Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по курсовой работе

по дисциплине «Программное обеспечение распределенных вычислительных систем»

Сервис такси

Выполнил студент гр. 23541/3 Смирнов С.В.

(подпись)

Преподаватель Стручков И.В.

(подпись)

“ ” 2019 г.

Санкт – Петербург

2019

### Цель работы

Разрабатывается система управления сервисом по обслуживанию такси. Назначение такой системы- обеспечение работы клиентов с электронными заявками на заказ такси и их обработка.

### Функциональные требования к системе

Служба заказа такси представляет из себя сервис, предоставляющий возможности вызова такси для клиента, исполнения заказа и оплаты поездки. Предполагается три стороны-участника:

* клиент
* оператор
* водитель.

В системе предусмотрена возможность регистрации новых пользователей для всех ролей.

Функции клиента:

* Просмотр выполненных заказов
* Заполнение заявки на новый заказ

Функции оператора:

* Просмотр новых/отклоненных заказов
* Просмотр свободных водителей
* Назначение водителя на заказ

Функции водителя:

* Подтверждение/отклонение новых заказов
* Просмотр ранее выполненных заказов
* Заполнение данных о совершенной поездке, получение оплаты

### Бизнес процессы в системе

**Заказ такси**

Участники:

* Клиент
* Оператор
* Водитель

Этапы:

1. Клиент оставляет заявку на оказание услуги заполняя форму
2. Оператор принимает заявку, назначает водителя в соответствии с его рейтингом(минимальным количеством жалоб от клиентов)
3. Водитель выбирает заказ(есть возможность принять или отказаться)

**Исполнение заказа**

Участники:

* Водитель
* Клиент

Этапы:

1. Водитель прибывает в назначенный адрес
2. В случае успешной посадки клиента, заказ исполняется.
3. В соответствии с километражом поездки и времени ожидания(водитель вводит значения в поля формы) принимается оплата.

**Подача жалобы**

Участники:

* Клиент
* Оператор

Этапы:

1. Клиент после оплаты оказанной услуги вправе оставить жалобу на видителя
2. Жалоба принимается и фиксируется оператором, который принимает решение о её состоятельности

### Описание вариантов использования

Варианты использования для всех трёх ролей приведены на рис.1.

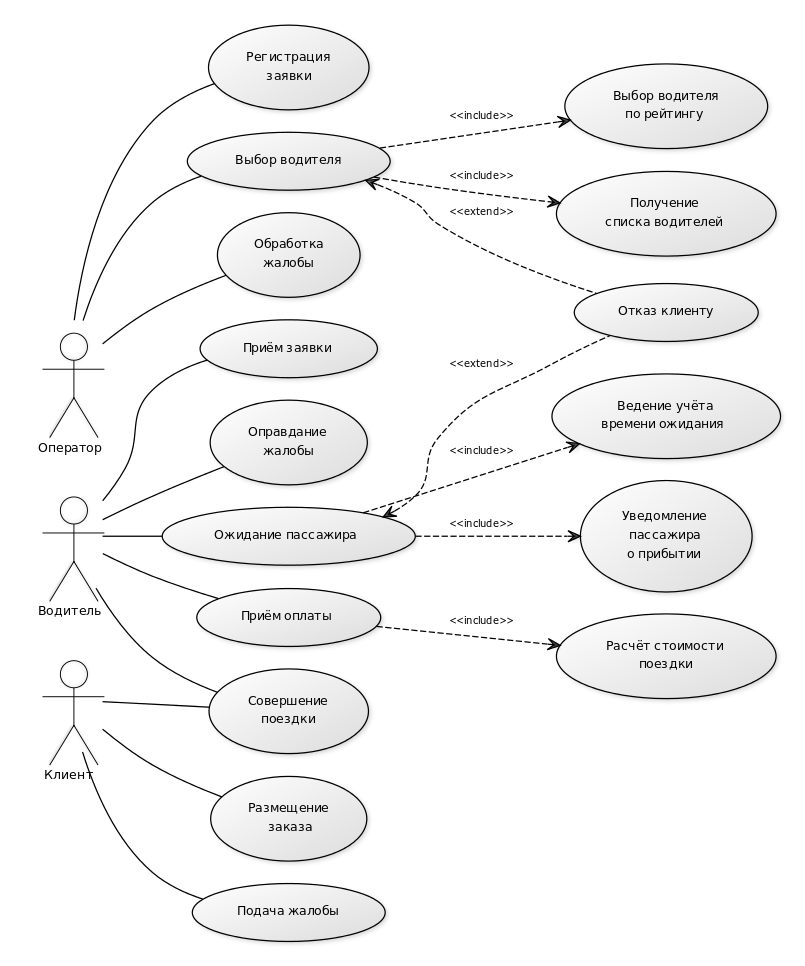


Рис.1. Варианты использования для всех ролей

### Динамическая объектная модель

### https://github.com/Sergei-Smirnov-95/Taxi/raw/master/diag2.jpg

Рис.2. Динамическая объектная модель

### Слой бизнес-логики

Слой бизнес логики был реализован с помощью использования шаблона "Модель предметной области".

В пакете **entity** описаны все сущности, определенные в системе.

Классы данного пакета являются сущностными, т.е. они лишь хранят сущности и имеют методы по извлечению/записи полей. Рассмотрим подробнее классы пакета entity:

* Complaint

Класс, описывающий жалобы. Хранит текст жалобы.

* CostCalculation

Класс, описывающий расчёт стоимости поездки по времени ожидания и пути. Хранит все три вличины.

* Driver

Класс, реализующий методы, такие как обработка занятости водителя, авторизация, выборка выполненных заказов, подтверждение принятие/отмены заказа и др. В методах реализована работа с статусом заказа и выборка необходимых заказов.

* Operator

Класс, реализующий методы авторизации, выборки свободных водителей, назначения водителя на невыполненные заказы.

* Order

Класс, реализующий методы для восстановления заказа, создание нового заказа и обработки статуса заказа.

* OrderStatus

Класс, реализующий методы для работы со статусами заказа.

* Passenger

Класс, реализующий методы для авторизации и первичного создания заказа.

* Tariff

Класс, для хранения параметров расчета стоимости выполненного заказа.

* User

Класс родитель, содержащий общие поля и методы для Passenger, Operator, Driver.

### Слой источников данных

В качестве СУБД была выбрана MySQL. Для работы с базой данных был использован Java Persistance API, в частности его реализация в библиотеке Hibernate. В базе данных хранятся сущности, описанные в пакете entity. Не транслируемые поля помечаются аннотацией @Transient. Описание сущностей производится с помощью специальных аннотаций JPA.

Для организации возможности хранения в одной таблице пользователей трёх типов, необходимо указать в родительском классе(User) столбец по которому будет происходить разделение по типу пользователей (@DiscriminatorColumn(“column\_name”)), а в каждом конкретном классе указать значение данного столбца(@DiscriminatorValue(“value”)).

Для каждой сущности был реализован свой репозиторий. Репозитории были реализованы с помощью библиотеки Spring Data. Стандартные методы по добавлению и извлечению объектов реализуются автоматически при использовании аннотации @Component и запуска приложения как @SpringBootApplication. Таким образом, ручное описание методов потребовалось в минимальном количестве случаев (запросы на извлечение по полю Login).

Для подхватывания аннотаций необходимо указать расположение источников данных:

@Bean  
public DataSource dataSource() {  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
 dataSource.setDriverClassName(env.getRequiredProperty("spring.datasource.driver-class-name"));  
 dataSource.setUrl(env.getRequiredProperty("spring.datasource.url"));  
 dataSource.setUsername(env.getRequiredProperty("spring.datasource.username"));  
 dataSource.setPassword(env.getRequiredProperty("spring.datasource.password"));  
 return dataSource;  
}

@Bean  
public EntityManagerFactory entityManagerFactory() {  
  
 HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
 vendorAdapter.setGenerateDdl(true);  
  
 LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();  
 factory.setJpaVendorAdapter(vendorAdapter);  
 factory.setPackagesToScan("edu.spbstu.taxi");  
 factory.setDataSource(dataSource());  
 factory.setPersistenceUnitName("taxi");  
 factory.setPersistenceProviderClass(HibernatePersistenceProvider.class);  
 factory.setJpaProperties(additionalProreties());  
 factory.afterPropertiesSet();  
 return factory.getObject();  
}

Конкретные значения записываются в файл конфигурации application.properties:

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/taxi\_db?allowPublicKeyRetrieval=true&useSSL=FALSE&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=pass

### Слой представления

В качестве фреймворка для слоя представления была выбрана технология Spring MVC. Данная технология позволяет реализовать паттерн Model-View-Controller при помощи слабо связанных компонентов. В качестве модели выступает слой бизнес-логики приложения. Контроллеры реализованы в виде RESTful контроллеров с помощью фреймворка Spring. Представление реализовано с помощью JavaScript фреймворка AngularJS и HTML страниц.

Работа происходит по следующему сценарию:

* + RESTful сервис принимает запрос от пользователя;
  + сервис вызывает соответствующие методы бизнес логики для обработки запроса;
  + сервис возвращает ответ на запрос в виде JSON-объекта;
  + AngularJS клиент получает ответ от сервера и рендерит соответствующую HTMLстраницу и показывает его в браузере.

Рассмотрим более подробно REST API, предоставляемый сервером

Driver:

* “rest/driver/{login}/authenticate”- запрос на аутентификацию
* “rest/driver/{login}” – запрос на добавление нового водителя
* “rest/driver/{login}/accept” – запрос на подтверждение заявки
* “rest/driver/{login}/decline” – запрос на отклонение заявки
* “rest/driver/{login}/newOrders” – запрос новых заказов для водителя
* “rest/driver/{login}/oldOrders” - запрос выполненных заказов для водителя
* “rest/driver/{login}/pay” – запрос на оплату поездки