(clientDetails.get(12));

this.posY = Integer.parseInt(clientDetails.get(13));

this.location = new Location(this.posX, this.posY);

setNumberOfDeliveriesLeft(Integer.parseInt(clientDetails.get(14)), false);

this.isDeleted = clientDetails.get(15);

}

}

/\*\*

\* Create order and assign a drone to it

\* @param recipient of type Client

\* @return object of type Order

\*/

private Order createOrder(Client recipient, Boolean withReturn){

try{

Order order = new Order(this, recipient, LocalDateTime.now().withNano(0), OrderStatus.Paid, withReturn);

String orderId = DBConnect.insertOrder(order);

if (Integer.parseInt(orderId) < 1)

throw new SQLException("Unable to save the order to database.");

order.setOrderId(orderId);

this.setNumberOfDeliveriesLeft(this.getNumberOfDeliveriesLeft()-1, true);

return order;

} catch (SQLException e){

e.printStackTrace();

}

return null;

}

//This method will always create orders without return

public Order createOrder(Client recipient) {

return createOrder(recipient, false);

}

public String getClientId() {

return clientId;

}

public String getSubscriptionCode() {

return subscriptionCode;

}

public int getPosX() { return posX; }

public int getPosY() { return posY; }

public int getNumberOfDeliveriesLeft() {

return numberOfDeliveriesLeft;

}

public Location getLocation(){ return this.location;}

public void setClientId(String clientId) {

this.clientId = clientId;

}

public void setNumberOfDeliveriesLeft(int numberOfDeliveriesLeft, Boolean updateDB) throws IllegalArgumentException {

int max = 0;

if (!subscriberCodes.contains(subscriptionCode))

throw new IllegalArgumentException("Invalid subscription code");

switch (this.subscriptionCode){

case "1" -> max = 50;

case "2" -> max = 150;

}

if (numberOfDeliveriesLeft < 0 || numberOfDeliveriesLeft > max)

throw new IllegalArgumentException("Invalid number of deliveries left.");

this.numberOfDeliveriesLeft = numberOfDeliveriesLeft;

if (updateDB) {

var param = new Pair<>(ClientFields.NumberOfDeliveriesLeft.getValue(), String.valueOf(numberOfDeliveriesLeft));

DBConnect.updateById(RequestType.CLIENTS, this.getClientId(), new ArrayList<>(Arrays.asList(param)));

}

}

public void updateSubscription(String subscriptionCode) throws IllegalArgumentException {

this.numberOfDeliveriesLeft = 0;

if (!subscriberCodes.contains(subscriptionCode))

throw new IllegalArgumentException("Invalid subscription code");

this.subscriptionCode = subscriptionCode;

switch (this.subscriptionCode){

case "1" -> numberOfDeliveriesLeft += 50;

case "2" -> numberOfDeliveriesLeft += 150;

}

}

public String[] sqlInsertValues(){

return Arrays.asList(

getPersonalId(), getFirstName(), getLastName(), getDateOfBirth().toString(),

getStreet(), getBuilding(), getCity(), getZipCode(), getEmail(), getTelephone(),

String.valueOf(getSubscriptionCode()), String.valueOf(getPosX()),

String.valueOf(getPosY()), String.valueOf(getNumberOfDeliveriesLeft())

).toArray(new String[0]);

}

}

package UserManagement;

import Enums.CommunicationCode;

import Enums.Fields.MessageFields;

import Enums.Types.RequestType;

import Services.DBConnect;

import Services.Parsers;

import Services.ServerManager;

import org.javatuples.Pair;

import java.time.LocalDateTime;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

import java.util.Arrays;

public class Messages {

private String messageId;

private String senderId;

private String recipientId;

private LocalDateTime sentDateTime;

private String message;

private Boolean delivered;

private ArrayList<String> messageData;

private ArrayList<String> clientResponse;

public Messages(ArrayList<String> messageData) {

try {

this.messageData = messageData;

setSender(messageData.get(0));

setRecipient(messageData.get(1));

setMessage(messageData.get(2));

setSentDateTime(LocalDateTime.now());

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public String getMessageId() {

return this.messageId;

}

public String getSenderId() {

return this.senderId;

}

public String getRecipientId() {

return this.recipientId;

}

public LocalDateTime getSentDateTime() {

return this.sentDateTime;

}

public String getMessage() {

return this.message;

}

public String getDelivered() { return this.delivered ? "1" : "0"; }

public void setMessageId(String messageId) {

this.messageId = messageId;

}

public void setSender(String senderTelephone) throws ClassNotFoundException {

this.senderId = DBConnect.getClientIdByTelephone(senderTelephone);

}

public void setRecipient(String recipientTelephone) throws ClassNotFoundException {

this.recipientId = DBConnect.getClientIdByTelephone(recipientTelephone);

}

public void setSentDateTime(LocalDateTime sentDateTime) throws IllegalArgumentException {

if (sentDateTime.isAfter(LocalDateTime.now()))

throw new IllegalArgumentException("Incorrect order Date/Time provided.");

this.sentDateTime = sentDateTime.truncatedTo(ChronoUnit.SECONDS);

}

public void setMessage(String message) throws IllegalArgumentException {

if (message.length() > 500)

throw new IllegalArgumentException("Message length is over 500 symbols.");

this.message = message;

}

public void updateMessageDelivered(Boolean updateDB) {

this.delivered = true;

ServerManager.sendToClient(Parsers.parseMessageToClient(CommunicationCode.MessageDelivered, this.clientResponse));

if (updateDB) {

var param = new Pair<>(MessageFields.Delivered.getValue(), getDelivered() );

DBConnect.updateById(RequestType.MESSAGES, this.getMessageId(), new ArrayList<>(Arrays.asList(param)));

}

}

/\*\*

\* Sends the message to recipient

\*/

public void sendMessage() {

String messageReceiver = Parsers.parseMessageToClient(CommunicationCode.NewMessage, this.messageData);

ServerManager.sendToClient(messageReceiver); //send notification via SMS API

}

public ArrayList<String> awaitClientResponse() {

String messageFromClient = ServerManager.receiveFromClient();

Pair<CommunicationCode, ArrayList<String>> parsedMessage;

if (messageFromClient != null) {

parsedMessage = Parsers.parseMessageFromClient(messageFromClient);

System.out.println(parsedMessage);

if (parsedMessage.getValue0().equals(CommunicationCode.MessageReceived)

&& getSenderId().equals(parsedMessage.getValue1().get(1))

&& getRecipientId().equals(parsedMessage.getValue1().get(0))

&& getMessage().equals(parsedMessage.getValue1().get(2)))

this.clientResponse = parsedMessage.getValue1();

}

return clientResponse;

}

public String[] sqlInsertValues() {

return Arrays.asList(

getSenderId(), getRecipientId(), getSentDateTime().toString(), getMessage()

).toArray(new String[0]);

}

}

package UserManagement;

import java.time.LocalDate;

import java.util.ArrayList;

public class Person {

private String personalId;

private String firstName;

private String lastName;

private LocalDate dateOfBirth;

private String street;

private String building;

private String city;

private String zipCode;

private String email;

private String telephone;

/\*\*

\* Create the instance of Users.Person. The person shall have mandatory fields

\* @param personDetails - ordered list of data: personalId, firstName, lastName, dateOfBirth,

\* street, building, city, zipCode, email, telephone. All the field values are mandatory,

\* except: firstName and building

\*/

public Person (ArrayList<String> personDetails){

try {

setPersonalId(personDetails.get(0));

setFirstName(personDetails.get(1));

setLastName(personDetails.get(2));

setDateOfBirth(personDetails.get(3));

setStreet(personDetails.get(4));

setBuilding(personDetails.get(5));

setCity(personDetails.get(6));

setZipCode(personDetails.get(7));

setEmail(personDetails.get(8));

setTelephone(personDetails.get(9));

} catch (IllegalArgumentException e){

e.printStackTrace();

}

}

public String getPersonalId() {

return personalId;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getFullName() { return getFirstName() + " " + getLastName(); }

public String getBuilding() {

if (building != null)

return building;

else return "";

}

public String getCity() {

return city;

}

public String getStreet() {

return street;

}

public String getZipCode() {

return zipCode;

}

public String getFullAddress(){

return street + " St, " + (building != null ? building + ", " : "") + city + ", " + zipCode;

}

public LocalDate getDateOfBirth() {

return dateOfBirth;

}

public String getEmail() {return email; }

public String getTelephone() {

return telephone;

}

public void setPersonalId (String personalId) throws IllegalArgumentException{

boolean isDigit = false;

for(int i = 0; i< personalId.length(); ++i) {

isDigit = Character.isDigit(personalId.charAt(i));

if (!isDigit) break;

}

if (personalId.isEmpty() || personalId.length() > 10 || !isDigit)

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.personalId = personalId;

}

public void setFirstName(String firstName) throws IllegalArgumentException{

if (firstName.length() > 30)

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.firstName = firstName;

}

public void setLastName(String lastName) throws IllegalArgumentException{

if (lastName.isEmpty() || lastName.length() > 30)

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.lastName = lastName;

}

public void setDateOfBirth (String dateOfBirth) throws IllegalArgumentException {

LocalDate dateOfBirthParsed = LocalDate.parse(dateOfBirth);

if (dateOfBirthParsed.isAfter(LocalDate.now()))

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.dateOfBirth = dateOfBirthParsed;

}

public void setBuilding(String building) throws IllegalArgumentException {

if (building.length() > 5)

throw new IllegalArgumentException("Fill in all the mandatory fields");

this.building = building;

}

public void setCity(String city) throws IllegalArgumentException {

if (city.isEmpty() || city.length() > 30)

throw new IllegalArgumentException("Fill in all the mandatory fields");

this.city = city;

}

public void setStreet(String street) throws IllegalArgumentException {

if (street.isEmpty() || street.length() > 40)

throw new IllegalArgumentException("Fill in all the mandatory fields");

this.street = street;

}

public void setZipCode(String zipCode) throws IllegalArgumentException {

if (zipCode.isEmpty() || zipCode.length() > 10)

throw new IllegalArgumentException("Fill in all the mandatory fields");

this.zipCode = zipCode;

}

public void setEmail (String email) throws IllegalArgumentException {

if (email.isEmpty() || email.length() > 45)

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.email = email;

}

public void setTelephone(String telephone) throws IllegalArgumentException {

if (telephone.length() > 15 || !telephone.startsWith("+"))

throw new IllegalArgumentException("Fill in all the mandatory fields correctly");

this.telephone = telephone;

}

}

import Services.ServerManager;

public class Main {

public static void main (String [] args) {

ServerManager.startServer();

}

}

DronePostUI (client) classes:

package Enums;

import java.util.\*;

public enum CommunicationCode {

RegisterClient("1"),

ClientRegistered("2"),

ClientAlreadyExists("3"),

ClientSavingError("4"),

ValidateTelephone("5"),//Server to ClientApp

GetClientDetails("6"),

NewOrder("7"),//ClientApp to Server

NewMessage("8"),//ClientApp to Server

ReceiverNotConnectedToServer("9"),//Server to ClientApp

MessageDelivered("10"),//Server to ClientApp(Receiver ClientApp)

OrderDelivered("11"),//Server to ClientApp (Sender)

OrderSent("12"),

ServerProcessingError("13"),

MessageReceived("14"),

HistoryRequest("15"),

HistoryReport("16");

private String codeId;

CommunicationCode (String codeId) {

this.codeId = codeId;

}

private static final Map<String,CommunicationCode> lookup = new HashMap<>();

static {

for(CommunicationCode cc : EnumSet.allOf(CommunicationCode.class))

lookup.put(cc.getCodeId(), cc);

}

public String getCodeId(){

return this.codeId;

}

public static ArrayList<String> codes(){

var codes = new ArrayList<String>();

for (CommunicationCode cc : CommunicationCode.values()) {

codes.add(cc.getCodeId());

}

return codes;

}

public static CommunicationCode getCode(String codeId){

return lookup.get(codeId);

}

}

package Enums;

import java.util.EnumSet;

import java.util.HashMap;

import java.util.Map;

public enum SubscriptionCode {

NO\_SUBSCRIPTION("0", "-"),

SMALL\_PACKAGE("1", "50 shipments for $99"),

BIG\_PACKAGE("2", "150 shipments for $179");

private String code;

private String description;

private static final Map<String,SubscriptionCode> codeLookup = new HashMap<>();

private static final Map<String,SubscriptionCode> descriptionLookup = new HashMap<>();

static {

for(SubscriptionCode sc : EnumSet.allOf(SubscriptionCode.class))

codeLookup.put(sc.getCode(), sc);

}

static {

for(SubscriptionCode sc : EnumSet.allOf(SubscriptionCode.class))

descriptionLookup.put(sc.getDescription(), sc);

}

SubscriptionCode(String code, String description){

this.code = code;

this.description = description;

}

public final String getCode(){

return this.code;

}

public static String getCode(String description) { return descriptionLookup.get(description).getCode(); }

public final String getDescription(){

return this.description;

}

public static String getDescription(String code){

return codeLookup.get(code).getDescription();

}

}

package Services;

import DialogsUI.LoginForm;

import DialogsUI.MainForm;

import DialogsUI.ReceivedMessageForm;

import DialogsUI.RegistrationForm;

import Enums.CommunicationCode;

import org.javatuples.Pair;

import javax.swing.\*;

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Arrays;

public class ClientManager {

private static final String appName = "Drone Post";

private static Socket clientSocket;

private static String host = "localhost";

private static int port = 9000;

private static BufferedReader fromServer;

private static PrintStream toServer;

private static String currentUserId;

public static void startClient() {

try {

clientSocket = new Socket(host, port);

System.out.println("Connected to server..." + clientSocket.getInetAddress() + " " + clientSocket.getPort());

fromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

toServer = new PrintStream(clientSocket.getOutputStream(), true);

openLoginForm();

} catch (IOException e) {

e.printStackTrace();

}

}

public static String receiveFromServer() {

try {

return fromServer.readLine();

} catch (IOException ie) {

ie.printStackTrace();

}

return null;

}

public static void sendToServer(String message) {

toServer.println(message);

}

public static void setCurrentUserId(String loggedInUserId) {

if (loggedInUserId != null && !loggedInUserId.isBlank())

currentUserId = loggedInUserId;

}

public static void openLoginForm() {

new LoginForm(appName);

}

public static void openRegistrationForm(JFrame form) {

form.dispose();

new RegistrationForm(appName);

}

public static void openOrderForm(JFrame form, ArrayList<String> currentUserData) {

form.dispose();

new MainForm(appName, currentUserData);

}

public static Boolean validateTelephone(String telephone) {

if (telephone.length() != 13) return false;

String message = Parsers.parseMessageToServer(CommunicationCode.ValidateTelephone, new ArrayList<>(Arrays.asList(telephone)));

ClientManager.sendToServer(message);

var response = Parsers.parseMessageFromServer(ClientManager.receiveFromServer());

if (response.getValue0() == CommunicationCode.ValidateTelephone)

switch (response.getValue1().get(0)) {

case "valid":

return true;

case "invalid":

return false;

default:

System.out.println("Received unknown response validating the telephone.");

}

return false;

}

public static void stop() {

try {

fromServer.close();

toServer.close();

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

public static void processServerResponse() {

new Thread(() -> {

while (!clientSocket.isClosed()) {

Pair<CommunicationCode, ArrayList<String>> parsedMessage = Parsers.parseMessageFromServer(receiveFromServer());

switch (parsedMessage.getValue0()) {

//case ReceiverNotConnectedToServer -> new ReceivedMessageForm("server", "Client not connected to server");;

case NewMessage -> {

if (parsedMessage.getValue1().get(0).equals(currentUserId)) {

new Thread(() -> {

new ReceivedMessageForm(parsedMessage.getValue1().get(1), parsedMessage.getValue1().get(2));

ArrayList<String> data = new ArrayList<>(Arrays.asList(parsedMessage.getValue1().get(1), parsedMessage.getValue1().get(0), parsedMessage.getValue1().get(2)));

sendToServer(Parsers.parseMessageToServer(CommunicationCode.MessageReceived, data));

}).run();

}

}

case MessageDelivered -> {//receive message.

new ReceivedMessageForm(parsedMessage.getValue1().get(1), "Message delivered");

return;

}

case OrderSent, OrderDelivered -> {

if (parsedMessage.getValue1().get(0).equals(currentUserId))

new ReceivedMessageForm("Drone", parsedMessage.getValue1().get(1));

}

case GetClientDetails -> {

setCurrentUserId(parsedMessage.getValue1().remove(0));

new Services().saveClientDetailsToFile(parsedMessage.getValue1());

return;

}

case ServerProcessingError -> throw new NullPointerException("Server processing error due to NULL message.");

case HistoryReport -> {

if (parsedMessage.getValue1().isEmpty())

new ReceivedMessageForm("Server", "You have not sent/received any orders yet.");

else new ReceivedMessageForm("Server", parsedMessage.getValue1());

return;

}

}

}

}).start();

}

}

package Services;

import Enums.CommunicationCode;

import org.javatuples.Pair;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.util.ArrayList;

public class Parsers {

public static LocalDateTime dateTimeConverter(String dateTime) {

if (dateTime.contains("."))

dateTime = dateTime.split("\\.")[0];

if (dateTime.contains(" "))

dateTime = dateTime.replace(" ","T");

return LocalDateTime.parse(dateTime, DateTimeFormatter.ISO\_LOCAL\_DATE\_TIME);

}

public static String dateTimeConverter(LocalDateTime dateTime, Boolean forNotification) {

if (!forNotification)

return dateTime.truncatedTo(ChronoUnit.SECONDS).toString().replace("T", " ");

return dateTime.truncatedTo(ChronoUnit.MINUTES).toString().replace("T", " at ");

}

public static ArrayList<ArrayList<String>> parseSentReceivedOrders (ArrayList<ArrayList<String>> orders, String clientId){

orders.forEach(order ->{

if (order.get(1).equals(clientId))

order.add("Sender");

if (order.get(2).equals(clientId))

order.add("Recipient");

});

return orders;

}

public static Pair<CommunicationCode, ArrayList<String>> parseMessageFromServer (String message) {

CommunicationCode code = null;

ArrayList<String> messageContent = new ArrayList<>();

try {

if (message == null || !message.contains(";"))

throw new IllegalArgumentException("Received invalid message from the client.");

String[] messageSplit = message.split(";");

code = parseCode(messageSplit[0]);

messageContent = parseMessage(messageSplit);

} catch (IllegalArgumentException e) {

e.printStackTrace();

}

return new Pair<>(code, messageContent);

}

public static String parseMessageToServer (CommunicationCode code, ArrayList<String> data){

StringBuilder response = new StringBuilder().append(code.getCodeId() + ";");

if (data != null)

response.append(String.join(";", data));

return response.toString();

}

private static CommunicationCode parseCode(String code){

CommunicationCode cc;

if (!CommunicationCode.codes().contains(code))

throw new IllegalArgumentException("Invalid message code");

else cc = CommunicationCode.getCode(code);

return cc;

}

private static ArrayList<String> parseMessage (String[] messageSplit) throws IllegalArgumentException {

ArrayList<String> messageContent = new ArrayList<>();

for (int i = 1; i < messageSplit.length; ++i) {

if (messageSplit[i].isBlank())

throw new IllegalArgumentException("Invalid message");

else messageContent.add(messageSplit[i]);

}

return messageContent;

}

}

package Services;

import java.io.\*;

import java.util.ArrayList;

import java.util.Scanner;

public class Services {

private final String folderAddress = "data\\";

private final String fileName = "currentUser.txt";

public Services() {}

public ArrayList<String> getClientDetailsFromFile() {

File outputFile = createOutputFile(folderAddress + fileName);

ArrayList<String> clientData = new ArrayList<>();

try (Scanner myReader = new Scanner(outputFile)){

while (myReader.hasNextLine())

clientData.add(myReader.nextLine().trim());

} catch (FileNotFoundException e){

e.printStackTrace();

}

return clientData;

}

public void saveClientDetailsToFile(ArrayList<String> dataSource) {

try {

File outputFile = createOutputFile(folderAddress + fileName);

FileWriter myWriter = new FileWriter(outputFile, false);

PrintWriter out = new PrintWriter(new BufferedWriter(myWriter));

dataSource.forEach(out::println);

out.close();

} catch (IOException e){

System.out.println("Error");

System.out.println(e.getMessage());

}

}

/\*\*

\* Checks if the requested output folder exists and creates it if absent

\*/

private void createOutputFolder(){

try {

File folder = new File(folderAddress);

if (!folder.isDirectory())

if (folder.mkdir())

System.out.println("Folder created: " + folder.getName());

else throw new IOException();

}

catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

/\*\*

\* Checks if the requested output file exists and creates it if absent

\* @param outputFile String containing the full name of the destination file

\* @return The requested output file

\*/

private File createOutputFile(String outputFile){

try {

createOutputFolder();

File file = new File(outputFile);

if (!file.exists())

if (file.createNewFile())

System.out.println("File created: " + file.getName());

else throw new IOException();

return file;

}

catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

return null;

}

}

package DialogsUI;

import Enums.CommunicationCode;

import Services.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Arrays;

public class LoginForm extends JFrame implements ActionListener, WindowListener {

/\* log in form at the beginning of program running, asking user if he usd the program from current machine and if not to register. \*/

private final String mainLabel = "Welcome to Drone Post app";

private final String welcomeLabel = "Please, authenticate using phone number";

private final String errorMessage = "User not found";

private static final ArrayList<String> formButtons = new ArrayList<>(Arrays.asList("Register", "Authenticate"));

private final JTextField titleLabel = new JTextField(mainLabel);

private final JTextField welcomeMessage = new JTextField(welcomeLabel);

private final JTextField clientNotification = new JTextField(20);

private final JTextField telephone = new JTextField(15);

private final JPanel titlePanel = new JPanel(new GridLayout(0, 1));

private final JPanel mainPanel = new JPanel(new GridLayout(0, 1));

private final JPanel loginPanel = new JPanel(new GridLayout(0, 2));

private final JButton registerButton = new JButton(formButtons.get(0));

private final JButton authButton = new JButton(formButtons.get(1));

private final JPanel loginFormButtons = new JPanel(new FlowLayout());

public LoginForm(String appName) {

super(appName);

addWindowListener(this);

setLayout((new BorderLayout(20, 15)));

setFormLabel();

setWelcomeLabel();

setFormBody();

setButtonsPanel(loginFormButtons, new ArrayList<>(Arrays.asList(registerButton, authButton)));

add(titlePanel, BorderLayout.NORTH);

add(mainPanel, BorderLayout.CENTER);

add(loginFormButtons, BorderLayout.SOUTH);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

pack();

setVisible(true);

setResizable(false);

}

@Override

public void actionPerformed(ActionEvent e) {

String buttonClicked = e.getActionCommand();

switch (buttonClicked) {

case "Register" -> {

System.out.println("Register");

ClientManager.openRegistrationForm(this);

}

case "Authenticate" -> {

System.out.println("Authenticate");

ArrayList<String> currentUserData = authenticate();

if (currentUserData != null)

ClientManager.openOrderForm(this, currentUserData);

}

}

}

private void setFormLabel() {

titleLabel.setEditable(false);

titleLabel.setBackground(null);

titleLabel.setBorder(null);

titleLabel.setFont(new Font(null, Font.BOLD, 14));

titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

titlePanel.add(titleLabel, BorderLayout.NORTH);

}

private void setWelcomeLabel() {

welcomeMessage.setEditable(false);

welcomeMessage.setBackground(null);

welcomeMessage.setBorder(null);

welcomeMessage.setFont(new Font(null, Font.PLAIN, 11));

welcomeMessage.setHorizontalAlignment(SwingConstants.LEFT);

mainPanel.add(welcomeMessage, BorderLayout.NORTH);

}

private void setFormBody() {

setFields();

String columnLabel = "Phone Number";

loginPanel.add(new Label(columnLabel));

loginPanel.add(telephone);

mainPanel.add(loginPanel, BorderLayout.CENTER);

setClientNotification(Color.GREEN, "Placeholder", false);

mainPanel.add(clientNotification, BorderLayout.SOUTH);

}

private void setFields() {

telephone.setText("+972");

telephone.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

int length = telephone.getText().length();

if ((length == 4 && e.getKeyChar() == '0') || !Character.isDigit(e.getKeyChar()) || (length == 13))

e.setKeyChar(Character.MIN\_VALUE);

if (clientNotification.isVisible()) hideNotification();

}

@Override

public void keyPressed(KeyEvent e) {

if ((e.getKeyCode() == KeyEvent.VK\_DELETE || e.getKeyCode() == KeyEvent.VK\_BACK\_SPACE)

&& clientNotification.isVisible())

hideNotification();

}

@Override

public void keyReleased(KeyEvent e) {

}

});

}

private void setClientNotification(Color color, String message, Boolean show) {

clientNotification.setBackground(null);

clientNotification.setBorder(null);

clientNotification.setText(message);

clientNotification.setForeground(color);

clientNotification.setHorizontalAlignment(SwingConstants.CENTER);

if (show)

showNotification();

else hideNotification();

}

private void showNotification() {

clientNotification.setVisible(true);

}

private void hideNotification() {

clientNotification.setVisible(false);

}

private void setButtonsPanel(JPanel panel, ArrayList<JButton> buttons) {

buttons.forEach(button -> {

panel.add(button);

button.setPreferredSize(new Dimension(120, 25));

button.addActionListener(this);

});

}

private ArrayList<String> authenticate() {

String typedPhone = telephone.getText();

ArrayList<String> currentUserData;

if (!ClientManager.validateTelephone(typedPhone)) {

setClientNotification(Color.RED, errorMessage, true);

return null;

} else {

ClientManager.sendToServer(Parsers.parseMessageToServer(CommunicationCode.GetClientDetails, new ArrayList<>(Arrays.asList(typedPhone))));

ClientManager.processServerResponse();

currentUserData = new Services().getClientDetailsFromFile();

}

return currentUserData;

}

@Override

public void windowOpened(WindowEvent e) {

}

@Override

public void windowClosing(WindowEvent e) {

}

@Override

public void windowClosed(WindowEvent e) {

this.dispose();

System.out.println("Window closed");

}

@Override

public void windowIconified(WindowEvent e) {

System.out.println("Minimized");

}

@Override

public void windowDeiconified(WindowEvent e) {

}

@Override

public void windowActivated(WindowEvent e) {

}

@Override

public void windowDeactivated(WindowEvent e) {

}

}

package DialogsUI;

import Enums.CommunicationCode;

import Services.ClientManager;

import Services.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Arrays;

import javax.swing.\*;

public class MainForm extends JFrame implements WindowListener, ActionListener {

private final String actionChoiceLabel = "Choose your action";

private final ArrayList<String> telephoneLabelText = new ArrayList<>(Arrays.asList("Recipient", "Sender"));

private final ArrayList<String> senderOrReceiverText = new ArrayList<>(Arrays.asList("Send package", "Receive package", "Send message"));

private final ArrayList<String> orderButtonLabels = new ArrayList<>(Arrays.asList("Send", "Receive", "Order history", "Sign out"));

private final String orderSent = "Order sent";

private final String messageSent = "Message sent";

private final JTextArea textArea = new JTextArea(3,20);

private final JTextField titleLabel = new JTextField();

private final JTextField telephone = new JTextField(20);

private final JTextField notification = new JTextField(20);

private final JLabel telephoneLabel = new JLabel(telephoneLabelText.get(0));

private final Choice senderOrReceiverChoices = new Choice();

private final JButton orderButton = new JButton(orderButtonLabels.get(0));

private final JButton logOutButton = new JButton(orderButtonLabels.get(3));

private final JButton orderHistoryButton = new JButton(orderButtonLabels.get(2));

final JPanel titlePanel = new JPanel(new GridLayout(0, 1));

final JPanel mainPanel = new JPanel(new GridLayout(0, 1));

final JPanel dataPanel = new JPanel(new GridLayout(0, 2));

final JPanel messagePanel = new JPanel(new BorderLayout());

final JPanel lowerPanel = new JPanel(new BorderLayout());

final JPanel buttonPanel = new JPanel(new FlowLayout());

private final ArrayList<String> currentUserData;

public MainForm(String appName, ArrayList<String> currentUserData) {

super(appName);

this.currentUserData = currentUserData;

addWindowListener(this);

setLayout(new BorderLayout(20, 20));

String welcomeMessage = "Welcome, " + currentUserData.get(1) + " " + currentUserData.get(2);

setFormLabel(welcomeMessage);

setFormBody();

setLowerPanel();

add(titlePanel, BorderLayout.NORTH);

add(mainPanel, BorderLayout.CENTER);

add(buttonPanel, BorderLayout.SOUTH);

setResizable(false);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

pack();

this.setVisible(true);

showPanels(new ArrayList<>(Arrays.asList(titlePanel, mainPanel)));

}

private void setFormLabel(String welcomeMessage) {

titleLabel.setEditable(false);

titleLabel.setBackground(null);

titleLabel.setBorder(null);

titleLabel.setText(welcomeMessage);

titleLabel.setFont(new Font(null, Font.BOLD, 14));

titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

titlePanel.add(titleLabel, BorderLayout.NORTH);

}

private void setChooseActionPanel() {

JLabel senderOrReceiverLabel = new JLabel(actionChoiceLabel);

dataPanel.add(senderOrReceiverLabel);

senderOrReceiverText.forEach(senderOrReceiverChoices::add);

senderOrReceiverChoices.addItemListener(e -> {

String str = (String) e.getItem();

System.out.println(str);

if (notification.isVisible())

notification.setVisible(false);

if (str.equals(senderOrReceiverText.get(0))) {//"Receiver phone number"

telephoneLabel.setText(telephoneLabelText.get(0));

orderButton.setText(orderButtonLabels.get(0));

messagePanel.setVisible(false);

} else if (str.equals(senderOrReceiverText.get(1))) {//"Sender phone number"

telephoneLabel.setText(telephoneLabelText.get(1));

orderButton.setText(orderButtonLabels.get(1));

messagePanel.setVisible(false);

} else { //"Receiver phone number"

telephoneLabel.setText(telephoneLabelText.get(0));

orderButton.setText(orderButtonLabels.get(0));

messagePanel.setVisible(true);

}

});

dataPanel.add(senderOrReceiverChoices);

}

private void setTelephone() {

dataPanel.add(telephoneLabel);

dataPanel.add(telephone);

telephone.setText("+972");

telephone.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

int length = telephone.getText().length();

if ((length == 4 && e.getKeyChar() == '0') || !Character.isDigit(e.getKeyChar()) || (length == 13))

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if ((e.getKeyCode() == KeyEvent.VK\_DELETE || e.getKeyCode() == KeyEvent.VK\_BACK\_SPACE)

&& telephone.getBackground() == Color.ORANGE)

telephone.setBackground(Color.WHITE);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

}

private void setMessagePanel() {

messagePanel.add(new JLabel("Your message:"), BorderLayout.NORTH);

messagePanel.add(textArea, BorderLayout.SOUTH);

mainPanel.add(messagePanel, BorderLayout.SOUTH);

messagePanel.setVisible(false);

}

private void setNotification(String text, Boolean visible) {

notification.setText(text);

notification.setFont(new Font(null, Font.BOLD, 13));

notification.setVisible(visible);

notification.setHorizontalAlignment(SwingConstants.CENTER);

}

private void setFormBody() {

setChooseActionPanel();

setTelephone();

dataPanel.setVisible(true);

mainPanel.add(dataPanel, BorderLayout.NORTH);

setMessagePanel();

}

private void setLowerPanel() {

ArrayList<JButton> buttons = new ArrayList<>(Arrays.asList(orderButton, orderHistoryButton, logOutButton));

for (JButton button : buttons) {

button.setSize(40, 10);

button.setPreferredSize(new Dimension(120, 25));

buttonPanel.add(button);

button.addActionListener(this);

}

setNotification("Placeholder", false);

lowerPanel.add(notification, BorderLayout.NORTH);

lowerPanel.add(buttonPanel, BorderLayout.SOUTH);

}

@Override

public void actionPerformed(ActionEvent e) {

new Thread(() -> {

String selectedAction = senderOrReceiverChoices.getSelectedItem();

String providedTelephone = telephone.getText();

String buttonPressed = e.getActionCommand();

if (!buttonPressed.equals(logOutButton.getText()) && !buttonPressed.equals(orderHistoryButton.getText())) {

if (!ClientManager.validateTelephone(providedTelephone)) {

new ReceivedMessageForm("App", "The provided phone number is not registered in the system.");

return;

}

}

switch (buttonPressed) {

case "Send" -> processSendRequest(selectedAction, providedTelephone);

case "Receive" -> processReceiveRequest(providedTelephone);

case "Order history" -> processHistoryRequest();

case "Sign out" -> {

this.dispose();

ClientManager.openLoginForm();

}

}

}).start();

}

private void processSendRequest(String selectedAction, String providedTelephone) {

if (selectedAction.equals(senderOrReceiverText.get(0))) {

String parsedData = Parsers.parseMessageToServer(CommunicationCode.NewOrder, new ArrayList<>(Arrays.asList(currentUserData.get(9), providedTelephone)));

ClientManager.sendToServer(parsedData);

ClientManager.processServerResponse();

setNotification(orderSent, true);

} else if (selectedAction.equals(senderOrReceiverText.get(2))) {

String message = textArea.getText();

if (message.isEmpty()) return;

String parsedData = Parsers.parseMessageToServer(CommunicationCode.NewMessage, new ArrayList<>(Arrays.asList(currentUserData.get(9), providedTelephone, message)));

ClientManager.sendToServer(parsedData);

ClientManager.processServerResponse();

setNotification(messageSent, true);

}

}

private void processReceiveRequest(String providedTelephone) {

String parsedData = Parsers.parseMessageToServer(CommunicationCode.NewOrder, new ArrayList<>(Arrays.asList(providedTelephone, currentUserData.get(9))));

ClientManager.sendToServer(parsedData);

ClientManager.processServerResponse();

setNotification(orderSent, true);

}

private void processHistoryRequest() {

String parsedData = Parsers.parseMessageToServer(CommunicationCode.HistoryRequest, new ArrayList<>(Arrays.asList(currentUserData.get(9))));

ClientManager.sendToServer(parsedData);

ClientManager.processServerResponse();

}

@Override

public void windowOpened(WindowEvent e) {

}

@Override

public void windowClosing(WindowEvent e) {

}

@Override

public void windowClosed(WindowEvent e) {

this.dispose();

System.out.println("Window closed");

}

@Override

public void windowIconified(WindowEvent e) {

System.out.println("Minimized");

}

@Override

public void windowDeiconified(WindowEvent e) {

}

@Override

public void windowActivated(WindowEvent e) {

}

@Override

public void windowDeactivated(WindowEvent e) {

}

private void colorFieldInvalid(JTextField field) {

field.setBackground(Color.ORANGE);

}

private void showPanels(ArrayList<JPanel> panels) {

panels.forEach(panel -> panel.setVisible(true));

}

private void hidePanels(ArrayList<JPanel> panels) {

panels.forEach(panel -> panel.setVisible(false));

}

}

package DialogsUI;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

import java.util.ArrayList;

public class ReceivedMessageForm extends JFrame implements WindowListener {

/\* deeply messages \*/

private static final String appName = "Drone Post";

public ReceivedMessageForm(String sender, String text) {

super(appName);

JLabel senderLabel = new JLabel("New message from " + sender);

senderLabel.setHorizontalAlignment(SwingConstants.CENTER);

JTextArea textArea = new JTextArea(3, 30);

textArea.setEditable(false);

textArea.setLineWrap(true);

textArea.setWrapStyleWord(true);

textArea.setText(text);

JScrollPane scroll = new JScrollPane(textArea, JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

add(senderLabel, BorderLayout.NORTH);

add(scroll, BorderLayout.SOUTH);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

addWindowListener(this);

pack();

setResizable(false);

setVisible(true);

System.out.println(text);

}

public ReceivedMessageForm(String sender, ArrayList<String> orders) {

super(appName);

StringBuilder text = new StringBuilder();

for (String order: orders){

text.append(order).append("\n");

}

JLabel senderLabel = new JLabel("New message from " + sender);

senderLabel.setHorizontalAlignment(SwingConstants.CENTER);

JTextArea textArea = new JTextArea(5, 50);

textArea.setEditable(false);

textArea.setLineWrap(true);

textArea.setWrapStyleWord(true);

textArea.setText(text.toString());

JScrollPane scroll = new JScrollPane(textArea, JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

add(senderLabel, BorderLayout.NORTH);

add(scroll, BorderLayout.SOUTH);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

addWindowListener(this);

pack();

setResizable(false);

setVisible(true);

System.out.println(text);

}

@Override

public void windowOpened(WindowEvent e) {

}

@Override

public void windowClosing(WindowEvent e) {

}

@Override

public void windowClosed(WindowEvent e) {

this.dispose();

}

@Override

public void windowIconified(WindowEvent e) {

}

@Override

public void windowDeiconified(WindowEvent e) {

}

@Override

public void windowActivated(WindowEvent e) {

}

@Override

public void windowDeactivated(WindowEvent e) {

}

}

package DialogsUI;

import Enums.CommunicationCode;

import Enums.SubscriptionCode;

import Services.\*;

import org.javatuples.Pair;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.time.LocalDate;

import java.time.format.DateTimeParseException;

import java.util.ArrayList;

import java.util.Arrays;

public class RegistrationForm extends JFrame implements ActionListener, MouseListener, WindowListener {

private final ArrayList<String> columnLabel = new ArrayList<>(Arrays.asList("Personal ID", "First name", "Last name", "Date of birth (YYYY-mm-dd)", "Street", "Building", "City", "Zip Code", "E-mail", "Phone number", "Choose your plan"));

private final ArrayList<String> subscriptionPlan = new ArrayList<>(Arrays.asList("-", "50 shipments for $99", "150 shipments for $179"));

private static final ArrayList<String> formButtons = new ArrayList<>(Arrays.asList("Back to Login Form", "Clear Form", "Pay & Register"));

private final ArrayList<Integer> keyEvents = new ArrayList<>(Arrays.asList(KeyEvent.VK\_DELETE, KeyEvent.VK\_BACK\_SPACE));

private final String registrationLabel = "Client Registration Form";

private final String successMessageText = "Information saved successfully.";

private final String clientAlreadyExistsMessageText = "Your phone number is already in the database. Please update client details.";

private final String savingErrorMessageText = "Error occurred when saving the client information. Please try again.";

private final String dataErrorMessageText = "Marked fields are empty or contain invalid information.";

private final JTextField addLabel = new JTextField(registrationLabel);

private final JTextField personalId = new JTextField(10);

private final JTextField firstName = new JTextField(15);

private final JTextField lastName = new JTextField(15);

private final JTextField dateOfBirth = new JTextField(10);

private final JTextField street = new JTextField(20);

private final JTextField building = new JTextField(5);

private final JTextField city = new JTextField(15);

private final JTextField zipCode = new JTextField(10);

private final JTextField email = new JTextField(15);

private final JTextField telephone = new JTextField(9);

private final Choice programChoices = new Choice();

private final JTextField clientNotification = new JTextField("Placeholder");

private final ArrayList<JTextField> fields = new ArrayList<>(

Arrays.asList(personalId, firstName, lastName, dateOfBirth, street, building, city, zipCode, email, telephone));

protected final JButton backToLoginFormButton = new JButton(formButtons.get(0));

protected final JButton clearButton = new JButton(formButtons.get(1));

protected final JButton registerButton = new JButton(formButtons.get(2));

final JPanel titlePanel = new JPanel(new GridLayout(1, 1));

final JPanel registrationPanel = new JPanel(new GridLayout(0, 2));

final JPanel regFormButtons = new JPanel(new FlowLayout());

public RegistrationForm(String appName) {

super(appName);

addWindowListener(this);

setLayout((new BorderLayout(20, 15)));

setFormLabel();

//Add Personal Data Form

setFormBody();

//Add registration form panel buttons

setButtonsPanel(regFormButtons, new ArrayList<>(Arrays.asList(backToLoginFormButton, clearButton, registerButton)));

//Build the Frame

add(titlePanel, BorderLayout.NORTH);

add(registrationPanel, BorderLayout.CENTER);

add(regFormButtons, BorderLayout.SOUTH);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

pack();

setResizable(false);

this.setVisible(true);

titlePanel.setVisible(true);

registrationPanel.setVisible(true);

regFormButtons.setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e) {

String buttonClicked = e.getActionCommand();

switch (buttonClicked) {

case "Back to Login Form" -> processBackToLoginFormRequest();

case "Clear Form" -> processCleanFieldsRequest();

case "Pay & Register" -> processPayRegisterRequest();

}

}

private void processBackToLoginFormRequest() {

this.dispose();

ClientManager.openLoginForm();

}

private void processCleanFieldsRequest() {

fields.forEach(item -> item.setText(""));

telephone.setText("+972");

programChoices.select(0);

}

private void processPayRegisterRequest() {

try {

ArrayList<String> clientData = retrieveRegData();

if (clientData == null)

throw new NullPointerException("No client data received");

String parsedData;

if (validateInput())

parsedData = Parsers.parseMessageToServer(CommunicationCode.RegisterClient, clientData);

else throw new IllegalArgumentException(dataErrorMessageText);

ClientManager.sendToServer(parsedData);

String response = ClientManager.receiveFromServer();

Pair<CommunicationCode, ArrayList<String>> parsedMessage = Parsers.parseMessageFromServer(response);

switch (parsedMessage.getValue0()) {

case ClientRegistered -> {

new Services().saveClientDetailsToFile(clientData);

showSuccessNotification();

Thread.sleep(3000);

ClientManager.openOrderForm(this, clientData);

}

case ClientAlreadyExists -> showClientAlreadyExistsNotification();

case ClientSavingError -> showSavingErrorNotification();

}

} catch (IllegalArgumentException iae) {

showDataErrorNotification();

iae.printStackTrace();

} catch (InterruptedException io) {

io.printStackTrace();

}

}

private Boolean validateInput() {

boolean valid = true;

String date = dateOfBirth.getText();

try {

if (LocalDate.parse(date).isAfter(LocalDate.now())) {

valid = false;

colorFieldInvalid(dateOfBirth);

}

} catch (DateTimeParseException dtp) {

showDataErrorNotification();

}

String e\_mail = email.getText();

if (e\_mail.split("@").length != 2

|| !email.getText().split("@")[1].contains(".")) {

valid = false;

colorFieldInvalid(email);

}

if (telephone.getText().length() < 12) {

valid = false;

colorFieldInvalid(telephone);

}

String plan = programChoices.getSelectedItem();

if (plan.equals(SubscriptionCode.NO\_SUBSCRIPTION.getDescription())) {

valid = false;

programChoices.setBackground(Color.ORANGE);

}

fields.forEach(field -> {

if (field.getText().equals(""))

colorFieldInvalid(field);

});

return valid;

}

private ArrayList<String> retrieveRegData() {

ArrayList<String> data = new ArrayList<>();

fields.forEach(field -> data.add(field.getText()));

//Add subscription plan

data.add(SubscriptionCode.getCode(programChoices.getSelectedItem()));

return data.size() != fields.size() + 1 ? null : data;

}

private void setFormLabel() {

addLabel.setEditable(false);

addLabel.setBackground(null);

addLabel.setBorder(null);

addLabel.setFont(new Font(null, Font.BOLD, 14));

addLabel.setHorizontalAlignment(SwingConstants.CENTER);

titlePanel.add(addLabel);

}

private void setFormBody() {

setFields();

subscriptionPlan.forEach(programChoices::add);

for (int i = 0; i < fields.size(); ++i) {

registrationPanel.add(new Label(columnLabel.get(i)));

registrationPanel.add(fields.get(i));

}

registrationPanel.add(new Label(columnLabel.get(columnLabel.size() - 1)));

registrationPanel.add(programChoices);

setClientNotification();

registrationPanel.add(clientNotification);

}

private void setFields() {

personalId.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (!Character.isDigit(e.getKeyChar()))

e.setKeyChar(Character.MIN\_VALUE);

if (personalId.getText().length() == 10)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (personalId.getBackground() == Color.ORANGE)

colorFieldValid(personalId, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

firstName.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (firstName.getText().length() == 30)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (firstName.getBackground() == Color.ORANGE)

colorFieldValid(firstName, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

lastName.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (lastName.getText().length() == 30)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (lastName.getBackground() == Color.ORANGE)

colorFieldValid(lastName, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

dateOfBirth.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

int length = dateOfBirth.getText().length();

if (Character.isDigit(e.getKeyChar()) && length < 10) {

if (length == 4 || length == 7)

dateOfBirth.setText(dateOfBirth.getText() + "-");

} else e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (dateOfBirth.getBackground() == Color.ORANGE)

colorFieldValid(dateOfBirth, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

street.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (street.getText().length() == 40)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (street.getBackground() == Color.ORANGE)

colorFieldValid(street, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

building.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (building.getText().length() == 5)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (building.getBackground() == Color.ORANGE)

colorFieldValid(building, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

city.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (city.getText().length() == 30)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (city.getBackground() == Color.ORANGE)

colorFieldValid(city, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

zipCode.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

if (zipCode.getText().length() == 10)

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (zipCode.getBackground() == Color.ORANGE)

colorFieldValid(zipCode, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

email.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

String text = email.getText();

if (text.contains("@") && e.getKeyChar() == '@')

e.setKeyChar(Character.MIN\_VALUE);

if (text.length() == 45) {

e.setKeyChar(Character.MIN\_VALUE);

if (!email.getText().contains("@") || !email.getText().contains("."))

colorFieldInvalid(email);

}

}

@Override

public void keyPressed(KeyEvent e) {

if (email.getBackground() == Color.ORANGE)

colorFieldValid(email, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

telephone.setText("+972");

telephone.addKeyListener(new KeyListener() {

@Override

public void keyTyped(KeyEvent e) {

int length = telephone.getText().length();

if ((length == 4 && e.getKeyChar() == '0') || !Character.isDigit(e.getKeyChar()) || (length == 13))

e.setKeyChar(Character.MIN\_VALUE);

}

@Override

public void keyPressed(KeyEvent e) {

if (telephone.getBackground() == Color.ORANGE)

colorFieldValid(telephone, e);

}

@Override

public void keyReleased(KeyEvent e) {

}

});

programChoices.addItemListener(e -> {

if (programChoices.getBackground() == Color.ORANGE

&& e.getItem() != subscriptionPlan.get(0))

programChoices.setBackground(Color.WHITE);

});

}

private void setButtonsPanel(JPanel panel, ArrayList<JButton> buttons) {

buttons.forEach(item -> {

panel.add(item);

item.addActionListener(this);

});

}

private void setClientNotification() {

clientNotification.setBackground(null);

clientNotification.setBorder(null);

clientNotification.setForeground(Color.RED);

hideNotification();

}

private void colorFieldInvalid(JTextField field) {

field.setBackground(Color.ORANGE);

}

private void colorFieldValid(JTextField field, KeyEvent e) {

ArrayList<Integer> keyEvents = this.keyEvents;

if (field == email) {

keyEvents.add(KeyEvent.VK\_AT);

keyEvents.add(KeyEvent.VK\_PERIOD);

}

if (Character.isDigit(e.getKeyChar())

|| Character.isAlphabetic(e.getKeyChar())

|| keyEvents.contains(e.getKeyCode()))

field.setBackground(Color.WHITE);

}

private void showClientAlreadyExistsNotification() {

clientNotification.setText(clientAlreadyExistsMessageText);

clientNotification.setForeground(Color.RED);

clientNotification.setVisible(true);

}

private void showDataErrorNotification() {

clientNotification.setText(dataErrorMessageText);

clientNotification.setForeground(Color.RED);

clientNotification.setVisible(true);

}

private void showSavingErrorNotification() {

clientNotification.setText(savingErrorMessageText);

clientNotification.setForeground(Color.RED);

clientNotification.setVisible(true);

}

private void showSuccessNotification() {

clientNotification.setText(successMessageText);

clientNotification.setForeground(Color.GREEN);

clientNotification.setVisible(true);

}

private void hideNotification() {

clientNotification.setVisible(false);

}

@Override

public void mouseClicked(MouseEvent e) {

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseReleased(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

@Override

public void windowOpened(WindowEvent e) {

}

@Override

public void windowClosing(WindowEvent e) {

}

@Override

public void windowClosed(WindowEvent e) {

this.dispose();

}

@Override

public void windowIconified(WindowEvent e) {

}

@Override

public void windowDeiconified(WindowEvent e) {

}

@Override

public void windowActivated(WindowEvent e) {

}

@Override

public void windowDeactivated(WindowEvent e) {

}

}

import Services.ClientManager;

public class Main {

public static void main (String [] args) {

ClientManager.startClient();

}

}

SQL code:

CREATE SCHEMA `dronepost` ;

CREATE TABLE `dronepost`.`clients` (

`clientId` INT NOT NULL AUTO\_INCREMENT,

`personalId` CHAR(10) NOT NULL,

`firstName` CHAR(30) NULL,

`lastName` CHAR(30) NOT NULL,

`dateOfBirth` DATE NULL,

`street` CHAR(40) NOT NULL,

`building` CHAR(5) NULL,

`city` CHAR(30) NOT NULL,

`zipCode` CHAR(10) NOT NULL,

`email` CHAR(45) NOT NULL,

`telephone` CHAR(15) NOT NULL Unique,

`subscriptionCode` CHAR(1) NOT NULL,

`posX` INT NOT NULL,

`posY` INT NOT NULL,

`numberOfDeliveriesLeft` CHAR(5),

`isDeleted` INT(1) NOT NULL Default 0,

PRIMARY KEY (`clientId`));

CREATE TABLE `dronepost`.`orders` (

`orderId` INT NOT NULL AUTO\_INCREMENT,

`senderId` INT NOT NULL,

`recipientId` INT NOT NULL,

`orderDateTime` DATETIME NOT NULL,

`orderStatus` CHAR(20) NOT NULL,

`assignedDroneId` INT(4) NOT NULL,

`withReturn` INT(1) NOT NULL,

`isDeleted` INT(1) NOT NULL Default 0,

PRIMARY KEY (`orderId`),

CONSTRAINT `senderId`

FOREIGN KEY (`senderId`)

REFERENCES `dronepost`.`clients` (`clientId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `recipientId`

FOREIGN KEY (`recipientId`)

REFERENCES `dronepost`.`clients` (`clientId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `dronepost`.`drones` (

`droneId` INT NOT NULL Unique,

`droneModel` CHAR(25) NOT NULL,

`droneStatus` CHAR(10) NOT NULL,

`isDeleted` INT(1) UNSIGNED NOT NULL DEFAULT 0,

PRIMARY KEY (`droneId`));

CREATE TABLE `dronepost`.`notifications` (

`notificationId` INT NOT NULL AUTO\_INCREMENT,

`recipientId` INT NOT NULL,

`orderId` INT NOT NULL,

`sentDateTime` DATETIME NOT NULL,

`message` VARCHAR(500) NULL,

`delivered` INT(1) NULL DEFAULT 0,

PRIMARY KEY (`notificationId`),

INDEX `recipientId\_idx` (`recipientId` ASC) ,

INDEX `orderId\_idx` (`orderId` ASC) ,

CONSTRAINT `recipientIdConstraint`

FOREIGN KEY (`recipientId`)

REFERENCES `dronepost`.`clients` (`clientId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `orderIdConstraint`

FOREIGN KEY (`orderId`)

REFERENCES `dronepost`.`orders` (`orderId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `dronepost`.`messages` (

`messageId` INT NOT NULL AUTO\_INCREMENT,

`senderId` INT NOT NULL,

`recipientId` INT NOT NULL,

`sentDateTime` DATETIME NOT NULL,

`message` VARCHAR(500) NULL,

`delivered` INT(1) NULL DEFAULT 0,

PRIMARY KEY (`messageId`),

INDEX `senderId\_idx` (`senderId` ASC) ,

INDEX `recipientId\_idx` (`recipientId` ASC) ,

CONSTRAINT `senderIdConstraint1`

FOREIGN KEY (`senderId`)

REFERENCES `dronepost`.`clients` (`clientId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `recipientIdConstraint1`

FOREIGN KEY (`recipientId`)

REFERENCES `dronepost`.`clients` (`clientId`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);