МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 8
З дисципліни: *“Кросплатформне програмування”*
На тему: *“Огляд основ та можливостей Spring (проект)”*

Лектор:

доц. каф. ПЗ

Дяконюк Л.М.

Виконали:

ст. гр. ПЗ-32

Герман А.-С. Р.
Коваленко Д.М.
Ковальов Р.К.
Козира О.Ф.
Руденко А.М.
Снісар В.І.

Прийняв:

ст. викл. каф. ПЗ

Шкраб Р.Р.

« ____ » _____ 2023 р.

∑= _____ .....

Львів – 2023

Тема роботи: Симулятор роботи залізничних кас.

Мета роботи: Створити додаток, що симулює роботу залізничних кас.

ЗАВДАННЯ

Створити проєкт, який реалізує запропоноване завдання. Відобразити заплановану діаграму взаємозв’язків між класами. При проектуванні використати паттерни з обґрунтуванням їх доцільності. Передбачити опрацювання типових виняткових ситуацій. Вибір технології для візуалізації програми відбувається студентами. При проєктуванні системи врахувати, що вона може еволюціонувати, а супровід програмного забезпечення має бути відбуватися з якомога меншими затратами.

Зокрема, слід передбачити, що в подальшому можлива зміна візуального відображення системи. Захист командної роботи передбачає створення презентації з зазначенням прізвищ учасників та зазначенням частини виконаної роботи, а також демонстрації роботи програми, обґрунтування вибраної моделі проєктування та аналізу застосованих технологій.

Варіант 1. Описати симулятор роботи залізничних кас

Програма повинна задовольняти наступні вимоги:

1. Можливість конфігурувати систему, визначаючи кількість кас та координати їх розташувань в приміщенні, визначати кількість входів в приміщення, в яких можуть з’являтись потенційні клієнти, визначати в певному діапазоні норму часу, який буде витрачатись на обслуговування одного квитка.

2. Програма повинна автоматично генерувати клієнтів за різними стратегіями щодо часу появи(через рівномірні проміжки часу, через випадкові проміжки часу і т. д.). Клієнт може мати особливий статус (інвалід, клієнт з дитиною, інша пільгова категорія). Наявність статусу визначає пріоритет обслуговування в касі.

3. Створені клієнти повинні мати намір купити задану генератором випадкову кількість квитків.

4. Клієнт має можливість обирати одну з кас за принципом вибору тієї, в черзі до якої є найменша кількість людей. Якщо кількість людей в черзі є однаковою, то клієнт обирає ту, яка є найближчою.

5. Одна з кас є резервною, тобто такою, яка обслуговує клієнтів при технічній несправності штатних кас. У випадку закриття каси на технічну перерву, клієнти з черги до цієї каси повинні бути переміщені до резервної каси зі збереженням порядку слідування.

6. У випадку, якщо сумарна кількість людей в приміщенні, перевищує допустиму норму, вокзал обмежує доступ до приміщення, генератор клієнтів повинен припинити генерацію до тих пір, поки кількість людей не стане меншою за 70% за норму.

7. У випадку, коли клієнта обслужили в касі, вважати, що він негайно покидає приміщення.

8. Візуалізація системи передбачає відображення переміщення та процес формування можливих черг та переміщення осіб в межах черги. Процес обслуговування клієнта повинен бути зафіксований в журналі логування з зазначеним номером каси, яка обслуговувала клієнта, унікальним номером клієнта та стартовим і кінцевим часом обслуговування клієнта біля каси.

ХІД ВИКОНАННЯ

Початкові діаграми:

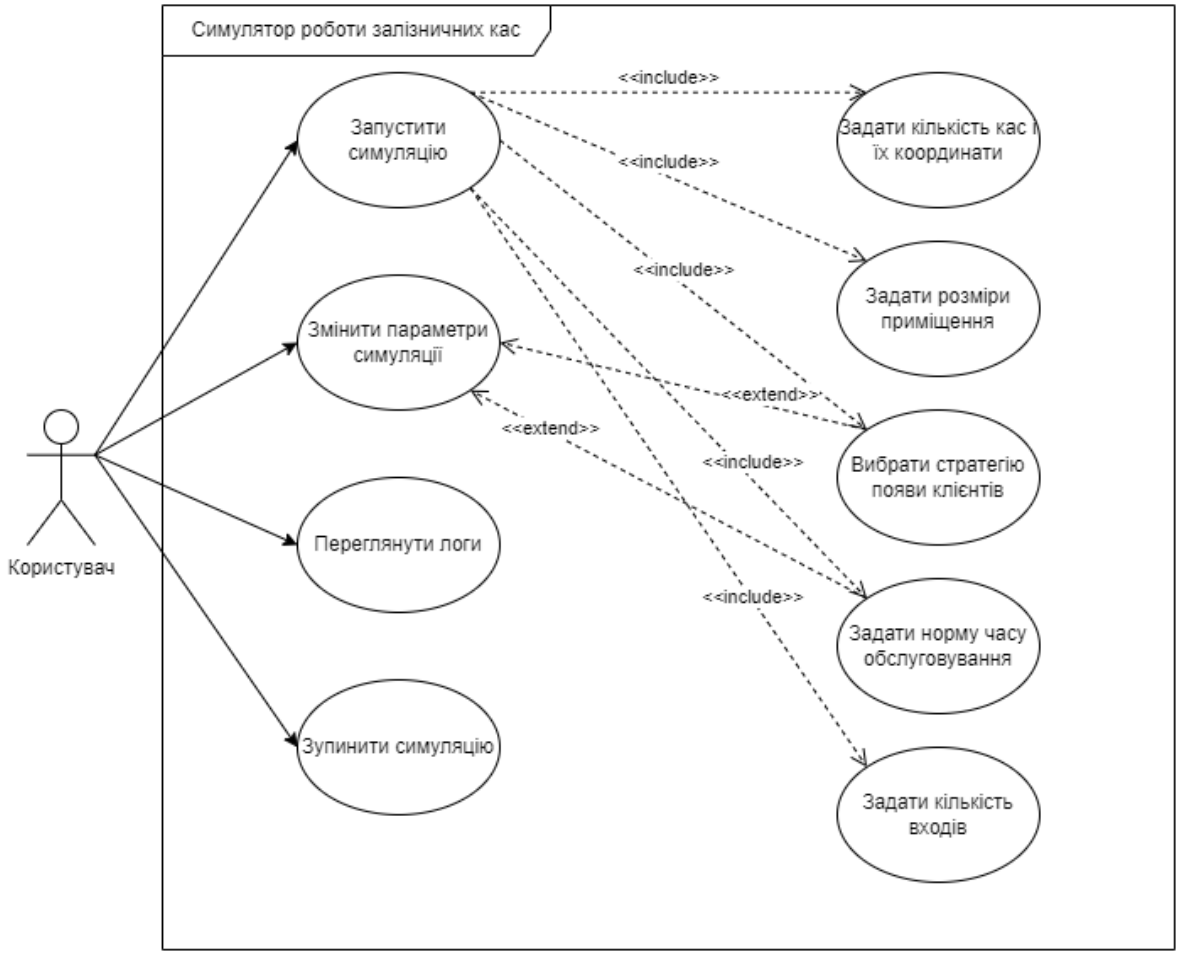

Рис.1. Початкова діаграми прецедентів

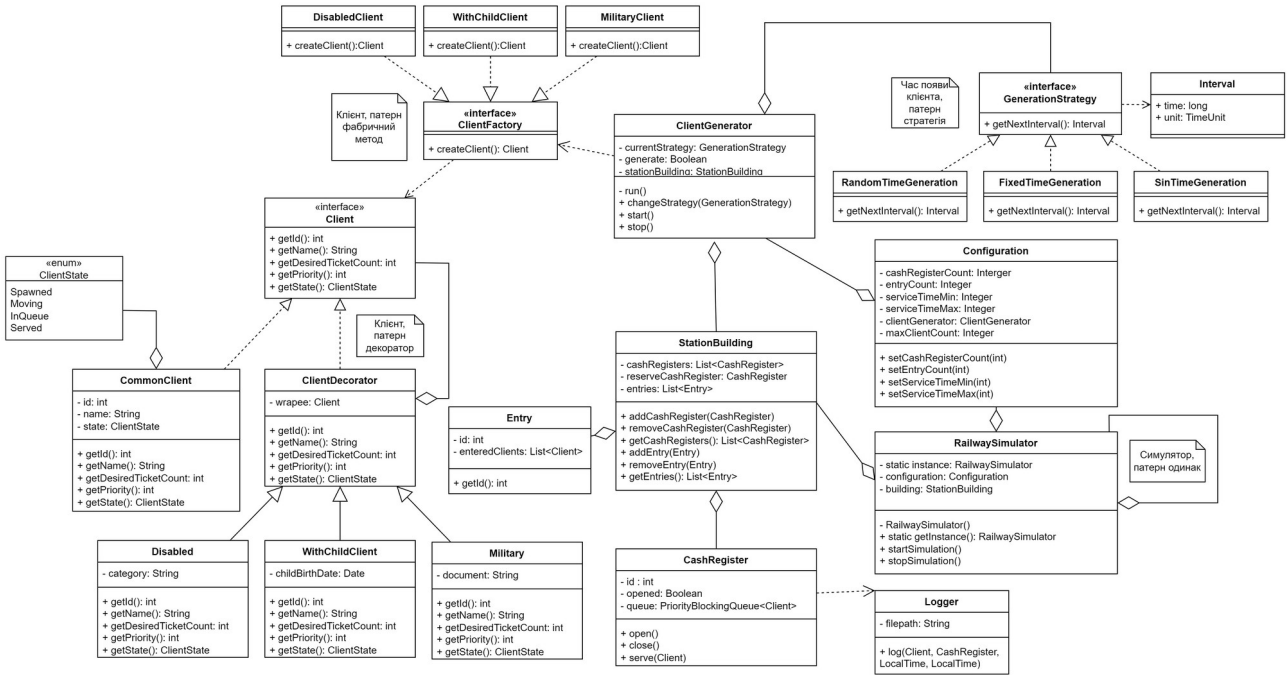

Рис.2. Початкова діаграма класів

Фінальні діаграми:

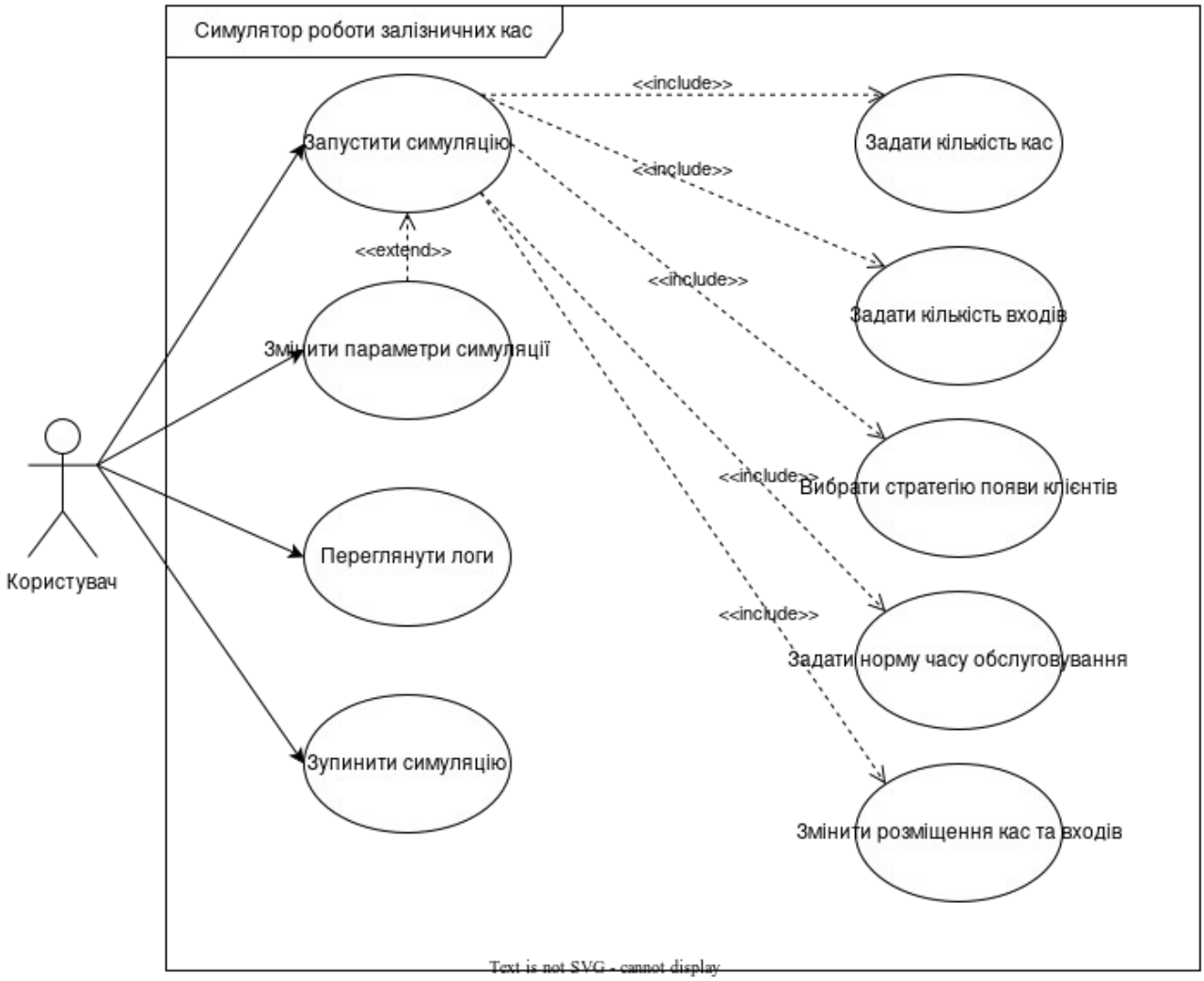

Рис.3. Фінальна діаграма прецедентів

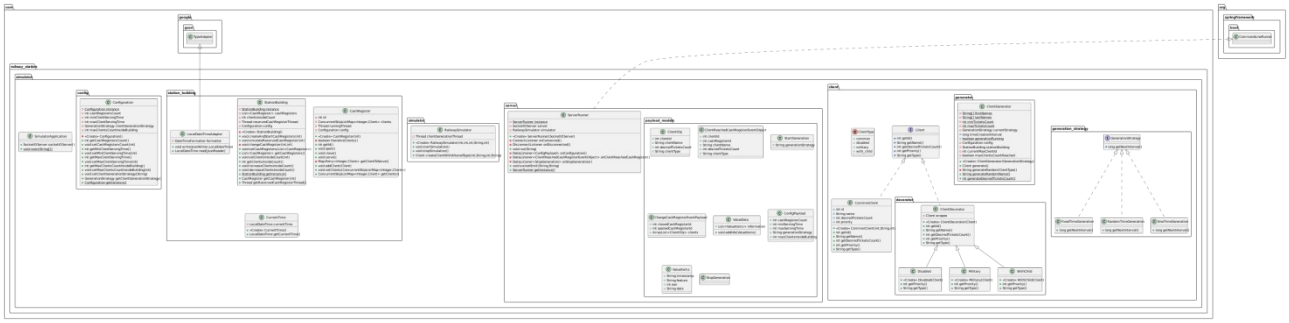

Рис.4. Фінальна діаграма класів

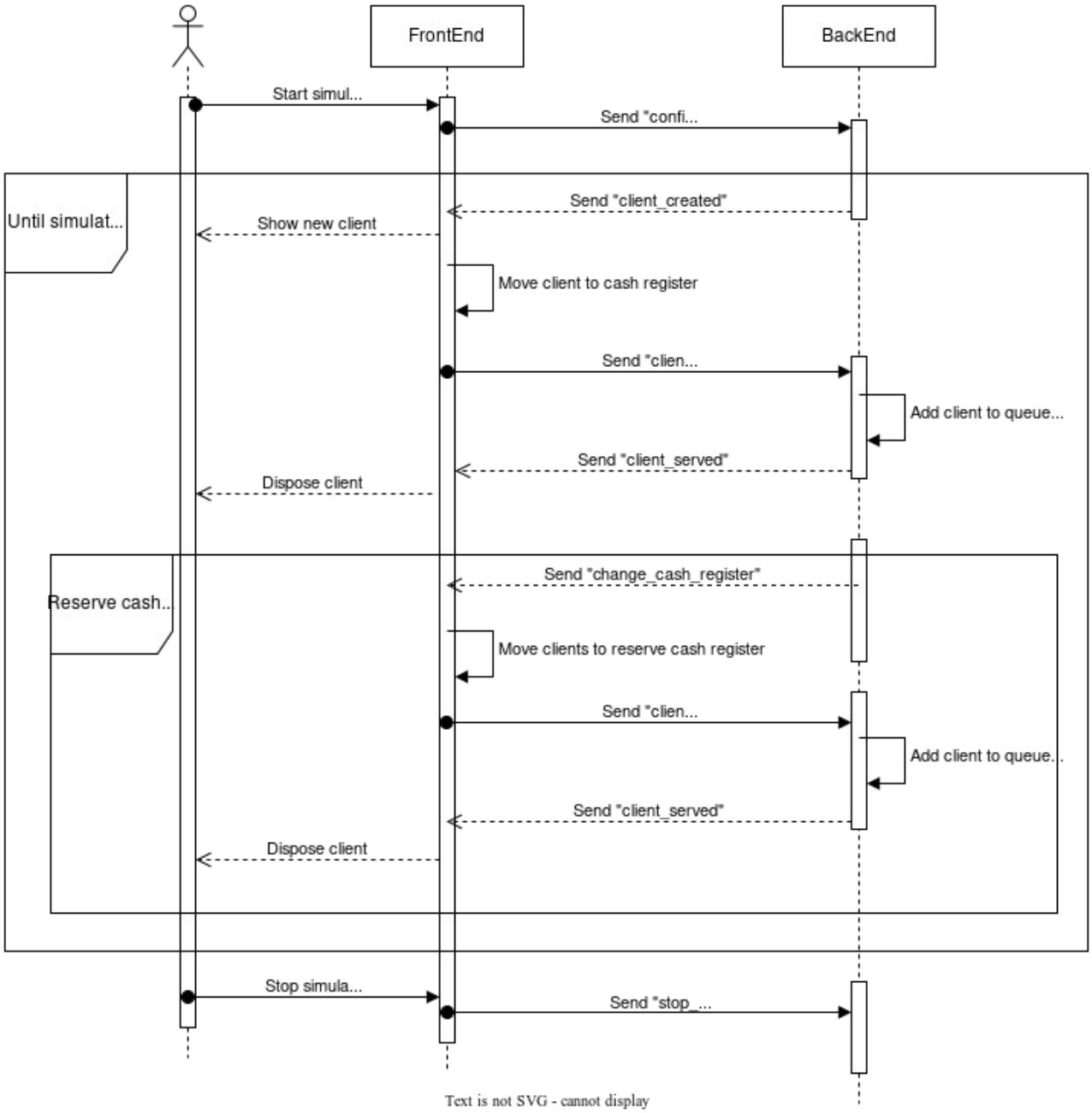

Рис.5. Діаграма послідовностей

Оригінали діаграм у хорошій якості та інші артефакти є на диску: https://drive.google.com/drive/u/0/folders/1ZrvOudPxRgIwU95F73QqLJ3fDRLkrhlt

Реалізовані патерни проектування:

1. Декоратор – використаний для створення пільгових категорій клієнтів, таким чином кожна пільга декорує звичайного клієнта і збільшує свій пріоритет в черзі, змінюючи метод *getPriority*. Також цей патерн дозволяє створювати комбіновані пільги, які матимуть ще вищий пріоритет.
2. Стратегія – необхідна для задавання стратегії генерації клієнтів у часі.

3. Сінглтон – застосований до класу конфігурації та класу, що запускає сервер, так як немає необхідності в кількох екземплярах.

Розподіл обов’язків:

Герман Андрій-Симон - Backend Developer - реалізація патернів та основних класів предметної області

Коваленко Дмитро - Project Manager, System Architect - розробка архітектури сервера, робота з вимогами

Ковальов Ренат - Frontend Developer - імплементація фронта

Козира Олександр - Team Lead, System Architect - керування командою, розробка архітектури

Руденко Альберт - Frontend Developer, UI/UX Designer - створення дизайну, фронт

Снісар Владислав - Backend Developer - налаштування сокетів, основна логіка серверу

Використані технології:

Spring Boot – запуск серверної частини проекту

SocketIO – зв’язок між сервером та клієнтом (REST не підходить – на сервері постійно генеруються події)

Kotlin + Jetpack Compose – клієнт

Основні класи:

Client

```
package com.railway_station.simulator.client;

public interface Client {
    int getId();
    String getName();
    int getDesiredTicketsCount();
    int getPriority();
    String getType();
}
```

CommonClient

```
package com.railway_station.simulator.client;

public class CommonClient implements Client {
    int id;
    String name;
    int desiredTicketsCount;
    int priority = 1;
    public CommonClient(int id, String name, int desiredTicketsCount) {
        this.id = id;
        this.name = name;
        this.desiredTicketsCount = desiredTicketsCount;
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public int getDesiredTicketsCount() {
        return desiredTicketsCount;
    }

    @Override
    public int getPriority() {
        return priority;
    }

    @Override
    public String getType() {
        return "common";
    }
}
```

ClientDecorator

```
package com.railway_station.simulator.client.decorator;

import com.railway_station.simulator.client.Client;

public class ClientDecorator implements Client {
    private Client wrapee;

    public ClientDecorator(Client client) {
        this.wrapee = client;
    }

    …
}
```

Disabled

```
package com.railway_station.simulator.client.decorator;

import com.railway_station.simulator.client.Client;

public class Disabled extends ClientDecorator {
```

```
    public Disabled(Client client) {
        super(client);
    }

    @Override
    public int getPriority() {
        return super.getPriority() * 4;
    }

    @Override
    public String getType() {
        return "disabled";
    }
}
```

GenerationStrategy

```
package com.railway_station.simulator.client.generation_strategy;

public interface GenerationStrategy {
    long getNextInterval();
}
```

FixedTimeGeneration

```
package com.railway_station.simulator.client.generation_strategy;

import com.railway_station.simulator.config.Configuration;

public class FixedTimeGeneration implements GenerationStrategy {
    @Override
    public long getNextInterval() {
        Configuration config = Configuration.getInstance();
        long minServingTime = config.getMinClientServingTime();
        long maxServingTime = config.getMaxClientServingTime();
        return (minServingTime + maxServingTime) / 2;
    }
}
```

ClientGenerator

```
package com.railway_station.simulator.client.generator;

import com.google.gson.Gson;
import com.railway_station.simulator.client.Client;
import com.railway_station.simulator.client.ClientType;
import
com.railway_station.simulator.client.generation_strategy.GenerationStrategy;
import com.railway_station.simulator.config.Configuration;
import com.railway_station.simulator.server.ServerRunner;
import com.railway_station.simulator.station_building.StationBuilding;

import java.util.*;
import java.util.concurrent.TimeUnit;

public class ClientGenerator {
    private static final String[] firstNames = {"Alice", "Bob", "Charlie",
"David", "Emma", "Frank", "Grace", "Henry", "Ivy", "Jack", "Katherine", "Leo",
"Mia", "Nathan", "Olivia"};
    private static final String[] lastNames = {"Smith", "Johnson", "Williams",
"Jones", "Brown", "Davis", "Miller", "Wilson", "Moore", "Taylor", "Anderson",
"Thomas", "Jackson", "White", "Harris"};
```

```
    private static final int minTicketsCount = 1;
    private static final int maxTicketsCount = 3;

    private GenerationStrategy currentStrategy;
    long timeCreationInterval;
    public static boolean generationRunning = false;
    private Configuration config = Configuration.getInstance();
    private StationBuilding stationBuilding = StationBuilding.getInstance();
    private int currentMaxClientId = -1;
    private boolean maxClientsCountReached = false;

    public ClientGenerator(GenerationStrategy strategy) {
        this.currentStrategy = strategy;
        timeCreationInterval = this.currentStrategy.getNextInterval();
    }

    public Client generate() {
        ClientGenerator.generationRunning = true;
        while (true) {
            try {
                TimeUnit.MILLISECONDS.sleep(timeCreationInterval);
            } catch (InterruptedException e) {
                return null;
            }
            if (stationBuilding.getClientsInsideCount() >=
config.getMaxClientsCountInsideBuilding()) {
                maxClientsCountReached = true;
                continue;
            } else if (maxClientsCountReached &&
(stationBuilding.getClientsInsideCount() >=
(config.getMaxClientsCountInsideBuilding() * 0.7))) {
                continue;
            } else {
                maxClientsCountReached = false;
            }
            String clientName = generateRandomName();
            int desiredTicketsCount = generateDesiredTicketsCount();
            String clientType = generateRandomClientType();

            Map<String, Object> clientData = new HashMap<>();
            currentMaxClientId++;
            clientData.put("clientId", currentMaxClientId);
            clientData.put("clientName", clientName);
            clientData.put("desiredTicketsCount", desiredTicketsCount);
            clientData.put("clientType", clientType);

            Gson gson = new Gson();
            String json = gson.toJson(clientData);

            ServerRunner server = ServerRunner.getInstance();
            server.socketEmit("client_created", json);
            StationBuilding.getInstance().increaseClientsInsideCount();
        }
    }

    private String generateRandomClientType() {
        List<ClientType> clientTypes = new ArrayList<>(){};
        clientTypes.add(ClientType.common);
        clientTypes.add(ClientType.disabled);
        clientTypes.add(ClientType.military);
        clientTypes.add(ClientType.with_child);

        Random random = new Random();
        int randomIndex = random.nextInt(clientTypes.size());
        return clientTypes.get(randomIndex).name();
    }
```

```
    public static String generateRandomName() {
        Random random = new Random();
        String firstName = firstNames[random.nextInt(firstNames.length)];
        String lastName = lastNames[random.nextInt(lastNames.length)];
        return firstName + " " + lastName;
    }

    public static int generateDesiredTicketsCount() {
        Random random = new Random();
        return random.nextInt(maxTicketsCount - minTicketsCount + 1) +
minTicketsCount;
    }
}
```

RailwaySimulator

```
package com.railway_station.simulator.simulator;

import com.railway_station.simulator.client.Client;
import com.railway_station.simulator.client.ClientType;
import com.railway_station.simulator.client.CommonClient;
import com.railway_station.simulator.client.decorator.Disabled;
import com.railway_station.simulator.client.decorator.Military;
import com.railway_station.simulator.client.decorator.WithChild;
import
com.railway_station.simulator.client.generation_strategy.GenerationStrategy;
import com.railway_station.simulator.client.generator.ClientGenerator;
import com.railway_station.simulator.config.Configuration;
import com.railway_station.simulator.station_building.CashRegister;
import com.railway_station.simulator.station_building.StationBuilding;

import java.util.ArrayList;
import java.util.List;

public class RailwaySimulator {
    private Thread clientGenerationThread;

    public RailwaySimulator(int cashRegistersCount, int minServingTime, int
maxServingTime, String generationStrategy, int maxClientsCountInsideBuilding) {
        Configuration config = Configuration.getInstance();
        config.setCashRegistersCount(cashRegistersCount);
        config.setMinClientServingTime(minServingTime);
        config.setMaxClientServingTime(maxServingTime);
        config.setClientGenerationStrategy(generationStrategy);
        config.setMaxClientsCountInsideBuilding(maxClientsCountInsideBuilding);
    }


    public void startSimulation() {
        Configuration config = Configuration.getInstance();
        StationBuilding stationBuilding = StationBuilding.getInstance();

        int cashRegistersCount = config.getCashRegistersCount();
        GenerationStrategy clientGenerationStrategy =
config.getClientGenerationStrategy();
        ClientGenerator clientGenerator = new
ClientGenerator(clientGenerationStrategy);

        stationBuilding.createAndStartCashRegisters(cashRegistersCount);
        this.clientGenerationThread = new Thread(clientGenerator::generate);
        clientGenerationThread.start();
    }

    public void stopSimulation() {
```

```
        StationBuilding stationBuilding = StationBuilding.getInstance();
        Thread reservecCashRegisterThread =
stationBuilding.getReservedCashRegisterThread();
        if (reservecCashRegisterThread != null) {
            reservecCashRegisterThread.interrupt();
        }
        List<CashRegister> cashRegisters = stationBuilding.getCashRegisters();
        for (var cashRegister: cashRegisters) {
            cashRegister.close();
        }
        stationBuilding.setCashRegisters(new ArrayList<CashRegister>());
        stationBuilding.setClientsInsideCount(0);
        if (clientGenerationThread != null) {
            clientGenerationThread.interrupt();
        }
    }

    public Client createClientWithAlteredType(int clientId, String clientName,
int desiredTicketsCount, String clientType) {
        Client client = new CommonClient(clientId, clientName,
desiredTicketsCount);
        if (clientType == ClientType.disabled.name()) {
            client = new Disabled(client);
        }
        if (clientType == ClientType.military.name()) {
            client = new Military(client);
        }
        if (clientType == ClientType.with_child.name()) {
            client = new WithChild(client);
        }
        return client;
    }
}
```

ServerRunner

```
package com.railway_station.simulator.server;

import com.corundumstudio.socketio.SocketIOServer;
import com.corundumstudio.socketio.listener.ConnectListener;
import com.corundumstudio.socketio.listener.DataListener;
import com.corundumstudio.socketio.listener.DisconnectListener;
import com.railway_station.simulator.client.Client;
import
com.railway_station.simulator.server.payload_models.ClientReachedCashRegisterEve
ntObject;
import com.railway_station.simulator.server.payload_models.ConfigPayload;
import com.railway_station.simulator.server.payload_models.StopGeneration;
import com.railway_station.simulator.simulator.RailwaySimulator;
import com.railway_station.simulator.station_building.CashRegister;
import com.railway_station.simulator.station_building.StationBuilding;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class ServerRunner implements CommandLineRunner {

    private static ServerRunner instance;
    private final SocketIOServer server;
    private RailwaySimulator simulator;

    @Autowired
    public ServerRunner(SocketIOServer server) {
        this.server = server;
```

```
        ServerRunner.instance = this;
        this.server.addConnectListener(onConnected());
        this.server.addDisconnectListener(onDisconnected());
        this.server.addEventListener("configuration", ConfigPayload.class,
onConfiguration());
        this.server.addEventListener("client_reached_cash_register",
ClientReachedCashRegisterEventObject.class, onClientReachedCashRegister());
        this.server.addEventListener("stop_generation", StopGeneration.class,
onStopGeneration());
    }

    private ConnectListener onConnected() {
        return client -> {
            System.out.println("Connected");
        };
    }

    private DisconnectListener onDisconnected() {
        return client -> {
            System.out.println("Disconnected");
            if (simulator != null) {
                simulator.stopSimulation();
            }
        };
    }

    @Override
    public void run(String... args) throws Exception {
        server.start();
        Thread.sleep(Integer.MAX_VALUE);
        server.stop();
    }

    // handler for "configuration" event
    private DataListener<ConfigPayload> onConfiguration() {
        return (client, configPayload, ackSender) -> {
            if (simulator != null) {
                simulator.stopSimulation();
            }

            int cashRegistersCount = configPayload.cashRegisterCount;
            int minServingTime = configPayload.minServingTime;
            int maxServingTime = configPayload.maxServingTime;
            String generationStrategy = configPayload.generationStrategy;
            int maxClientsCountInsideBuilding =
configPayload.maxClientsInsideBuilding;

            simulator = new RailwaySimulator(cashRegistersCount, minServingTime,
maxServingTime, generationStrategy, maxClientsCountInsideBuilding);
            simulator.startSimulation();
        };
    }

    // handler for "client_reached_cash_register" event
    private DataListener<ClientReachedCashRegisterEventObject>
onClientReachedCashRegister() {
        return (client, eventPayload, ackSender) -> {
            int cashRegisterId = eventPayload.cashRegisterId;
            int clientId = eventPayload.clientId;
            String clientName = eventPayload.clientName;
            int desiredTicketsCount = eventPayload.desiredTicketsCount;
            String clientType = eventPayload.clientType;

            Client stationClient =
simulator.createClientWithAlteredType(clientId, clientName, desiredTicketsCount,
clientType);
```

```
            CashRegister cashRegister =
StationBuilding.getInstance().getCashRegister(cashRegisterId);
            cashRegister.addClient(stationClient);
        };
    }

    // handler for "stop_generation" event
    private DataListener<StopGeneration> onStopGeneration() {
        return (client, eventPayload, ackSender) -> {
            if (simulator != null) {
                simulator.stopSimulation();
            }
        };
    }

    public void socketEmit(String eventName, String jsonifiedData) {
        server.getBroadcastOperations().sendEvent(eventName, jsonifiedData);
    }

    public static ServerRunner getInstance() {
        return instance;
    }

}
```

SimulatorApplication

```
package com.railway_station.simulator;

import com.corundumstudio.socketio.Configuration;
import com.corundumstudio.socketio.SocketIOServer;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class SimulatorApplication {
    @Bean
    public SocketIOServer socketIOServer() {
       Configuration config = new Configuration();
       config.setHostname("localhost");
       config.setPort(9092);

       final SocketIOServer server = new SocketIOServer(config);
       return server;
    }

    public static void main(String[] args) {
       SpringApplication.run(SimulatorApplication.class, args);
    }

}
```

Весь код доступний на github: https://github.com/geekylthyosaur/kpp/tree/dev

Результати виконання програми:

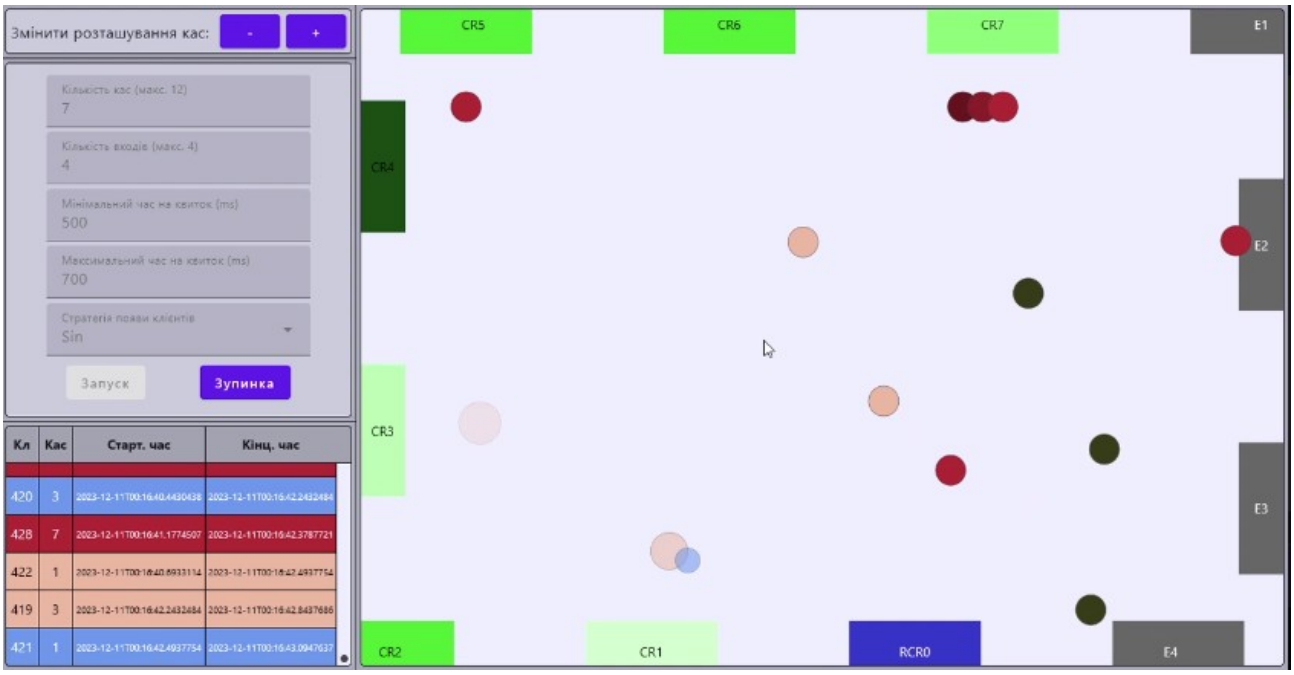

Рис.6. Вигляд програми

Відео роботи також є на диску.

ВИСНОВКИ

Під час виконання даної лабораторної роботи було розроблено в команді додаток, що симулює роботу залізничних кас, з можливістю конфігурувати систему відповідно до вимог. Для реалізації використано багатопотоковість та кілька патернів проектування для гнучкості системи, розроблено необхідні артефакти (діаграми), обрано оптимальну архітектуру, що використовує сокети для взаємодії між клієнтом та сервером, відтворено рух пасажирів по кривій Безьє із колізіями.