Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по курсовому проекту**

**Дисциплина**: Программное обеспечение распределенных вычислительных систем

**Тема**: Система организации экскурсий

Выполнил студент гр. 13541/4 Степанов Д.С.

(подпись)

Руководитель Стручков И.В.

(подпись)

“ ” 2017 г.

Санкт - Петербург

2017

**Цель работы**

Целью данной работы является разработка системы организации экскурсий с использованием фреймворков Spring и Angular JS . Разрабатываемая система предназначена для автоматизации процесса организации экскурсий.

**Выполнение работы**

**Сбор функциональных требований**

Сбор функциональных требований

Описание бизнес-процессов:

Экскурсия Участники: пользователь, транспортировщик, маршрутная система. Сущности: маршрут, автомобиль, фотоотчёт

Этапы:

1) Указание основных объектов и дополнительной информации (дата, кол-во и т.д.)

2) Назначение транспортировщика

3) Добавление в маршрутную систему

4) Добавление туристов

5) Формирование фотоотчётов

Регистрация транспортировщика Участники: транспортировщик, маршрутная система, БД ГИБДД. Сущности: автомобиль

Этапы:

1) Подача заявки на оформление транспортировщиком

2) Проверка введенных данных

3) Добавление в маршрутную систему

Оплата услуг транспортировщика Участники: организатор экскурсии, маршрутная система, транспортировщик Сущности: договор, квитанция, экскурсия, банковская карта Этапы:

1) Организатор экскурсии и транспортировщик договариваются о цене транспортировки

2) Организатор вводит информацию о своей банковской карте

3) Маршрутная система осуществляет оплату транспортировки

4) Система отправляет квитанции транспортировщику и организатору

Система должна:

• Обеспечивать авторизованный доступ к своим сервисам;

• Позволять пользователям создавать экскурсионные маршруты и присоединяться к уже созданным;

• Давать возможность пользователям выкладывать фотографии экскурсионных маршрутов;

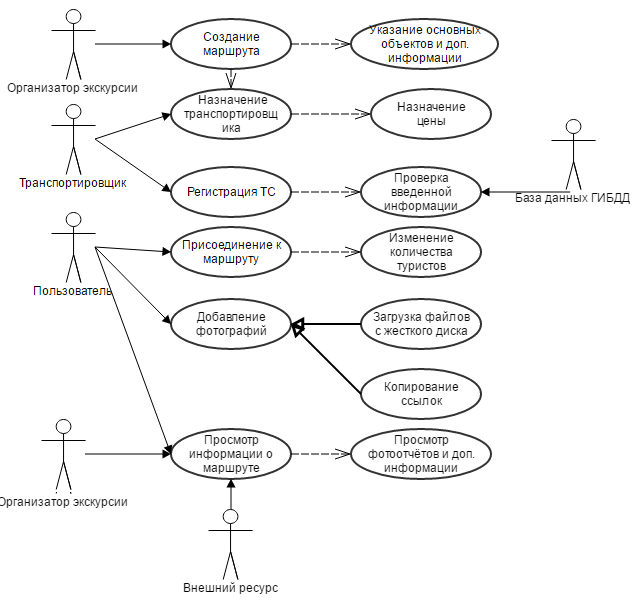
• Позволять транспортировщикам зарегистрировать своё ТС;

• Система на основе численности туристической группы должна предлагать организатору похода наиболее подходящего транспортировщика;

• Выводить дополнительную информацию об экскурсионном маршруте, взятую из открытых источников (википедия или картинки google);

• Формировать фотоотчёты после завершения экскурсии.

**Разработка вариантов использования (обобщенная диаграмма прецедентов)**

 Подробное описание всех вариантов использования (текстовое описание с альтернативами)

**Создание экскурсионного маршрута**

1. Организатор экскурсии пишет текстовое описание маршрута с указанием основных экскурсионных объектов
2. Организатор экскурсии вводит информации о максимальном количестве туристов, о необходимом снаряжении, дате и месте выезда и т.д.
3. Система выводит список транспортировщиков, выделяя более подходящих.
4. Организатор выбирает транспортировщика
5. Транспортировщик подтверждает участие и договаривается о цене с организатором
6. Организатор производит оплату услуг транспортировщика
7. Маршрут добавляется в маршрутную систему

**Альтернатива**

На шаге 2 при вводе некорректной даты система выдает предложение её исправить, после чего переходит на шаг 3

Если на шаге 4 организатор не находит подходящего транспортировщика, то он может отменить создание маршрута

Если на шаге 5 транспортировщик не подтверждает участие или не договаривается о цене, то возвращение на шаг 4

**Регистрация ТС**

1. Транспортировщик вводит информацию о своём ТС, включая страховку и номер свидетельства о регистрации
2. Производится проверка действительности введенных данных с помощью внешнего запроса
3. Добавление ТС в маршрутную систему

**Альтернатива**

Если на шаге 2 проверка действительности не прошла, то возвращение на шаг 1 с указанием поля, где проверка не прошла

**Присоединение к маршруту**

1. Пользователь присоединяется к маршруту
2. Система изменяет количество туристов в указанном маршруте
3. Система добавляет пользователя к участникам похода

**Альтернатива**

Если на шаге 2 превышено максимальное количество туристов, то операция отменяется

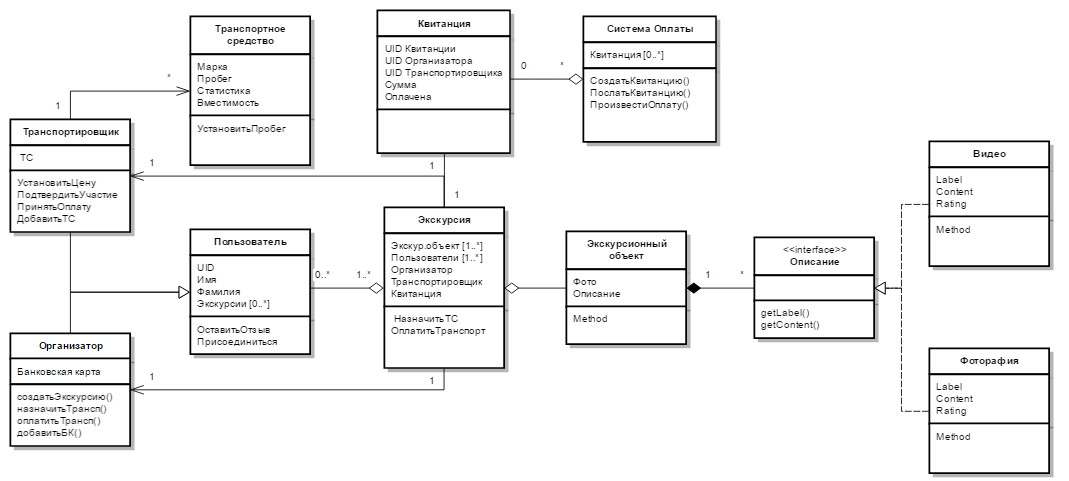
**Добавление фотографий**

1. Пользователь выбирает каким образом будет добавлять фотографии (загружать с диска или добавлять ссылки)
2. Пользователь добавляет фотографии
3. Система добавляет фотографии к маршруту

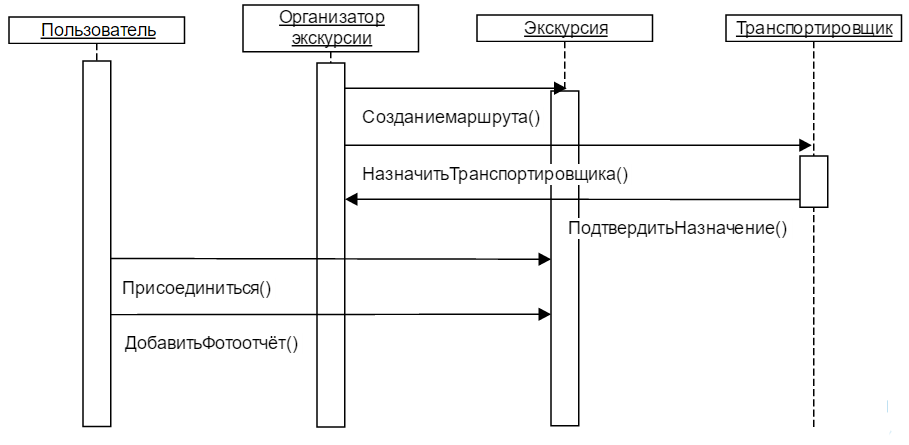
**Просмотр информации о маршруте**

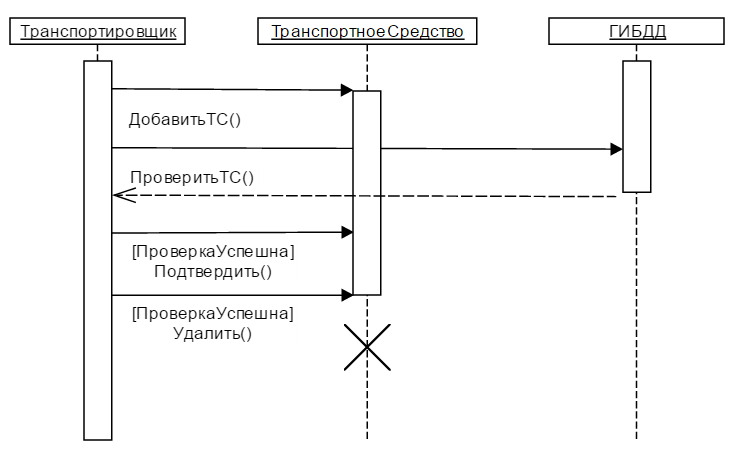
1. Производится запрос к внешнему ресурсу для информации об основных экскурсионных объектах
2. Маршрутная система предоставляет информацию об участниках похода, их контактах, времени выезда, транспортировщике, телефонах местных аварийных служб и т.д

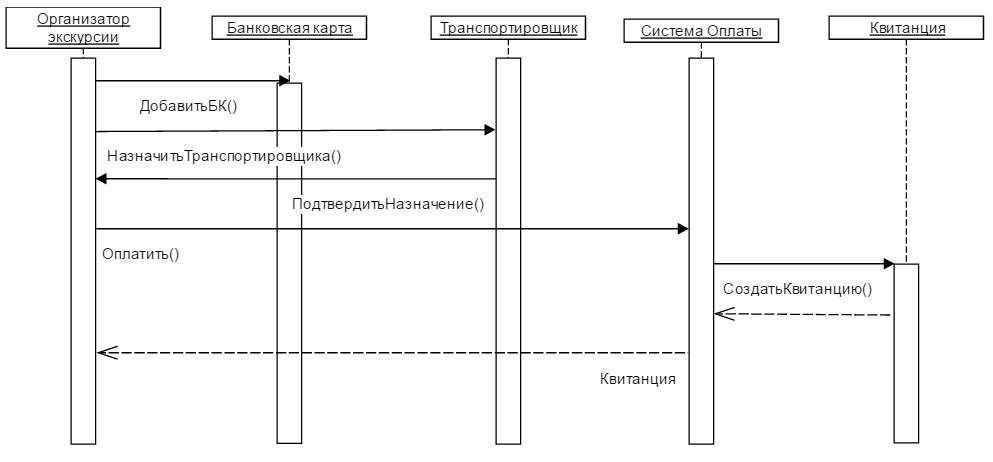
Моделирование предметной области при помощи диаграммы классов



Моделирование предметной области при помощи диаграмм последовательностей







**Бизнес-логика** Для написания бизнес-логики был использован паттерн "Модель предметной области". Данный паттерн был выбран ввиду следующих факторов:

* Предполагается сеть взаимосвязанных объектов
* Паттерн позволяет напрямую отобразить модель в исходный код
* Требуется высокая масштабируемость

Для создания пользователей и экскурсий был применен паттерн "Фабричный метод". Это порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Весь код бизнес-логики находится в пакете [businesslogic](https://github.com/DaniilStepanov/ExcursionOrgSystem/tree/master/src/main/java/businesslogic). Ниже приведено описание всех классов, определяющих роли:

* Класс [User](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - базовый класс, содержащий поля, присущие всем остальным типам пользователей:
  + int getUID() - возвращает UID пользователя
  + String getLogin() - возвращает логин
  + int getMoney() - возвращает текущий баланс пользователя
  + setMoney(int money) - устанавливает баланс
  + subMoney(int amount) - метод для снятия денег
  + addMoney(int amount) - метод для начисления денег
* Класс [Driver](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - класс, определяющий роль водителя. Наследован от класса User:
  + addVehicle(String model, int mileage, int capacity, String numbers) - добавление транспортного средства(ТС)
  + checkVehicle() - верификация ТС с помощью внешнего запроса
  + agree() - установка водителя на экскурсию
  + disagree() - отказ водителя от назначения на экскурсию
  + getVehicle() - получение текущего ТС
  + getExcursion() - получение текущей экскурсии водителя
  + getGivenPrice() - получение назначенной организатором экскурсии премии водителю
  + setDriverFree() - освобождение водителя от экскурсии
  + setDriverBusy(Excursion e) - назначение водителя на экскурсию
  + setGivenPrice(int price) - установка предложенной организатором цены
  + isFree() - свободен ли водитель
  + isAgree() - согласен ли водитель на предложение организатора
* Класс [Organizator](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - класс, определяющий роль организатора. Наследован от класса User:
  + createExcursion(String name) - создание экскурсии
  + getExcursion() - получение текущей экскурсии организатора
  + beginExcursion() - начало текущей экскурсии
  + endExcursion() - завершение текущей экскурсии
  + sendOfferToDriver(Driver d, int price) - метод для отправления предложения водителю
  + setDriver(Driver d) - установка водителя на текущую экскурсию
  + payToDriver(Driver d, int sum) - оплата услуг водителя
* Класс [Vehicle](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - класс, определяющий роль транспортного средства.
  + getModel() - получение модели ТС
  + getMileage() - получение пробега
  + getCapacity() - получение вместимости
  + getNumbers() - получение государственных регистрационных знаков
  + isChecked() - получение информации о том, верифицировано ли ТС
* Интерфейс[Description](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - интерфейс, реализующий все методы для получения информации об экскурсионном объекте
  + getLabel() - получение описания
  + getContent() - получение содержимого Данный интерфейс реализуют 2 класса: [Photo](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8)и [Text](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8)
* Класс [ExcursionObject](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - класс, определяющий экскурсионный объект.
  + printDescriptionInString() - печать описания экскурсионного объекта в строку
  + getObjUID() - получение UID
  + getDescription(int index) - получение описания
  + addDescription(Description d) - добавление описания
* Класс [Excursion](https://github.com/DaniilStepanov/ExcursionOrgSystem/wiki/%D0%A1%D0%BB%D0%BE%D0%B9-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8) - класс, определяющий экскурсию.
  + setExcursionInfo(int minTourists, int maxTourists, String equipment, java.sql.Date departureDate, int status) - установка информации об экскурсии
  + beginExcursion() - начало экскурсии
  + endExcursion() - конец экскурсии
  + addComment(String userName, String comment) - добавление комментария к эскурсии
  + setDriver(Driver d) - назначение водителя
  + addExcursionObject(ExcursionObject obj) - добавление экскурсионного объекта
  + printExcursionInStr() - печать информации об экскурсии в строку
  + getExsursionObjects() - получение всех экскурсионных объектов, входящих в экскурсию
  + addUser(User u) - добавление пользователя
  + delUser(User u) - удаление пользователя
  + getMinTourists() - получение минимального количество туристов
  + getMaxTourists() - получение максимального количество туристов
  + getEquipment() - получение списка необходимой экипировки
  + getDepartureDate() - получение даты отправления
  + getStatus() - получение текущего статуса экскурсии: установка, набор участников, в процессе или окончена
  + getStringStatus() - получение текстового статуса
  + setPay(boolean b, Receipt receipt) - занесение информации об оплате экскурсии
  + getDriver() - получение текущего водителя
  + getReceipt() - получение чека
  + getUsers() - получение списка туристов
  + getUID() - получение UID экскурсии
  + isPaid() - получение информации о том, оплачена ли экскурсия
  + getName() - получение названия экскурсии

**Слой источников данных**

В качестве СУБД была выбрана система MySQL. Для работы с базой данных был использован Java Persistance API. Java Persistence API (JPA) — спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных.

Существует несколько реализаций этого интерфейса, одна из самых популярных использует для этого Hibernate. В базе данных хранятся все сущности, описанные в пакете entity. Entity (Сущность) — класс, связанный с БД с помощью аннотации (@Entity). К такому классу предъявляются следующие требования:

* Должен иметь пустой конструктор (public или protected)
* Не может быть вложенным, интерфейсом или enum
* Не может быть final и не может содержать final-полей/свойств
* Должен содержать хотя бы одно @Id-поле

При этом entity может:

* Содержать непустые конструкторы
* Наследоваться и быть наследованным
* Содержать другие методы и реализовывать интерфейсы

Стоит отметить, что Entities могут быть связаны друг с другом (один-к-одному, один-ко-многим, многие-к-одному и многие-ко-многим).

Ниже представлен пример Entity класса:

@Entity  
**public class** Driver {  
 @Id  
 @Column(name = **"ID"**)  
 @GeneratedValue  
 **private** Long **id**;  
  
 @Column(name = **"ISFREE"**)  
 **private boolean isFree**;  
  
 @Column(name = **"ISAGREE"**)  
 **private boolean isAgree**;  
  
 @Column(name = **"GIVENPRICE"**)  
 **private int givenPrice**;  
  
 @OneToOne  
 **private** User **user**;  
  
 @OneToOne  
 @JoinColumn(name = **"vehicle\_id"**)  
 **private** Vehicle **vehicle**;  
  
 @OneToOne  
 **private** Excursion **excursion**;

}

Репозитории для каждой сущности были реализованы с помощью библиотеки Spring-data.

Spring Data позволяет легче создавать Spring-управляемые приложения которые используют новые способы доступа к данным. Для использования библиотеки необходимо создать интерфейс, наследованный от JpaRepository. Наследование от JpaRepository дает возможность вызывать такие методы как:

* save
* findOne
* exists
* findAll
* count
* delete
* deleteAll

без необходимости реализовывать их имплементацию.

Для каждого класса из Entity был реализован свой интерфейс, содержащий необходимые методы, которые были автоматически сгенерированы. Ниже приведен пример репозитория для класса Driver:

@Repository  
**public interface** DriverRepository **extends** JpaRepository<Driver, Long> {  
 Optional<Driver> findById(Long id);  
 Optional<Driver> findByUserLogin(String login);  
 Optional<Driver> findByUserId(Long id);  
 List<Driver> findAll();  
}

**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. Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними. В данной работе модель – это бизнес-логика. Контроллер – RESTful контроллер. Представление – AngularJS[4] и HTML.

Ключевое отличие между традиционным MVC контроллером и контроллером RESTful web-сервиса выше в создании тела HTTP-ответа. Вместо того, чтобы опираться на view-технологию для рендеринга на серверной стороне сообщения приветствия в HTML, RESTful web-сервис контроллер просто заполняет и возвращает Greeting объект. Данные объекта будут записаны напрямую в HTTP-ответ как JSON.

Этот код использует @RestController аннотацию, помечая класс как контроллер, где каждый метод возвращает объект вместо представления.

Greeting объект должен быть конвертирован в JSON. Благодаря поддержке Spring HTTP конвертера, не требуется выполнять эту конвертацию вручную.

Цель фреймворка AngularJs — расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки. AngularJS спроектирован с убеждением, что декларативное программирование лучше всего подходит для построения пользовательских интерфейсов и описания программных компонентов, в то время как императивное программирование отлично подходит для описания бизнес-логики. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двустороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление.

С помощью директив AngularJS можно создавать пользовательские HTML-теги и атрибуты, чтобы добавить поведение некоторым элементам:

* ng-bind. Автоматически заменяет текст HTML-элемента на значение переданного выражения.
* ng-controller. Определяет JavaScript-контроллер для вычисления HTML-выражений в соответствии с MVC.
* ng-repeat. Создает экземпляр для каждого элемента из коллекции.
* ng-show и ng-hide. Показывает или скрывает элемент в зависимости от значения логического выражения.
* ng-if. Удаляет или создает часть дерева в зависимости от значения выражения. Если значение выражения, назначенного ngIf, равно false, элемент удаляется из DOM, иначе — вновь клонированный элемент вставляется в DOM.
* …

При неправильной работе серверная сторона генерирует исключение с сообщением о ошибке. Исключение перехватывается и генерируется alert с соответствующим сообщением.

Ниже приведен пример контроллера:

**function** *UserService*($resource) {  
 **return** $resource(**'rest/user/:login'**, { **login**: **'@login'** });  
}  
  
**function** *DriverService*($resource) {  
 **return** $resource(**'rest/driver/:login'**, { **login**: **'@login'**});  
}  
  
**function** *AgreeService*($resource) {  
 **return** $resource(**'rest/driver/:login/agree'**, { **login**: **'@login'**});  
}  
  
**function** *DisagreeService*($resource) {  
 **return** $resource(**'rest/driver/:login/disagree'**, { **login**: **'@login'**});  
}  
  
**function** *AllExcursions*($resource) {  
 **return** $resource(**'rest/user/allExcursions'**, {  
 **query**: {**method**: **'get'**, **isArray**: **true**, **cancellable**: **true**}  
 });  
}  
  
**function** *DriverController*($scope, $routeParams,  
 UserService,  
 DriverService,  
 AgreeService,  
 DisagreeService,  
 AllExcursions) {  
  
 **var** *url* = **function** () {  
 **return** {**login**:$routeParams.**login**};  
 };  
  
 **this**.**user** = UserService.get(*url*());  
 **this**.**dr** = DriverService.get(*url*());  
 **this**.**allExcursions** = AllExcursions.**query**(*url*());  
  
 **this**.addVehicle = **function** () {  
 **window**.**location**.**href** = **'#/driver/'** + **this**.**user**.**login** + **'/addVehicle'**;  
 };  
  
 **this**.agree = **function** () {  
 AgreeService.**query**(*url*());  
 };  
  
 **this**.disagree = **function** () {  
 DisagreeService.**query**(*url*());  
 };  
  
 **this**.radioChanged = **function** () {  
 $scope.**desc** = $scope.role;  
 };  
  
}  
  
**angular** .*module*(**'app'**)  
 .**factory**(**'UserService'**, *UserService*)  
 .**factory**(**'DriverService'**, *DriverService*)  
 .**factory**(**'AgreeService'**, *AgreeService*)  
 .**factory**(**'DisagreeService'**, *DisagreeService*)  
 .**factory**(**'AllExcursions'**, *AllExcursions*)  
 .controller(**'DriverController'**, *DriverController*);

Соответствующая html-страница:

<**div style="padding-left**: 30**px**;**"**>  
 <**h3**>Welcome, {{driver.**user**.**name**}}</**h3**>  
 <**p**>**&nbsp;**</**p**>  
  
 Model: {{driver.**dr**.vehicle.model}} <**br**/>  
 Capacity: {{driver.**dr**.vehicle.capacity}}<**br**/>  
 Numbers: {{driver.**dr**.vehicle.numbers}}<**br**/>  
 <**div ng-if="driver.dr.vehicle.isChecked == true"**>  
 Checked <**br**/>  
 </**div**>  
 <**div ng-if="driver.dr.vehicle.isChecked == false"**>  
 Not checked <**br**/>  
 </**div**>  
  
 <**p**>  
 <**button ng-click="driver.addVehicle()"**>Add new vehicle</**button**> **&nbsp;** <**button ng-click="driver.checkVehicle()"**>Check vehicle</**button**>  
 </**p**>  
  
 <**p**> List of excursions:  
 <**div ng-repeat="*it* in driver.allExcursions"**>  
 <**div**><**input type="radio" ng-value="*it*.description" ng-model="$parent.role" ng-change="driver.radioChanged()"**/>{{***it***.**name**}}  
 </**div**>  
 </**div**>  
 Description: <**span ng-bind="desc"**></**span**>  
  
 <**div ng-if="driver.dr.givenPrice != 0"**>  
 You have an offer from {{driver.**dr**.**excursion**.**name**}} <**br**/>  
 Money: {{driver.**dr**.givenPrice}}  
 <**p**>  
 <**button ng-click="driver.agree()"**>Yes</**button**>  
 <**button ng-click="driver.disagree()"**>No</**button**>  
 </**p**>  
 </**div**>  
</**div**>

**Тестирование**

Все бизнес процессы были протестированы с помощью JUnit. JUnit — библиотека для модульного тестирования программного обеспечения на языке Java. Функциональность:

* assertEquals
* assertFalse
* assertNotNull
* assertNull
* assertNotSame
* assertSame
* assertTrue

В файле [DriverTest](https://github.com/DaniilStepanov/ExcursionOrgSystem/blob/master/src/test/java/DriverTest.java) содержатся следующие тесты водителя:

* testDriverCreation() - проверка правильного создания водителя и добавления автомобиля
* testFree() - проверка правильности назначения на экскурсию и освобождения от неё водителя

В файле [ExcursionTest](https://github.com/DaniilStepanov/ExcursionOrgSystem/blob/master/src/test/java/ExcursionTest.java) содержатся следующие тесты для экскурсии:

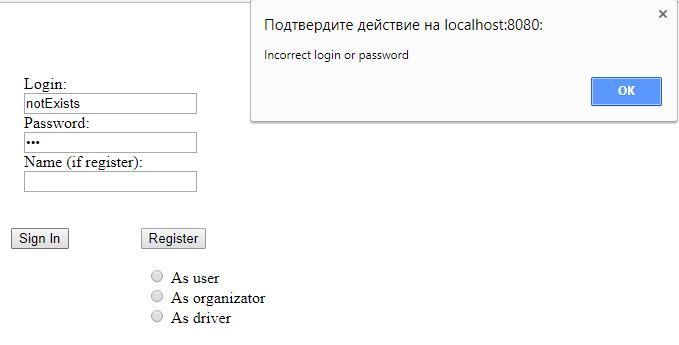
* testCreation() - проверка правильного создания и добавления экскурсионных объектов
* testAddingUsers() - проверка правильности добавления пользователей в экскурсию
* testDriverWithoutVehicleSetting() - проверка обработки назначения водителя без автомобиля на экскурсию
* testDeletion() - проверка правильности удаления экскурсии

В файле [MainTest](https://github.com/DaniilStepanov/ExcursionOrgSystem/blob/master/src/test/java/MainTest.java) содержатся тесты бизнес-процессов

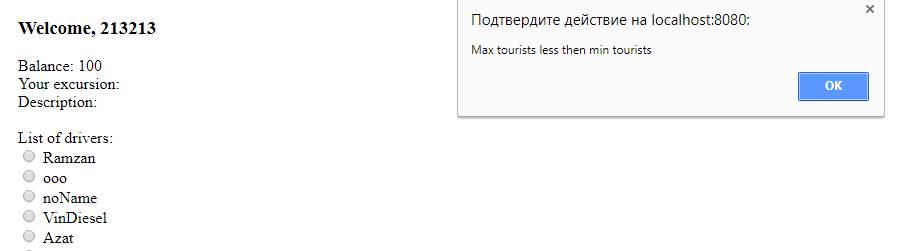
**Функциональное тестирование**

Функциональное тестирование приложения производилось вручную через графический пользовательский интерфейс. Были протестированы все бизнес-процессы в приложении, проверена обработка ошибочных ситуаций. Список тестов:

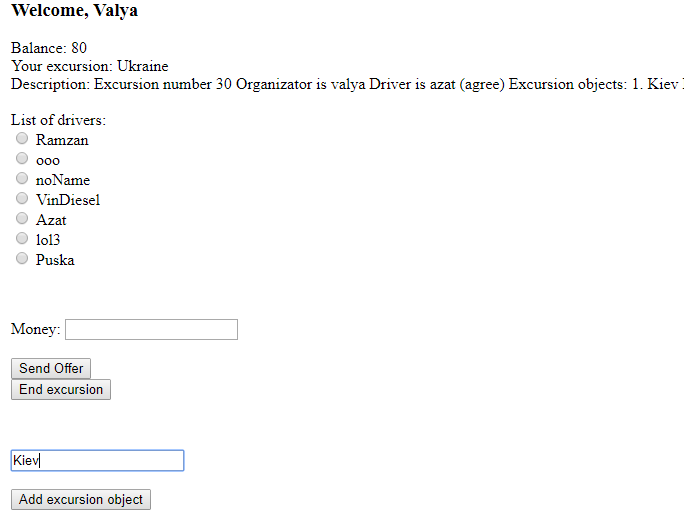
* Попытка залогиниться с неправильным логином или паролем:



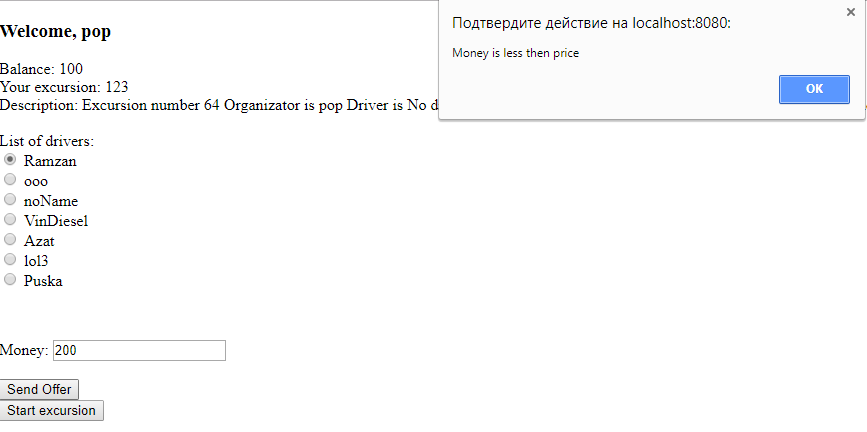
* Попытка создать экскурсию с некорректными параметрами:



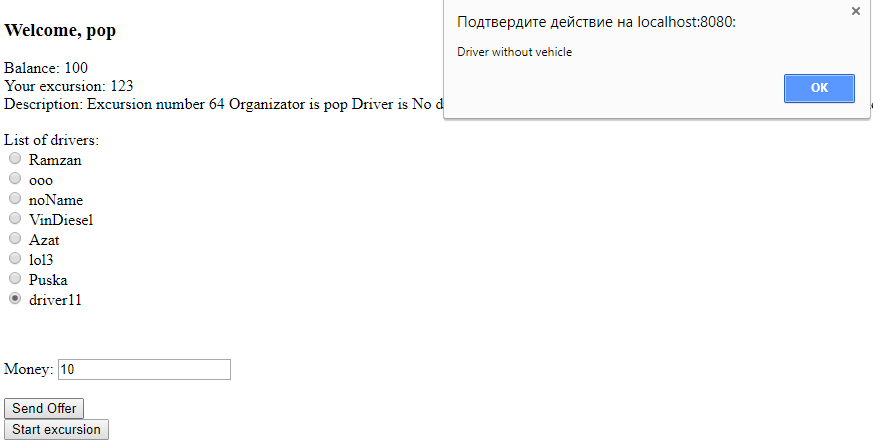
* Попытка создать экскурсионный объект:



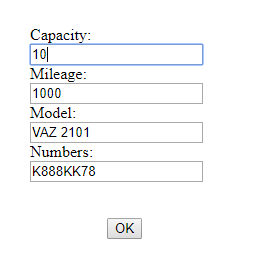
* Попытка предложить водителю сумму более, чем есть на счету:



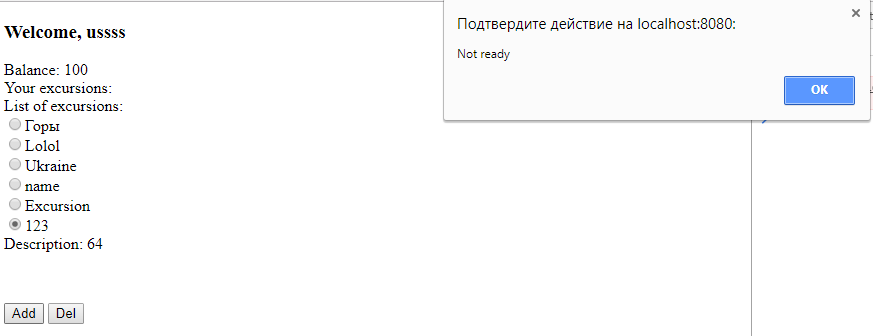
* Попытка послать предложение водителю без верифицированного транспортного средства:



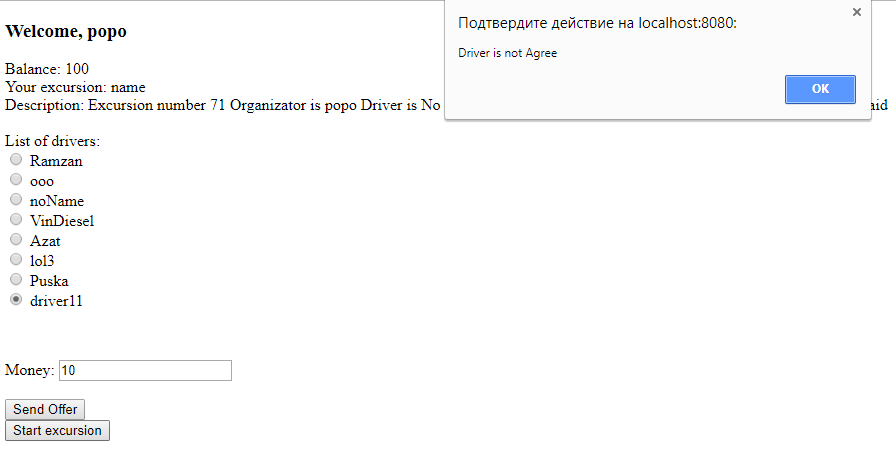
* Добавление нового транспортного средства:



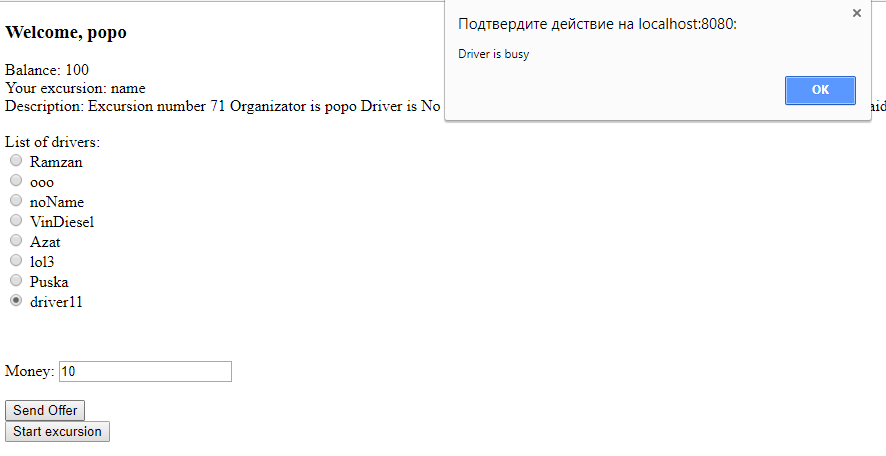
* Но пока водитель на экскурсию не назначен, пользователь не может присоединиться к ней:



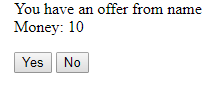
* Попытка начать экскурсию с не согласившимся водителем:



* Попытка добавить водителя, который уже занят:

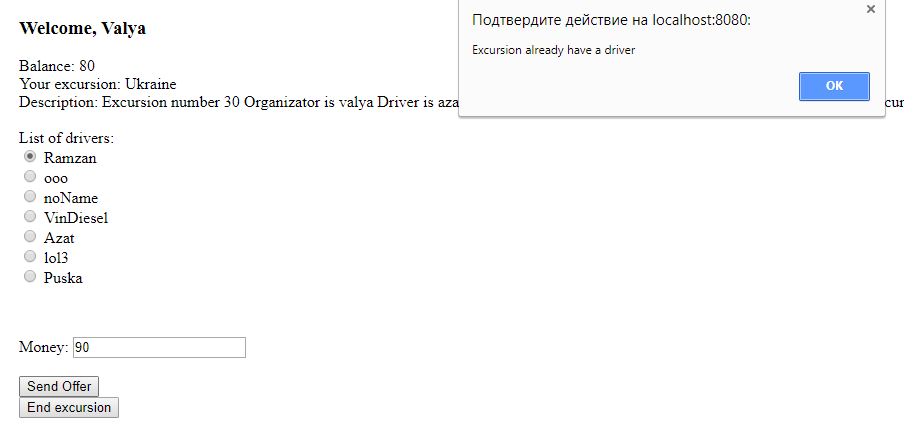


* После отправки предложения водителю оно появляется в соответствующем окне, водитель может согласиться или нет

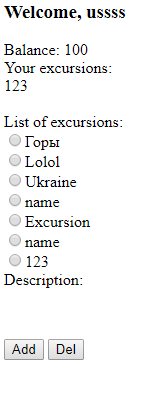


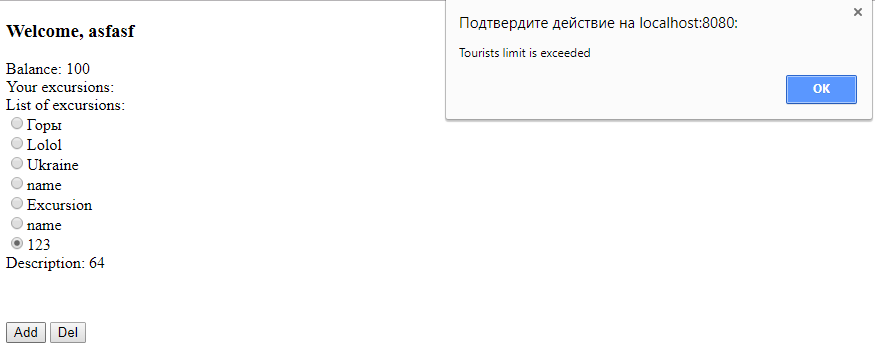
После согласия окно организатора должно преобразоваться: водитель, который согласился помечается отметкой (Your driver), а в окне с информацией об экскурсии напротив водителя появляется надпись (agree)

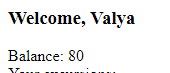
* Попытка добавить водителя, если другой уже назначен:

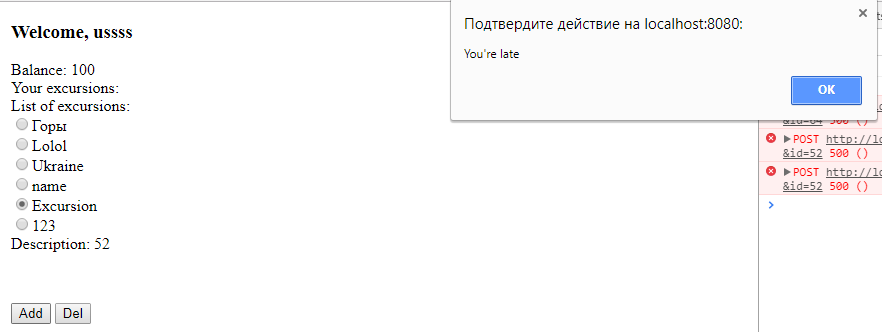


* Попытка добавления пользователя к экскурсии с назначенным водителем:



* Попытка добавления пользователя к экскурсии без свободных мест: 
* После начала экскурсии должна произойти оплата услуг водителя и, соответственно, измениться баланс водителя и организатора:



* Попытка добавить пользователя после завершения экскурсии: 

**Инструкция системного администратора**

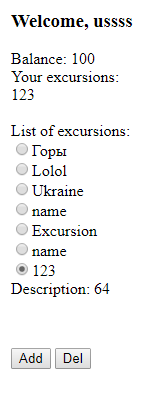
* Скачать и распаковать сервер Apache Tomcat 7 по ссылке https://tomcat.apache.org/download-90.cgi
* Скачать проект по ссылке https://github.com/DaniilStepanov/DistExcursionOrgSystem
* Установить СУБД MySQL.
* Создать базу данных с названием eosdb
* В файле application.properties задать параметры БД
* Собрать war-архив
* Запустить сервер Tomcat

**Инструкция пользователя**

На главной странице сайта находится форма авторизации. Если пользователь хочет зарегистрироваться, он вводит дополнительное поле «имя» и выбирает роль, отметив соответствующую кнопку в нижней части формы. В зависимости от выбранной роли, перед пользователем могут появиться либо окно пользователя, либо окно организатора либо окно водителя.

**Окно пользователя**

Окно пользователя выглядит следующим образом:



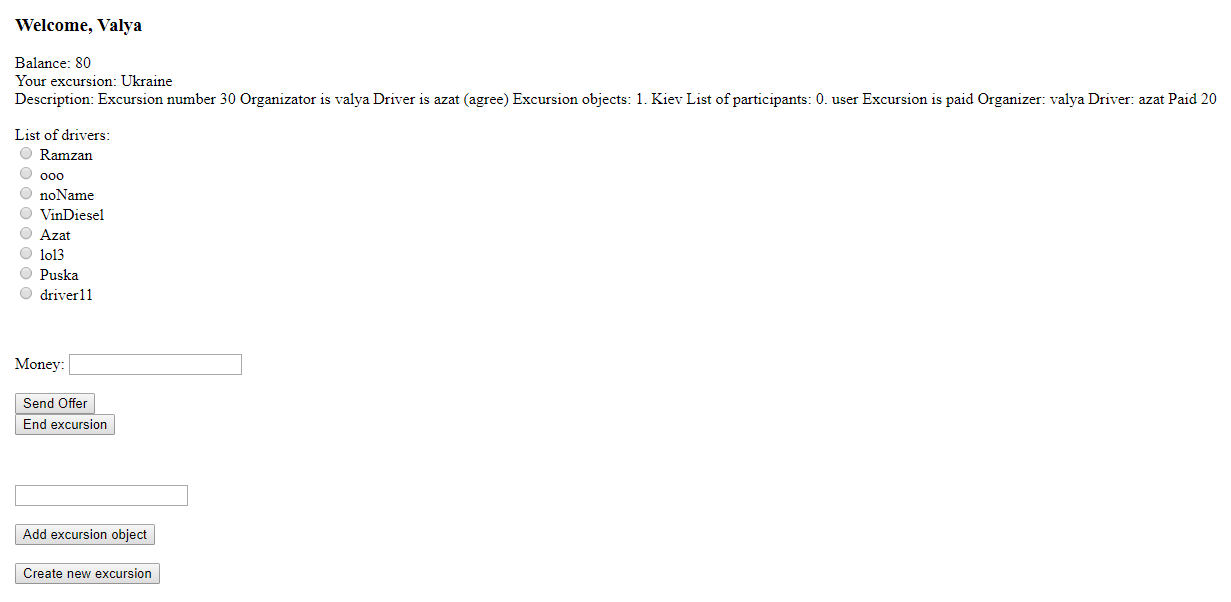
В нем отображаются следующие параметры пользователя:

* Имя
* Баланс
* Список экскурсий пользователя
* Список всех экскурсий

Выбрав экскурсию из списка и нажав кнопку add или del, можно соответственно попытаться присоединиться к выбранной экскурсии или выйти из неё.

**Окно организатора**

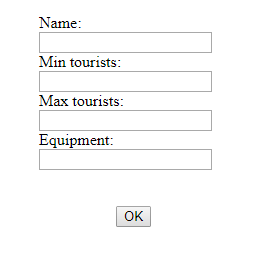
Окно организатора выглядит следующим образом:



В нем отображаются следующие параметры организатора:

* Имя
* Баланс
* Название организуемой экскурсии и ее описание
* Список всех водителей
* Строка для ввода денежного предложения выбранному водителю
* Кнопка для отправки предложения водителю
* Кнопка начала экскурсии, или, если экскурсия уже начата, кнопка завершения
* Поле для ввода описания экскурсионного объекта, который будет добавлен к текущей экскурсии
* Кнопка создания текущей экскурсии

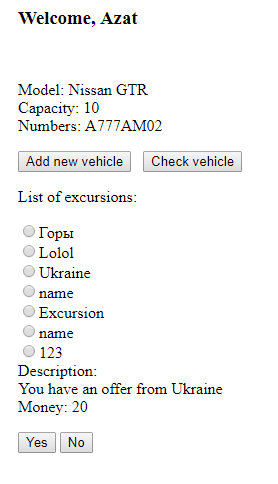
При нажатии на кнопку создания новой экскурсии появляется следующее окно для ввода данных:



При нажатии на кнопку ОК экскурсия будет создана и сохранена.

**Окно водителя**

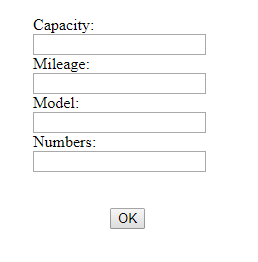
Окно водителя выглядит следующим образом:



В нем отображаются следующие параметры водителя:

* Имя
* Баланс
* Информация об автомобиле
* Кнопка для добавления нового автомобиля
* Кнопка для верифицирования нового автомобиля
* Список экскурсий
* При наличии предложений работы, снизу появится текущее предложение

При нажатии на кнопку создания новой экскурсии появляется следующее окно для ввода данных:



При нажатии на кнопку ОК автомобиль будет создан и сохранен.

**Вывод**

В ходе данной работы была реализована система организации экскурсий. В процессе выполнения работы были изучены архитектурные шаблоны и шаблоны проектирования слоев программного обеспечения. Изученные шаблоны проектирования были затем для создания собственного приложения. Для проектирования слоя бизнес-логики был использован шаблон "Модель предметной области". Для организации слоя источников-данных был использован стандарт JPA и его реализация Hibernate, что позволило не писать скрипты на языке MySQL, достаточно просто создать необходимый Java-класс со специальными аннотациями. Репозитории создавались с помощью библиотеки Spring-data, которая показала себя очень удобной и простой в использовании. В предыдущей версии проекта, с репозиториями, написанными вручную, возникало очень много багов. Использование Spring-data позволило легко и изящно решить данную проблему.

Также были изучены простые примеры создания RESTful-клиентов с использованием AngularJS. AngularJS показал себя удобным средством с большим количеством документации, но возможностей IDE Inellij Idea не хватает, чтобы подхватывать ошибки на лету и подсвечивать их в реальном времени, что приводит к большим временным затратам на их поиск и исправление.

Возможные улучшения проекта:

* проведение полного функционального тестирования
* добавление возможности прикрепления фотографий
* оптимизация написанного кода
* изучение CSS и добавление дизайна

**Список использованных источников:**

1. Java Persistence Api. https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html
2. Spring-data. http://projects.spring.io/spring-data/
3. Spring MVC. https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html
4. Angular JS. https://angularjs.org/