Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт

по курсу «Программное обеспечение распределённых вычислительных систем»

по теме «Разработка системы "Система страхования"»

Выполнил студент гр. 3540901/81501:
Ерниязов Т. Е.

Проверил преподаватель:
Стручков И. В.

Санкт-Петербург
2020 г.

Содержание

1 Анализ задания

1.1 Формулировка задания

Необходимо спроектировать и реализовать систему "Сервис страхования здоровья"(Insurance Service), которая предназначена для автоматизации процессов страхования. Система должна предоставлять пользователям возможность приобретения полиса страхования. Также система должна позволять выполнять запрос на выплату в случае страховых случаев. Более того должна быть доступна возможность обновления данных о здоровье клиента.

1.2 Функциональные требования

Клиент - участник, инициирующий процесс оформления/закрытия полиса и получения страховых выплат через оператора страховой компании. Он может:

- оформление заявок на оформление полиса/обновление данных о здоровье/получение страховых выплат
- отказ от страхования

Оператор - участник, оформляющий запросы на оформление полиса от клиентов, уведомляющий страховых агентов о необходимости провести расследование страхового случая, а также выносящий вердикт о страховой выплате. В его обязанности входит:

- приём и обработка любых заявок от клиента
- вынесение вердикта по данным от страхового клиента по заявкам клиента

Страховой агент - участник, который занимается оценкой здоровья клиента и проводит расследование, в случае если клиент требует страховые выплаты. Он выполняет следующие действия:

- проводит первоначальную оценку здоровья клиента
- рассматривает страховые случаи и назначает выплаты.

1.3 Описание бизнес-процессов

1.4 Оформление полиса

Участники

- Клиент
- Оператор
- Страховой агент

Этапы

- Оформление заявки - подпроцесс, при котором клиент указывает свои контактные данные, выбирает опциональные услуги. Клиент может оформить заявку самостоятельно через интерфейс пользователя. Конечным результатом оформления заказа является запись всех необходимых данных о заказе в базу данных, а также отображение данных о заявке в интерфейсе пользователя, доступное только операторам и страховым агентам.
- Оценка состояния здоровья - подпроцесс, при котором страховой агент связывается с клиентом и направляет его на обследование. Информирование страховых агентов о заказе включается в обязанности оператора. После обследования страховые агенты вносят данные о здоровье клиента в базу и переводят статус заявки клиента в состояние готовой к оплате.
- Оплата заказа - подпроцесс, при котором клиент оплачивает страхование здоровья.

1.5 Выплата денег по страховому случаю

Участники

- Клиент
- Оператор
- Страховой агент

Этапы

- Запрос о страховой выплате - подпроцесс, при котором клиент сообщает через интерфейс пользователя, о произошедшем страховом случае.
- Расследование - подпроцесс, при котором страховой агент проводит расследование страхового случая (собирает все данные о нем и заполняет дополняет заявку клиента). Также страховой агент уведомляет оператора о необходимости вынести вердикт по страховой выплате.
- Принятие решения - подпроцесс, при котором оператор, руководствуясь данными о здоровье клиента в базе данных и данными страхового агента после расследования, выносит вердикт о страховой выплате. После этого он переводит деньги клиенту или уведомляет его об отклонении заявки.

1.6 Обновление состояния здоровья

Участники

- Клиент
- Оператор

Этапы

- Сообщение об изменениях - подпроцесс, при котором клиент сообщает оператору об изменениях в состоянии своего здоровья.
- Обновление данных - подпроцесс, при котором оператор обновляет хранящиеся данные клиента.

1.7 Варианты использования

1.7.1 Клиент

Подача заявки на получение полиса

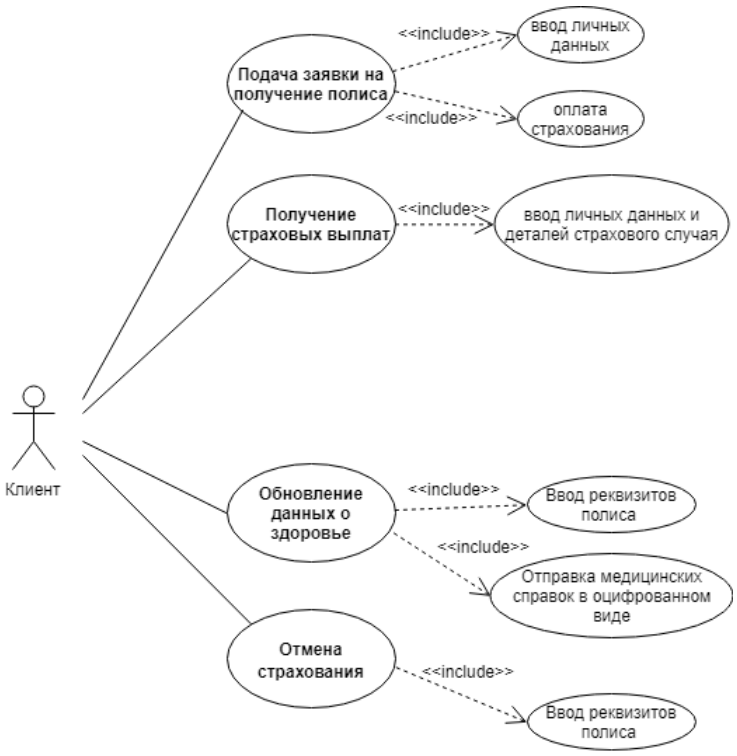
- Клиент отправляет заявку в систему на получение полиса.
- Клиент предоставляет личные данные и выбирает длительность страхования.
- Оператор передает данные о клиенте страховому агенту для прохождения обследования.
- Страховой агент передает результаты обследования оператору.
- Оператор одобряет заявку и высылает клиенту стоимость полиса.
- Клиент оплачивает стоимость полиса.
- Система подтверждает оплату полиса.
- Система создаеьт для клиента страховой полис в его личном кабинете.

Альтернатива 1 - Оплата не прошла. Система уведомляет об этом клиента. Для получения полиса клиент должен пополнить кошелек и оплатить заново.

Альтернатива 2 - У клиента выявлены значительные отклонения в здоровье. Оператор уведомляет клиента, что он не может получить полис.

Получение страховых выплат

- Клиент отправляет заявку в систему на получение страховых выплат.
- Оператор переводит ее на агента.



- Страховой агент проводит расследование.
- Страховой агент заносит в базу результаты.
- Оператор выносит вердикт по страховой выплате.
- Оператор осуществляет перевод клиенту.

Альтернатива 1 - Оператор отказывает в выплатах. Оператор делает вывод, что произошедший случай с клиентом не страховой. Он уведомляет об этом клиента и не переводит страховые выплаты.

Альтернатива 2 - Оплата не прошла. Система уведомляет об этом оператора. Для завершения оператор должен пополнить счет и оплатить заново.

Обновление данных о здоровье

- Клиент отправляет заявку в систему на обновление данных о здоровье.
- Клиент прикрепляет оцифрованную справку.
- Оператор одобряет заявку.
- Оператор обновляет данные о здоровье клиента.

Альтернатива 1 - Некорректные данные. Клиент предоставляет некорректные данные. Система отправляет уведомление, указывающее на некорректные данные. Клиент должен отправить заявку с корректными данными.

Подача заявки на продление договора с брокером

- Клиент отправляет заявку в систему на продление договора с брокером за 7 дней до конца срока действия текущего договора
- Система подтверждает заявку и высылает уведомление клиенту о продлении договора на заданный период

Альтернатива 1 - Клиент предоставил неправильные данные подтверждающие его изменения в здоровье. Оператор уведомляет об этом клиента.

Отмена страхования

- Клиент отправляет заявку в систему на отмену страхованию.
- Система обрабатывает заявку и прекращает действия полиса клиента.

1.7.2 Оператор

Обработка заявок от клиентов

- Оператор в личном кабинете выбирает необработанные заявки.
- Оператор одобряет и выполняет действия в заявке.

Альтернатива 1 - Отказ заявки. Оператор не одобряет заявку и не выполняет её. Заявка будет отклонена, клиент будет уведомлен

1.7.3 Страховой агент

Первоначальное обследование клиента

- Страховой агент получает уведомления от оператора с данными клиента.
- Страховой агент отправляет клиента на обследование в определенное медицинское учреждение.
- Страховой агент получает данные о здоровье клиента с внешней системы.
- Страховой агент заполняет данные в систему.

Размещение средств в хранилище

- Администратор получает заявку клиента от брокера с конкретной суммой
- Администратор сохраняет её в базе данных
- Администратор размещает средства в хранилище

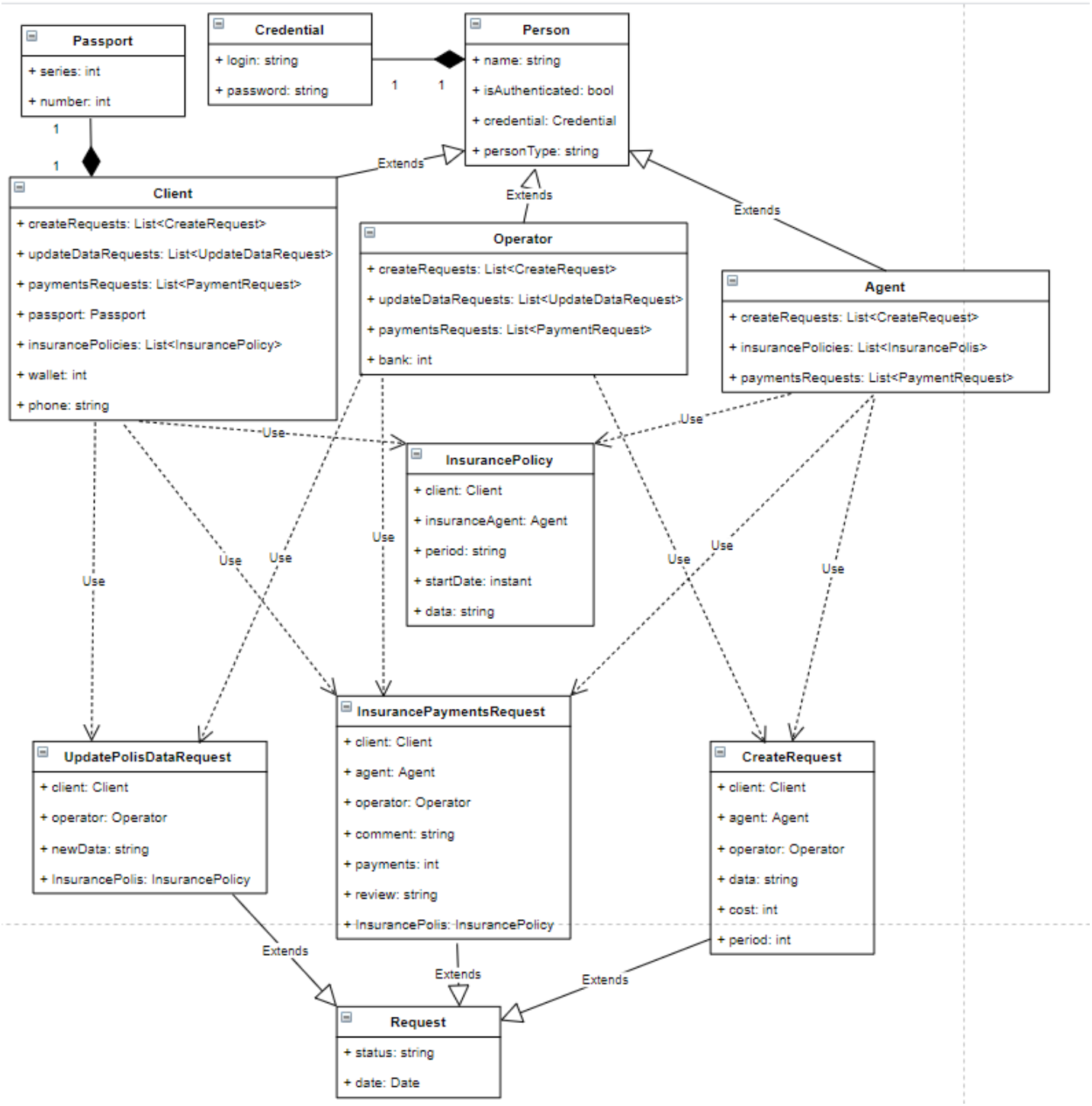
Расследование

- Страховой агент получает уведомления от оператора с данными клиента.
- Страховой агент проводит расследование.
- Страховой агент передает данные оператору.

2 Реализация

2.1 Объектно-ориентированное проектирование с учётом особенностей технологии

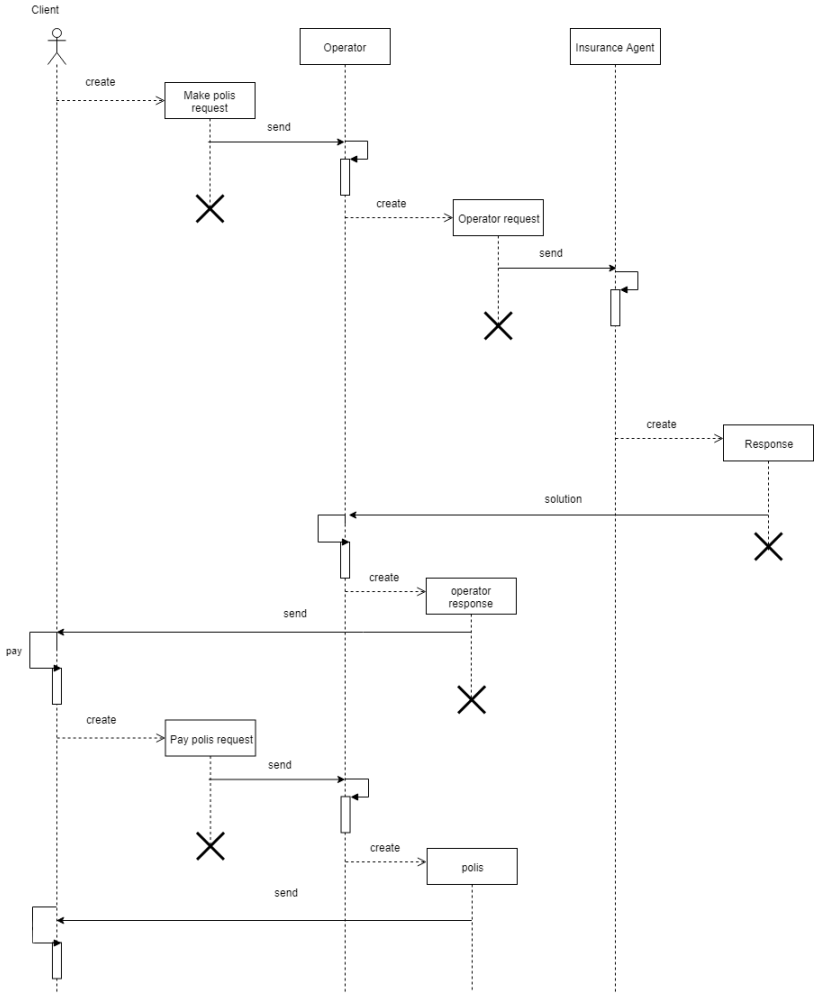
2.1.1 Статическая модель предметной области



На данной диаграмме видно, что есть общая сущность Person от которой наследуются 3 основных актора системы: клиент, страховой агент и оператор. Клиент взаимосвязан со следующими сущностями: паспорт, полис страхования, различные заявки. Запросы являются связующими звеньями со всеми участниками, поэтому прикреплены ко всем, они содержат в себе описание бизнес процесса.

2.1.2 Динамическая модель предметной области

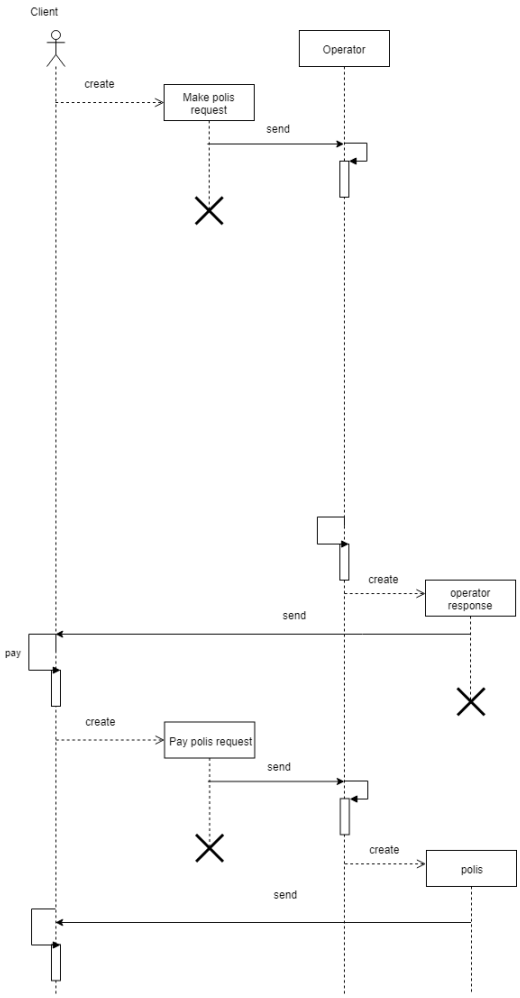
Общая схема



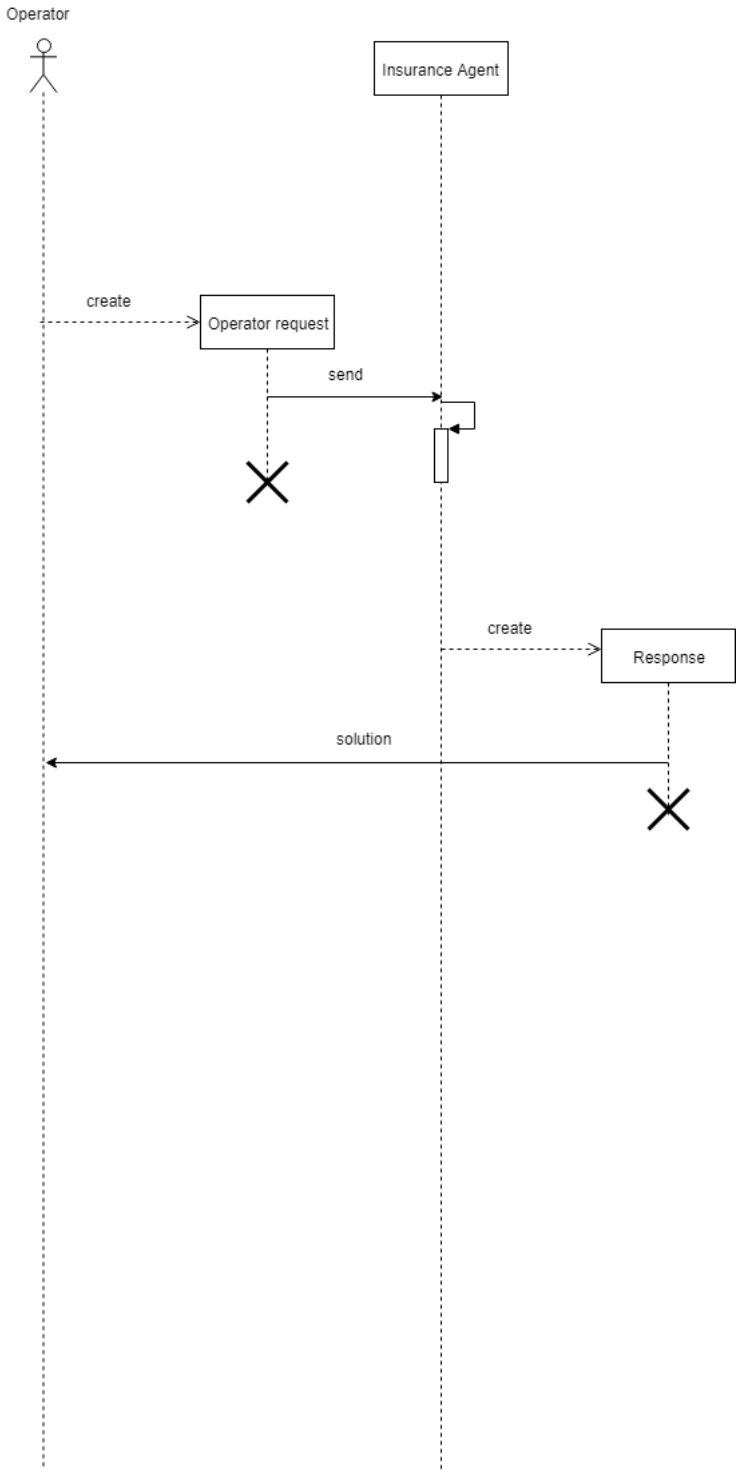
На данной диаграмме можно наблюдать процесс создания заявки на оформеления полиса. Клиент создает запрос на создание и отправляет его оператору. Оператор обрабатывает данный запрос, и после выбирает агента и отправляет запрос ему, обновляя статус запроса ("processed"). Агент принимает запрос от оператора и заполняет информация о клиенте. Далее он переводит заявку обратно на оператора. Он в свою очередь одобряет его и назначает цену за полис или отклоняет заявку в зависимости от данных, полученных от агента. Если заявка одобрена, то клиент видит это и может оплатить полис. После оплаты система создает для него полис.

Ниже на трёх диаграммах представлены отдельные части общей диаграммы, непосредственно отображающие связи между клиентом - оператором и оператора - агента.

Клиент-Оператор


Client
Operator
create
Make polis request
send
create
operator response
send
pay
create
Pay polis request
send
create
polis
send

Оператор-Агент



3 Описание программы

3.1 Backend

Для реализации бекенда, поставленной задачи был использован Spring Framework. Прежде всего были описаны модели, которые приводились выше. Отличительной чертой в реализации следует отметить наследование от базового класса AbstractEntity, который содержит в себе только уникальный идентификатор. Все модели расширяют данный класс, следовательно у каждой модели есть ID, по которому можно осуществлять поиск в базе данных.

Для взаимодействия с базой данных был написан слой репозиториев. Каждый репозиторий расширяет базовый интерфейс CommonRepository, который в свою очередь расширяет CrudRepository. Это сделано для того, что для каждого репозитория были доступны CRUD-методы.

Над слоем репозиториев был написан слой сервисов. При реализации данного слоя также был выполнен базовый сервис, от которого отнаследованы все остальные сервисы. В сервисах написана вся бизнес-логика данной системы. В каждом сервисе есть методы, которые описывают действия, которые могут быть выполнены конкретной сущностью.

Над слоем сервисов был написан слой REST-контролеров. В данном слое, также была вынесена асбтракция базового контролера, для возможности использования CRUD-методов. В каждом контролере присутствуют методы, которые являются endpoint приложения. В реализации данных методов вызываются соответствующие методы из слоя сервисов.

Все константы, используемые в реализации данной системы были вынесены в специальный файл. Также была настроена swagger-конфигурация для удобного отображения и взаимодействия с endpoint-ами системы. Кроме того, была настроена CORS-конфигурация, - то есть указано, кто может посылать запросы системы извне.

3.2 Frontend

Для реализации фронтенда, поставленной задачи был использован Angular. Было создано приложение, состоящая из компонентов. Каждый компонент представляет из себя совокупность файлов верстки, стилей и бизнес-логики, написанной на языке Typescript. Каждый компонент является отдельным экраном и инкапсулирует в себе соответствующую логику. Также в данном приложение были написаны сервисы, которые посылают запросы и принимают ответы с соответсвующих endpoint'ов бекенда. Константы также были вынесены в отдельный файл. Данное приложение взаимодействует с приложением бекенда.

4 Методика и результаты тестирования

4.1 Варианты использования

Методика тестирования представляла собой перечисление всех возможных вариантов использования системы для всех действующих лиц. Ниже представлены и описаны варианты использования, а также последовательности действий для их выполнения:

4.1.1 Система

- регистрация в системе - необходимо ввести следующую информацию
  - имя действующего лица
  - логин для входа в систему
  - пароль для входа в систему
  - тип действующего лица (клиент, оператор, страховой агент)
- вход в систему - необходимо ввести следующую информацю
  - логин для входа в систему
  - пароль для входа в систему

4.1.2 Клиент

- просмотр личной информации
- просмотр страхового полиса
  - создание полиса
    - выбор длительности действия
  - обновление данных
  - получение выплат
  - закрытие полиса
- просмотр существующих запросов
  - оплата полиса
- просмотр завершившихся запросов

4.1.3 Оператор

- просмотр личной информации
- просмотр запросов, которые не поступили в обработку
  - одобрение или отклонение запросов на обноление данных
  - перевод заявок на выплату и оформление полиса на агента.
- просмотр запросов в обработке, привязанных к оператору
  - одобрение или отклонение запросов на выплату или оформление полиса
  - назначение цены за полис
  - выплата денег клиенту по страховому случаю
- просмотр завершившихся запросов

4.1.4 Страховоой агент

- просмотр личной информации
- просмотр информации о закрепленных клиентах
- просмотр информации о закрепленных заявках
- заполнение информации о здоровье клиента после осмотра
- назначение страховых выплат

4.2 Ручное тестирование

Было проведено ручное тестирование двух частей системы: бекенда и фронтенда.

4.2.1 Backend

В качестве ручного тестирования со стороны бекенда были выполнены все возможные запросы с REST-клиента. В данных запросах были заданы соответствующие endpoint'ы и заполнены необходимые параметры. В результате были получены ожидаемые ответы, что показывает верную работу севрверной части системы. Для данного вида тестирования был использован REST-клиент Postman.

4.2.2 Frontend

В качестве ручного тестирования со стороны фронтенда были выполнены все возможные сценарии для каждого пользователя, используя пользовательский интерфейс. Все сценарии были успешно завершены, тем самым подтверждая корректную работу всей системы.

5 Инструкция системному администратору по развёртыванию приложения

Для развертывания данной системы необходимо наличие любой операционной системы, например Windows/Linux/MacOS. Далее перечислены все средства, требуемые к установки для развёртывания системы локально на машине:

- Gradle - обычная установка, согласно прилагающейся инструкции
- Postgres - обычная установка, согласно прилагающейся инструкции, версия 9.5 и выше
- Java - обычная установка, согласно прилагающейся инструкции, версия 1.8 и выше
- Node.js - обычная установка, согласно прилагающейся инструкции, версия 12.13.0 и выше
- любой браузер, например Google Chrome или Mozilla

После установки всех средств, представленных выше, необходимо включить запустить службу (Windows) или процесс (Linux/MacOS) postgres и создать базу данных со следующим названием: **insurance**. После создания базы данных можно приступать к запуску системы.

6 Инструкция пользователю по запуску приложения

Для запуска системы необходимо последовательно выполнить следующие действия:

- в папке core выполнить команду ./gradlew bootRun
- проект настроен таким образом, что в базе данных автоматически создадутся все необходимые таблицы
- открыть папку ui
- выполнить команду **npm install**
- выполнить команду **npm build**
- выполнить команду **npm serve**
- открыть, установленный браузер и ввести в адресную строку следующий адрес: **http://localhost:4200/**

7 Вывод

В ходе данной работы была разработана информационная система "Insurance Service"("Система страхованимя"), предназначенная для удобства выполнения различных операций по страхованию. В процессе разработки были изучены архитектурные шаблоны, шаблоны проектирования слоев программного обеспечения. Также были пройдены следующие этапы проектирования информационной системы: выявление функциональных требований, описание бизнес-процессов, разработка вариантов использования. В результате получены полезные знания в области проектирования архитектур программного обеспечения, которые очень пригодятся в работе над реальными проектами.

С точки зрения завершенности можно оценить систему, как готовую для использования. Данный вывод можно сделать исходя из успешного ручного тестирования всех возможных методов серверной части.

С точки зрения основных свойств распределенных систем, можно оценить системы следующим образом: система является открытой и готова к расширению, также систему можно назвать прозрачной. Систему также можно масштабировать различными способами, можно как реплицировать, так и шардировать. Особых тестов производительности не проводилось, но можно утверждать, что система точно сможет выдержать нагрузку в несколько десятков тысяч пользователей, так как ограничений для этого нет. С точки зрения удоства использования, система представлена довольно удобной и интуитивно понятной, поэтому система будет понятна сразу, даже если вы ей ещё не пользовались.

8 Приложение - листинги

Ниже представлены некоторые фрагменты кода, весь код можно посмотреть по ссылке: **https://github.com/Lorismelik/insurance-service**

Листинг 1: ClientService

```
package com.kspt.insurance.services.actors;

import com.kspt.insurance.configuration.Constants;
import com.kspt.insurance.models.actors.InsuranceAgent;
import com.kspt.insurance.models.requests.ClosePolisRequest;
import com.kspt.insurance.models.requests.CreatePolisRequest;
import com.kspt.insurance.models.requests.InsurancePaymentsRequest;
import com.kspt.insurance.models.requests.UpdatePolisDataRequest;
import com.kspt.insurance.models.actors.Client;
import com.kspt.insurance.models.system.*;
import com.kspt.insurance.repositories.actors.ClientRepository;
import com.kspt.insurance.repositories.actors.OperatorRepository;
import com.kspt.insurance.repositories.requests.ClosePolisRequestRepository;
import com.kspt.insurance.repositories.requests.CreatePolisRequestRepository;
import com.kspt.insurance.repositories.requests.InsurancePaymentsRequestRepository;
import com.kspt.insurance.repositories.requests.UpdatePolisDataRequestRepository;
import com.kspt.insurance.repositories.system.InsurancePolisRepository;
import com.kspt.insurance.repositories.system.PassportRepository;
import com.kspt.insurance.services.CrudService;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.Instant;
import java.util.*;

@Service
public class ClientService extends CrudService<Client, ClientRepository> {

    private final PassportRepository passportRepository;
    private final CreatePolisRequestRepository createPolisRequestRepository;
    private final OperatorRepository operatorRepository;
    private final InsurancePolisRepository insurancePolisRepository;
    private final UpdatePolisDataRequestRepository updatePolisDataRequestRepository;
    private final InsurancePaymentsRequestRepository insurancePaymentsRequestRepository;
    private final ClosePolisRequestRepository closePolisRequestRepository;

    public ClientService(@NotNull final ClientRepository clientRepository,
                         @NotNull final PassportRepository passportRepository,
                         @NotNull final CreatePolisRequestRepository createPolisRequestRepository,
                         @NotNull final OperatorRepository operatorRepository,
                         @NotNull final InsurancePolisRepository insurancePolisRepository,
                         @NotNull final UpdatePolisDataRequestRepository updatePolisDataRequestRepository
                             ,
                         @NotNull final InsurancePaymentsRequestRepository
                             insurancePaymentsRequestRepository,
                         @NotNull final ClosePolisRequestRepository closePolisRequestRepository) {
        super(clientRepository);
        this.updatePolisDataRequestRepository = updatePolisDataRequestRepository;
        this.operatorRepository = operatorRepository;
        this.passportRepository = passportRepository;
        this.createPolisRequestRepository = createPolisRequestRepository;
        this.insurancePolisRepository = insurancePolisRepository;
        this.insurancePaymentsRequestRepository = insurancePaymentsRequestRepository;
        this.closePolisRequestRepository = closePolisRequestRepository;
    }

    @Override
    public Client update(@NotNull final Long id,
                         @NotNull final Client client) {
        if (repository.existsById(id)) {
            if (client.getIsAuthenticated()) {
                client.setId(id);
                return repository.save(client);
            }
        }
        return null;
    }

    public List<CreatePolisRequest> getCreatePolisRequestById(@NotNull final Long clientId) {
        return  createPolisRequestRepository.findByClientId(clientId);
    }

    public List<InsurancePaymentsRequest> getInsurancePaymentsRequestById(@NotNull final Long clientId) {
        return  insurancePaymentsRequestRepository.findByClientId(clientId);
    }

    public List<UpdatePolisDataRequest> getUpdatePolisDataRequestById(@NotNull final Long clientId) {
        return  updatePolisDataRequestRepository.findByClientId(clientId);
    }

    public List<ClosePolisRequest> getClosePolisRequestById(@NotNull final Long clientId) {
        return  closePolisRequestRepository.findByClientId(clientId);
```

```
82      }
83
84      public List<InsurancePolis> getPolis(@NotNull final Long clientId) {
85          return insurancePolisRepository.findByClientId(clientId);
86      }
87
88      @Transactional
89      public Client setPassport(@NotNull final Long clientId,
90                                @NotNull final Map<String, String> data) {
91          Client client = repository.findById(clientId).orElse(null);
92          if (client == null || !client.getIsAuthenticated()) return null;
93          final int series = Integer.parseInt(data.get("series"));
94          final int number = Integer.parseInt(data.get("number"));
95          final Passport passport = passportRepository.save(new Passport(series, number));
96          client.setPassport(passport);
97          return repository.save(client);
98      }
99
100     public Client setPhone(@NotNull final Long clientId,
101                            @NotNull final Map<String, String> data) {
102         Client client = repository.findById(clientId).orElse(null);
103         if (client == null || !client.getIsAuthenticated()) return null;
104         final String phone = data.get("phone");
105         client.setPhone(phone);
106         return repository.save(client);
107     }
108
109     @Transactional
110     public CreatePolisRequest createRequestForPolis(@NotNull final Long clientId,
111                                                      @NotNull final Map<String, String> data) {
112         Optional<Client> optionalClient = repository.findById(clientId);
113         if (!optionalClient.isPresent()) return null;
114         Client client = optionalClient.get();
115         final String period = data.get("period");
116         CreatePolisRequest request = new CreatePolisRequest(clientId, period);
117         CreatePolisRequest savedRequest = createPolisRequestRepository.save(request);
118         List<CreatePolisRequest> createPolisRequests = client.getCreatePolisRequests();
119         createPolisRequests.add(savedRequest);
120         client.setCreatePolisRequests(createPolisRequests);
121         repository.save(client);
122         return savedRequest;
123     }
124
125     @Transactional
126     public UpdatePolisDataRequest createRequestForUpdatePolis(@NotNull final Long clientId,
127                                                               @NotNull final Map<String, String> data) {
128         Client client = repository.findById(clientId).orElse(null);
129         if (client == null || !client.getIsAuthenticated()) return null;
130         final String newData = data.get("newData");
131         final Long polisId = Long.parseLong(data.get("polisId"));
132         UpdatePolisDataRequest request = new UpdatePolisDataRequest(polisId, null, clientId, newData,
                Constants.UpdatePolisDataStatus.CREATED);
133         UpdatePolisDataRequest savedRequest = updatePolisDataRequestRepository.save(request);
134         List<UpdatePolisDataRequest> updatePolisDataRequest = client.getUpdatePolisDataRequests();
135         updatePolisDataRequest.add(savedRequest);
136         client.setUpdatePolisDataRequests(updatePolisDataRequest);
137         repository.save(client);
138         return savedRequest;
139     }
140
141     @Transactional
142     public InsurancePaymentsRequest createRequestForInsurancePayments(@NotNull final Long clientId,
143                                                                        @NotNull final Map<String, String>
                                                                               data) {
144         Client client = repository.findById(clientId).orElse(null);
145         if (client == null || !client.getIsAuthenticated()) return null;
146         final String additionalData = data.get("additionalData");
147         final Long polisId = Long.parseLong(data.get("polisId"));
148         InsurancePolis polis = insurancePolisRepository.findById(polisId).orElse(null);
149         if (polis == null) return null;
150         InsurancePaymentsRequest request = new InsurancePaymentsRequest(clientId, additionalData, polisId
                , polis.getInsuranceAgentId());
151         InsurancePaymentsRequest savedRequest = insurancePaymentsRequestRepository.save(request);
152         List<InsurancePaymentsRequest> insurancePaymentsRequests = client.getInsurancePaymentsRequest();
153         insurancePaymentsRequests.add(savedRequest);
154         client.setInsurancePaymentsRequest(insurancePaymentsRequests);
155         repository.save(client);
156         return savedRequest;
157     }
158
159     @Transactional
160     public InsurancePolis payForPolis(@NotNull final Long clientId,
161                                       @NotNull final Map<String, String> data) {
162         Client client = repository.findById(clientId).orElse(null);
163         if (client == null || !client.getIsAuthenticated()) return null;
164         final Long requestId = Long.parseLong(data.get("requestId"));
165         Optional<CreatePolisRequest> optionalRequest = createPolisRequestRepository.findById(requestId);
166         if (!optionalRequest.isPresent()) return null;
167         CreatePolisRequest createPolisRequest = optionalRequest.get();
168         if (client.getWallet() >= createPolisRequest.getCost() && !createPolisRequest.getStatus().equals(
                Constants.CreatePolisStatus.DECLINED)) {
```

```
            client.setWallet(client.getWallet() - createPolisRequest.getCost());
            operatorRepository.findById(createPolisRequest.getOperatorId()).ifPresent(
                    x -> {x.setBank(x.getBank() + createPolisRequest.getCost()); operatorRepository.save(
                        x);}
            );
            createPolisRequest.setStatus(Constants.CreatePolisStatus.COMPLETED);
            createPolisRequestRepository.save(createPolisRequest);
            InsurancePolis polis = new InsurancePolis(clientId,
                    createPolisRequest.getInsuranceAgentId(),
                    createPolisRequest.getPeriod(),
                    createPolisRequest.getStartdate(),
                    createPolisRequest.getData(),
                    Constants.PolisStatus.OPENED
                    );
            return insurancePolisRepository.save(polis);

        }
        return null;
    }

    public Client updateWallet(@NotNull final Long clientId,
                               @NotNull final Integer money) {
        Client client = repository.findById(clientId).orElse(null);
        if (client == null || !client.getIsAuthenticated()) return null;
        client.setWallet(client.getWallet() + money);
        return repository.save(client);
    }

    @Transactional
    public ClosePolisRequest closePolis(@NotNull final Long clientId,
                                        @NotNull final Long polisId) {
        Client client = repository.findById(clientId).orElse(null);
        if (client == null || !client.getIsAuthenticated()) return null;
        Optional<InsurancePolis> optionalInsurancePolis = insurancePolisRepository.findById(polisId);
        if (!optionalInsurancePolis.isPresent()) return null;
        InsurancePolis insurancePolis = optionalInsurancePolis.get();
        insurancePolis.setStatus(Constants.PolisStatus.CLOSED);
        ClosePolisRequest request = new ClosePolisRequest(polisId, clientId, Instant.now());
        insurancePolisRepository.save(insurancePolis);
        return closePolisRequestRepository.save(request);

    }
}
```

Листинг 2: CommonController

```
package com.kspt.insurance.controllers;

import com.kspt.insurance.models.AbstractEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

public interface CommonController<T extends AbstractEntity> {

    @GetMapping("count")
    long count();

    @PostMapping("create")
    T create(@RequestBody T entity);

    @DeleteMapping("deleteAll")
    void deleteAll();

    @DeleteMapping("delete/{id}")
    void deleteById(@PathVariable Long id);

    @GetMapping("exist/{id}")
    boolean existById(@PathVariable Long id);

    @GetMapping("getAll")
    List<T> getAll();

    @GetMapping("get/{id}")
    Optional<T> getById(@PathVariable Long id);

    @PutMapping("update/{id}")
    T update(@PathVariable Long id, @RequestBody T entity);
}
```

Листинг 3: create.request.popup

```
import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
import {Router} from '@angular/router';
import {OperatorService} from '../../services/operator.service';
import {StoreService} from '../../services/store.service';
import {CreateRequest} from '../../models/requests/CreateRequest';
import {ClientService} from '../../services/client.service';
```

```
import {AgentService} from '../../services/agent.service';
import {RoleEnumModel} from '../../models';


@Component({
    selector: 'create-request-popup',
    templateUrl: './create.request.popup.html'
})
export class CreateRequestPopup implements OnInit {
    @Input() private request: CreateRequest;
    @Input() private role: RoleEnumModel;
    @Output() public onCloseEvent = new EventEmitter<any>();
    private operator: string;
    private client: string;
    private agent: string;
    private agents: any[] = [];
    private date: string;
    private selectedAgent: any;
    private agentData: string;
    private cost: number;

    constructor(private router: Router,
                private storeService: StoreService,
                private clientService: ClientService,
                private operatorService: OperatorService,
                private agentService: AgentService) {}

    ngOnInit() {
        this.request.operatorId ? this.operatorService.getById(this.request.operatorId).subscribe(data =>
             this.operator = data.name, error => alert("Error is occured")) : this.operator = 'Not
            stated';
        this.request.clientId && this.clientService.getById(this.request.clientId).subscribe(data => this
            .client = data.name, error => alert("Error is occured"));
        this.request.insuranceAgentId ? this.agentService.getById(this.request.insuranceAgentId).
            subscribe(data => this.agent = data.name, error => alert("Error is occured")) : this.agent =
             'Not stated';
        this.agentService.getAll().subscribe(data => {
            data.forEach(x => this.agents.push({name: x.name, id: x.id}));
        });
        this.date = new Date(this.request.startdate).toDateString();
    }

    approve() {
        this.operatorService.approveCreateRequest(this.storeService.getId(), this.request.id, this.cost,
            true)
            .subscribe(x => this.onCloseEvent.emit(), error => alert("Error is occured"));
    }

    decline() {
        this.operatorService.approveCreateRequest(this.storeService.getId(), this.request.id, this.cost,
            false)
            .subscribe(x => this.onCloseEvent.emit(), error => alert("Error is occured"));
    }

    close() {
            this.onCloseEvent.emit();
    }

    save() {
        if (this.storeService.getRole() === RoleEnumModel.Operator) {
            if (this.selectedAgent && this.storeService.getRole() === RoleEnumModel.Operator && this.
                request.status === 'created') {
                this.operatorService.delegateCreateRequest(this.storeService.getId(), this.selectedAgent,
                     this.request.id).subscribe(x => this.onCloseEvent.emit(), error => alert("Error is
                    occured"));
            }
        }

        if (this.storeService.getRole() === RoleEnumModel.InsuranceAgent) {
            if (this.request.status === 'processed') {
                this.agentService.processCreateRequest(this.storeService.getId(), this.agentData, this.
                    request.id).subscribe(x => this.onCloseEvent.emit(), error => alert("Error is
                    occured"));
            }
        }
    }

    pay() {
        this.clientService.getById(this.request.clientId).subscribe(x => {
            if (x.wallet >= this.request.cost) {
                this.clientService.payForPolis(this.storeService.getId(), this.request.id).subscribe(res
                    => this.onCloseEvent.emit(), e => alert("Error is occured"));
            } else {
                alert("You havent enough money!")
            }
        }, error => alert("Error is occured"));
    }
}
```

Листинг 4: InsuranceAgentService

```java
package com.kspt.insurance.services.actors;

import com.kspt.insurance.configuration.Constants;
import com.kspt.insurance.models.actors.Client;
import com.kspt.insurance.models.requests.CreatePolisRequest;
import com.kspt.insurance.models.actors.InsuranceAgent;
import com.kspt.insurance.models.requests.InsurancePaymentsRequest;
import com.kspt.insurance.repositories.actors.ClientRepository;
import com.kspt.insurance.repositories.actors.InsuranceAgentRepository;
import com.kspt.insurance.repositories.requests.CreatePolisRequestRepository;
import com.kspt.insurance.repositories.requests.InsurancePaymentsRequestRepository;
import com.kspt.insurance.services.CrudService;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.stream.Collectors;

@Service
public class InsuranceAgentService extends CrudService<InsuranceAgent, InsuranceAgentRepository> {

    private final ClientRepository clientRepository;
    private final CreatePolisRequestRepository createPolisRequestRepository;
    private final InsurancePaymentsRequestRepository insurancePaymentsRequestRepository;

    public InsuranceAgentService(@NotNull final InsuranceAgentRepository repository,
                                 @NotNull final CreatePolisRequestRepository createPolisRequestRepository,
                                 @NotNull final ClientRepository clientRepository,
                                 @NotNull final InsurancePaymentsRequestRepository insurancePaymentsRequestRepository) {
        super(repository);
        this.createPolisRequestRepository = createPolisRequestRepository;
        this.clientRepository = clientRepository;
        this.insurancePaymentsRequestRepository = insurancePaymentsRequestRepository;
    }

    public List<Client> getClientsById(@NotNull final Long agentId) {
        Set<Long> clientsIds = new HashSet<>();
        createPolisRequestRepository.findAll().stream().filter(x -> {
            if (x.getInsuranceAgentId() != null) return x.getInsuranceAgentId().equals(agentId); else return false;
        }).forEach(x -> clientsIds.add(x.getClientId()));
        insurancePaymentsRequestRepository.findAll().stream().filter(x -> {
            if (x.getInsuranceAgentId() != null) return x.getInsuranceAgentId().equals(agentId); else return false;
        }).forEach(x -> clientsIds.add(x.getClientId()));
        return (List<Client>) clientRepository.findAllById(clientsIds);

    }

    public List<CreatePolisRequest> getCreatePolisRequestById(@NotNull final Long agentId) {
        return  createPolisRequestRepository.findByInsuranceAgentId(agentId);
    }

    public List<InsurancePaymentsRequest> getInsurancePaymentsRequestById(@NotNull final Long agentId) {
        return  insurancePaymentsRequestRepository.findByInsuranceAgentId(agentId);
    }

    public CreatePolisRequest processCreatePolisRequest(@NotNull final Long agentId,
                                                        @NotNull final Map<String, String> data) {
        InsuranceAgent agent = repository.findById(agentId).orElse(null);
        if (agent == null || !agent.getIsAuthenticated()) return null;
        String polisData = data.get("data");
        Long requestId = Long.parseLong(data.get("requestId"));
        Optional<CreatePolisRequest> optionalCreatePolisRequest = createPolisRequestRepository.findById(requestId);
        if (!optionalCreatePolisRequest.isPresent()) return null;
        CreatePolisRequest createPolisRequest = optionalCreatePolisRequest.get();
        createPolisRequest.setData(polisData);
        createPolisRequest.setStatus(Constants.CreatePolisStatus.WAITING_FOR_APPROVE);
        return createPolisRequestRepository.save(createPolisRequest);
    }

    public InsurancePaymentsRequest processInsurancePaymentsRequest(@NotNull final Long agentId,
                                                                    @NotNull final Map<String, String> data) {
        InsuranceAgent agent = repository.findById(agentId).orElse(null);
        if (agent == null || !agent.getIsAuthenticated()) return null;
        Long requestId = Long.parseLong(data.get("requestId"));
        Optional<InsurancePaymentsRequest> optionalInsurancePaymentsRequest = insurancePaymentsRequestRepository.findById(requestId);
        if (!optionalInsurancePaymentsRequest.isPresent()) return null;
        InsurancePaymentsRequest insurancePaymentsRequest = optionalInsurancePaymentsRequest.get();
        String review = data.get("review");
        Integer payments = Integer.parseInt(data.get("payments"));
        insurancePaymentsRequest.setReview(review);
        insurancePaymentsRequest.setPayments(payments);
        insurancePaymentsRequest.setStatus(Constants.InsurancePaymentsStatus.WAITING_FOR_APPROVE);
        return insurancePaymentsRequestRepository.save(insurancePaymentsRequest);
```

```
    }
}
```

Листинг 5: SystemService

```
package com.kspt.insurance.services;

import com.kspt.insurance.configuration.Constants;
import com.kspt.insurance.models.actors.Operator;
import com.kspt.insurance.models.actors.InsuranceAgent;
import com.kspt.insurance.models.actors.Client;
import com.kspt.insurance.models.actors.Person;
import com.kspt.insurance.models.system.Credentials;
import com.kspt.insurance.repositories.actors.InsuranceAgentRepository;
import com.kspt.insurance.repositories.actors.ClientRepository;
import com.kspt.insurance.repositories.actors.OperatorRepository;
import com.kspt.insurance.repositories.system.CredentialsRepository;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;

@Service
public class SystemService {

    private final OperatorRepository operatorRepository;
    private final ClientRepository clientRepository;
    private final InsuranceAgentRepository insuranceAgentRepository;
    private final CredentialsRepository credentialsRepository;

    public SystemService(@NotNull final OperatorRepository operatorRepository,
                         @NotNull final ClientRepository clientRepository,
                         @NotNull final InsuranceAgentRepository insuranceAgentRepository,
                         @NotNull final CredentialsRepository credentialsRepository) {
        this.operatorRepository = operatorRepository;
        this.clientRepository = clientRepository;
        this.insuranceAgentRepository = insuranceAgentRepository;
        this.credentialsRepository = credentialsRepository;
    }

    public Person signUp(@NotNull final Map<String, Object> data) {
        final String login = data.get("login").toString();
        final String password = data.get("password").toString();
        final Credentials credentials = new Credentials(login, password);

        final String personType = data.get("personType").toString();
        credentials.setType(personType);
        final String name = data.get("name").toString();

        switch (personType) {
            case Constants.Actor.CLIENT: {
                Client client = new Client(name, personType);
                client.setCredentials(credentials);
                return clientRepository.save(client);
            }
            case Constants.Actor.INSURANCE_AGENT: {
                InsuranceAgent insuranceAgent = new InsuranceAgent(name, personType);
                insuranceAgent.setCredentials(credentials);
                return insuranceAgentRepository.save(insuranceAgent);
                        }
            case Constants.Actor.OPERATOR: {
                Operator operator = new Operator(name, personType);
                operator.setBank(1000);
                operator.setCredentials(credentials);
                return operatorRepository.save(operator);
            }
            default:
                return null;
        }
    }

    public Person signIn(@NotNull final Map<String, Object> data) {
        final String login = data.get("login").toString();
        final String password = data.get("password").toString();
        final Credentials credentialsWithoutId = new Credentials(login, password);
        final Credentials credentials = credentialsRepository.findByLoginAndPassword(
                credentialsWithoutId.getLogin(),
                credentialsWithoutId.getPassword()).orElse(null);
        if (credentials != null) {
            final String personType = credentials.getType();
            try {
                switch (personType) {
                    case Constants.Actor.CLIENT: {
                        Client client = clientRepository.findByCredentials(credentials).orElse(null);
                        if (client != null) {
                            client.setIsAuthenticated(true);
                            return clientRepository.save(client);
                        }
                        break;
```

```
                    }
                    case Constants.Actor.INSURANCE_AGENT: {
                        InsuranceAgent insuranceAgent = insuranceAgentRepository.findByCredentials(
                            credentials).orElse(null);
                        if (insuranceAgent != null) {
                            insuranceAgent.setIsAuthenticated(true);
                            return insuranceAgentRepository.save(insuranceAgent);
                        }
                        break;
                    }
                    case Constants.Actor.OPERATOR: {
                        Operator operator = operatorRepository.findByCredentials(credentials).orElse(null
                            );
                        if (operator != null) {
                            operator.setIsAuthenticated(true);
                            return operatorRepository.save(operator);
                        }
                        break;
                    }
                }
            } catch (Exception ignored) {
            }
        }
        return null;
    }

    public boolean signOut(@NotNull final Map<String, Object> data) {
        final Long id = Long.parseLong(data.get("id").toString());
        final String personType = data.get("personType").toString();
        switch (personType) {
            case Constants.Actor.CLIENT: {
                Client client = clientRepository.findById(id).orElse(null);
                if (client != null) {
                    client.setIsAuthenticated(false);
                    clientRepository.save(client);
                    return true;
                }
                break;
            }
            case Constants.Actor.INSURANCE_AGENT: {
                InsuranceAgent insuranceAgent = insuranceAgentRepository.findById(id).orElse(null);
                if (insuranceAgent != null) {
                    insuranceAgent.setIsAuthenticated(false);
                    insuranceAgentRepository.save(insuranceAgent);
                    return true;
                }
                break;
            }
            case Constants.Actor.OPERATOR: {
                Operator operator = operatorRepository.findById(id).orElse(null);
                if (operator != null) {
                    operator.setIsAuthenticated(false);
                    operatorRepository.save(operator);
                    return true;
                }
                break;
            }
        }
        return false;
    }
}
```

Листинг 6: OperatorService

```
package com.kspt.insurance.services.actors;

import com.kspt.insurance.configuration.Constants;
import com.kspt.insurance.models.actors.Client;
import com.kspt.insurance.models.requests.CreatePolisRequest;
import com.kspt.insurance.models.requests.InsurancePaymentsRequest;
import com.kspt.insurance.models.requests.UpdatePolisDataRequest;
import com.kspt.insurance.models.actors.InsuranceAgent;
import com.kspt.insurance.models.actors.Operator;
import com.kspt.insurance.repositories.actors.ClientRepository;
import com.kspt.insurance.repositories.actors.InsuranceAgentRepository;
import com.kspt.insurance.repositories.actors.OperatorRepository;
import com.kspt.insurance.repositories.requests.CreatePolisRequestRepository;
import com.kspt.insurance.repositories.requests.InsurancePaymentsRequestRepository;
import com.kspt.insurance.repositories.requests.UpdatePolisDataRequestRepository;
import com.kspt.insurance.repositories.system.InsurancePolisRepository;
import com.kspt.insurance.services.CrudService;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

```

```java
@Service
public class OperatorService extends CrudService<Operator, OperatorRepository> {

    private final ClientRepository clientRepository;
    private final CreatePolisRequestRepository createPolisRequestRepository;
    private final InsuranceAgentRepository insuranceAgentRepository;
    private final InsurancePolisRepository insurancePolisRepository;
    private final UpdatePolisDataRequestRepository updatePolisDataRequestRepository;
    private final InsurancePaymentsRequestRepository insurancePaymentsRequestRepository;

    public OperatorService(@NotNull OperatorRepository repository,
                           @NotNull final ClientRepository clientRepository,
                           @NotNull final InsuranceAgentRepository insuranceAgentRepository,
                           @NotNull final CreatePolisRequestRepository createPolisRequestRepository,
                           @NotNull final UpdatePolisDataRequestRepository
                               updatePolisDataRequestRepository,
                           @NotNull final InsurancePolisRepository insurancePolisRepository,
                           @NotNull final InsurancePaymentsRequestRepository
                               insurancePaymentsRequestRepository) {
        super(repository);
        this.insurancePolisRepository = insurancePolisRepository;
        this.updatePolisDataRequestRepository = updatePolisDataRequestRepository;
        this.insuranceAgentRepository = insuranceAgentRepository;
        this.createPolisRequestRepository = createPolisRequestRepository;
        this.clientRepository = clientRepository;
        this.insurancePaymentsRequestRepository = insurancePaymentsRequestRepository;
    }

    public List<CreatePolisRequest> getCreatePolisRequestById(@NotNull final Long operatorId) {
        return  createPolisRequestRepository.findByOperatorId(operatorId);
    }

    public List<InsurancePaymentsRequest> getInsurancePaymentsRequestById(@NotNull final Long operatorId)
         {
        return  insurancePaymentsRequestRepository.findByOperatorId(operatorId);
    }

    public List<UpdatePolisDataRequest> getUpdatePolisDataRequestById(@NotNull final Long operatorId) {
        return  updatePolisDataRequestRepository.findByOperatorId(operatorId);
    }

    @Transactional
    public CreatePolisRequest delegateCreateRequestToAgent(@NotNull final Long operatorId,
                                                           @NotNull final Map<String, String> data) {
        Long requestId = Long.parseLong(data.get("requestId"));
        Operator operator = repository.findById(operatorId).orElse(null);
        if (operator == null || !operator.getIsAuthenticated()) return null;
        Long agentId = Long.parseLong(data.get("agentId"));

        Optional<CreatePolisRequest> requestOptional = createPolisRequestRepository.findById(requestId);
        if (!requestOptional.isPresent()) return null;
        CreatePolisRequest request = requestOptional.get();
        request.setOperatorId(operatorId);
        request.setInsuranceAgentId(agentId);
        request.setStatus(Constants.CreatePolisStatus.PROCESSED);
        CreatePolisRequest savedRequest = createPolisRequestRepository.save(request);
        InsuranceAgent agent = insuranceAgentRepository.findById(agentId).get();
        List<CreatePolisRequest> operatorRequests = operator.getCreatePolisRequests();
        operatorRequests.add(savedRequest);
        List<CreatePolisRequest> agentRequests = agent.getCreatePolisRequests();
        agentRequests.add(savedRequest);
        operator.setCreatePolisRequests(operatorRequests);
        agent.setCreatePolisRequests(agentRequests);
        repository.save(operator);
        insuranceAgentRepository.save(agent);
        return savedRequest;
    }

    @Transactional
    public InsurancePaymentsRequest delegateGetInsurancePaymentsRequestToAgent(@NotNull final Long
         operatorId,
                                                                               @NotNull final Map<String,
                                                                                    String> data) {
        Long requestId = Long.parseLong(data.get("requestId"));
        Operator operator = repository.findById(operatorId).orElse(null);
        if (operator == null || !operator.getIsAuthenticated()) return null;
        Optional<InsurancePaymentsRequest> requestOptional = insurancePaymentsRequestRepository.findById(
             requestId);
        if (!requestOptional.isPresent()) return null;
        InsurancePaymentsRequest request = requestOptional.get();
        request.setOperatorId(operatorId);
        request.setStatus(Constants.InsurancePaymentsStatus.PROCESSED);
        InsurancePaymentsRequest savedRequest = insurancePaymentsRequestRepository.save(request);
        InsuranceAgent agent = insuranceAgentRepository.findById(request.getInsuranceAgentId()).get();
        List<InsurancePaymentsRequest> operatorRequests = operator.getInsurancePaymentsRequests();
        operatorRequests.add(savedRequest);
        List<InsurancePaymentsRequest> agentRequests = agent.getInsurancePaymentsRequests();
        agentRequests.add(savedRequest);
        operator.setInsurancePaymentsRequests(operatorRequests);
        agent.setInsurancePaymentsRequests(agentRequests);
        repository.save(operator);
```

```
112         insuranceAgentRepository.save(agent);
113         return savedRequest;
114     }
115
116     public List<CreatePolisRequest> getUnresolvedCreateRequests(Long id) {
117         Operator operator = repository.findById(id).orElse(null);
118         if (operator == null || !operator.getIsAuthenticated()) return null;
119         return createPolisRequestRepository.findAll()
120                 .stream()
121                 .filter(x -> x.getInsuranceAgentId() == null && x.getOperatorId() == null)
122                 .collect(Collectors.toList());
123     }
124
125     public List<CreatePolisRequest> getCreateRequestsByOperator(Long id) {
126         Operator operator = repository.findById(id).orElse(null);
127         if (operator == null || !operator.getIsAuthenticated()) return null;
128         return createPolisRequestRepository.findByOperatorId(id);
129     }
130
131     public List<UpdatePolisDataRequest> getUpdateRequestsByOperator(Long id) {
132         Operator operator = repository.findById(id).orElse(null);
133         if (operator == null || !operator.getIsAuthenticated()) return null;
134         return updatePolisDataRequestRepository.findByOperatorId(id);
135     }
136
137     public List<InsurancePaymentsRequest> getInsurancePaymentsRequestsByOperator(Long id) {
138         Operator operator = repository.findById(id).orElse(null);
139         if (operator == null || !operator.getIsAuthenticated()) return null;
140         return insurancePaymentsRequestRepository.findByOperatorId(id);
141     }
142
143     public List<UpdatePolisDataRequest> getUnresolvedUpdateRequests(Long id) {
144         Operator operator = repository.findById(id).orElse(null);
145         if (operator == null || !operator.getIsAuthenticated()) return null;
146         return updatePolisDataRequestRepository.findAll()
147                 .stream()
148                 .filter(x -> x.getOperatorId() == null)
149                 .collect(Collectors.toList());
150     }
151
152     public List<InsurancePaymentsRequest> getUnresolvedInsurancePaymentsRequests(Long id) {
153         Operator operator = repository.findById(id).orElse(null);
154         if (operator == null || !operator.getIsAuthenticated()) return null;
155         return insurancePaymentsRequestRepository.findAll()
156                 .stream()
157                 .filter(x -> x.getOperatorId() == null)
158                 .collect(Collectors.toList());
159     }
160
161     @Transactional
162     public CreatePolisRequest makeDecisionForCreateRequest(@NotNull final Long operatorId,
163                                                            @NotNull final Map<String, String> data) {
164         Long requestId = Long.parseLong(data.get("requestId"));
165         Operator operator = repository.findById(operatorId).orElse(null);
166         if (operator == null || !operator.getIsAuthenticated()) return null;
167         boolean approved = Boolean.parseBoolean(data.get("approved"));
168         Optional<CreatePolisRequest> requestOptional = createPolisRequestRepository.findById(requestId);
169         if (!requestOptional.isPresent()) return null;
170         CreatePolisRequest request = requestOptional.get();
171         if (!approved) {
172             request.setStatus(Constants.CreatePolisStatus.DECLINED);
173         } else {
174             request.setStatus(Constants.CreatePolisStatus.APPROVED);
175             Integer cost = Integer.parseInt(data.get("cost"));
176             request.setCost(cost);
177         }
178         return createPolisRequestRepository.save(request);
179     }
180
181     @Transactional
182     public InsurancePaymentsRequest makeDecisionForGetInsurancePaymentsRequest(@NotNull final Long
            operatorId,
183                                                            @NotNull final Map<String, String> data) {
184         Long requestId = Long.parseLong(data.get("requestId"));
185         Operator operator = repository.findById(operatorId).orElse(null);
186         if (operator == null || !operator.getIsAuthenticated()) return null;
187         boolean approved = Boolean.parseBoolean(data.get("approved"));
188         Optional<InsurancePaymentsRequest> requestOptional = insurancePaymentsRequestRepository.findById(
                requestId);
189         if (!requestOptional.isPresent()) return null;
190         InsurancePaymentsRequest request = requestOptional.get();
191         Optional<Client> clientOptional = clientRepository.findById(request.getClientId());
192         if (!clientOptional.isPresent()) return null;
193         Client client = clientOptional.get();
194         if (!approved) {
195             request.setStatus(Constants.InsurancePaymentsStatus.DECLINED);
196         } else {
197             if (operator.getBank() >= request.getPayments()) {
198                 request.setStatus(Constants.InsurancePaymentsStatus.COMPLETED);
199                 operator.setBank(operator.getBank() - request.getPayments());
200                 client.setWallet(client.getWallet() + request.getPayments());
```

```
                clientRepository.save(client);
                repository.save(operator);
            }
        }
        return insurancePaymentsRequestRepository.save(request);
    }

    @Transactional
    public UpdatePolisDataRequest makeDecisionForUpdateDataRequest(@NotNull final Long operatorId,
                                                                   @NotNull final Map<String, String> data) {
        Long requestId = Long.parseLong(data.get("requestId"));
        Operator operator = repository.findById(operatorId).orElse(null);
        if (operator == null || !operator.getIsAuthenticated()) return null;
        boolean approved = Boolean.parseBoolean(data.get("approved"));
        Optional<UpdatePolisDataRequest> requestOptional = updatePolisDataRequestRepository.findById(
            requestId);
        if (!requestOptional.isPresent()) return null;
        UpdatePolisDataRequest request = requestOptional.get();
        if (!approved) {
            request.setStatus(Constants.UpdatePolisDataStatus.DECLINED);
        } else {
            request.setStatus(Constants.UpdatePolisDataStatus.COMPLETED);
            String newData = request.getNewData();
            insurancePolisRepository.findById(request.getInsurancePolisId()).ifPresent( x -> {
                x.setData(newData);
                insurancePolisRepository.save(x);
            });

        }
        request.setOperatorId(operatorId);
        return updatePolisDataRequestRepository.save(request);
    }

    public Operator updateBank(@NotNull final Long operatorId,
                               @NotNull final Integer money) {
        Operator operator = repository.findById(operatorId).orElse(null);
        if (operator == null || !operator.getIsAuthenticated()) return null;
        operator.setBank(operator.getBank() + money);
        return repository.save(operator);
    }
}
```