# Отчет о проверке № 7557754

Дата выгрузки: 2022-12-19 23:25:53
Пользователь: dmitry.vert@mail.ru, ID: 7557754

Отчет предоставлен сервисом «Антиплагиат»
на сайте antiplagius.ru/

## Информация о документе

№ документа: 7557754
Имя исходного файла: ИКБО-24-20_РСЧИР_Верт.docx
Размер файла: 3.88 МБ
Размер текста: 35257
Слов в тексте: 4467
Число предложений: 252

## Информация об отчете

Дата: 2022-12-19 23:25:53 - Последний готовый отчет
Оценка оригинальности: 100%
Заимствования: 0%

100.00% 0%

## Источники:

| Доля в тексте | Ссылка |
|---|---|

## Информация о документе:

МИНОБРНАУКИ РОССИИФедеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет" РТУ МИРЭА Институт информационных технологий (ИТ) Кафедра инструментального и прикладного программного обеспечения (ИиППО) КУРСОВАЯ РАБОТА по дисциплине: Разработка серверных частей интернет-ресурсов по профилю: Разработка программных продуктов и проектирование информационных систем направления профессиональной подготовки: 09.03.04 "Программная инженерия" Тема: Серверная часть веб-приложения "Сервис управления прокатом электросамокатов" Студент: Верт Дмитрий Андреевич Группа: ИКБО-24-20 Работа представлена к защите _____/Верт Д.А./ (подпись и ф.и.о. студента) Руководитель: Синицын Анатолий Васильевич, старший преподаватель Работа допущена к защите _____/Синицын А.В./ (подпись и ф.и.о. рук-ля) Оценка по итогам защиты: _____ _____ / _____ / _____ / _____ / (подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту) М. РТУ МИРЭА. 2022 АННОТАЦИЯ Отчёт содержит 47 страниц, 36 иллюстраций, 6 листингов (большая часть из которых расположены в приложении А в конце отчёта), 4 литературных источника. Целью данной курсовой работы являлось создание серверной части веб-приложения на тему "Сервис управления прокатом электросамокатов". В результате выполнения задания было разработано серверное приложение с MVC (Model View Controller) архитектурой, предоставляющее RESTful (Representational State Transfer) API (Application Programming Interface), который обеспечивает функционал по регистрации, авторизации и редактирования информации пользователей, поиска ближайших точек проката, тарификации использования самокатов (по тарифной ставке, либо по подписке), просмотра истории поездок, а так же по управлению точками проката и самокатами для администраторов. В дополнение в вышеописанному был разработан элементарный одностраничный сайт с документацией, позволящий выполнять все эти манипуляции в более удобном для пользователей формате графического интерфейса. Для реализации слоя серверной логики использовалась Java 11, Spring Framework (Spring Context, Spring MVC, Spring Security, Spring Boot), в качестве ORM-фреймворка (Object-Relational Mapping) применялся Hibernate, система сборки и контроля зависимостей - Maven, для тестов были использованы JUnit и Mockito, а в роли СУБД (система управления базами данных) выступила PostgreSQL. Исходный код разработанного приложения доступен для просмотра и скачивания по соответствующей ссылке в сети Интернет. СОДЕРЖАНИЕ ПЕРЕЧЕНЬ СОКРАЩЕНИЙ 5 ВВЕДЕНИЕ 7 1 ОБЩИЕ СВЕДЕНИЯ 8 1.1 Обоснование выбора средств ведения разработки 8 2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ 9 3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ 10 3.1 Анализ предметной области 10 3.2 Разработка серверной части веб-приложения 13 3.3 Разработка клиентского представления веб-приложения 23 3.4 Обзор разработанного

программного продукта 25 3.5 Подготовка к защите курсовой работы 37 ЗАКЛЮЧЕНИЕ 38 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 39 ПРИЛОЖЕНИЕ А 41 ПЕРЕЧЕНЬ СОКРАЩЕНИЙ БД-База данныхСУБД-Система управления базами данныхAPI-Application Programming Interface (программный интерфейс)CRUDCreate, Read, Update, Delete (перечень базовых операций - создания, чтения, обновления, удаления)CSS-Cascading Style Sheets (формальный язык описания внешнего вида документа, написанного с использованием языка разметки)DAO-Data Access Object (объект, предоставляющий стандартные операции над сущностями, такие как добавление, обновление и удаление)DDL-Data Definition Language (группа операторов определения данных в SQL)DML-Data Manipulation Language (группа операторов управления данными в SQL)DTO-Data Transfer Object (объект, представляющий программную сущность и хранящий соответствующие данные из полей БД)HTML-HyperText Markup Language (стандартизированный язык разметки документов для просмотра веб-страниц в браузере)IDE-Integrated Development Environment (интегрированная среда разрабоотки - комплекс программных средств, используемый программистами для разработки программного обеспечения)JDK-Java Development Kit (бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java)JPA-Java Persistence API (спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных)JSON-JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript)JWT-JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON)MVC-Model View Controller (архитектура проектирования приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер)MVP-Minimum Viable Product (минимально жизнеспособный продукт - продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями)ORM-Object-Relational Mapping (технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая "виртуальную объектную базу данных)POJO-Plain Old Java Object ("старый добрый Java-объект", простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели)REST-Representational State Transfer (архитектурный стиль взаимодействия компонентов распределённого приложения в сети)UI-User Interface (пользовательский интерфейс) ВВЕДЕНИЕ На данный момент информационные технологии проникли во все сферы государства, общества и бизнеса. Трудно представить организацию любого процесса без задействия современных технологий, поскольку они значительно его упрощают, систематизируют и оптимизируют. В частности, каждое уважающее себя учебное учреждение имеет свой интернет-портал, а любой сколько-нибудь значительный бизнес обязан иметь собственный сайт или приложение для ведения предпринимательской деятельности через Интернет. Более того, некоторые бизнес-услуги могли и вовсе быть немыслимы до широкого распространения веб-приложений. Целью данной курсовой работы является разработка серверной части веб-приложения "Сервис управления прокатом электросамокатов" с использованием современных технологий на базе фреймворка Spring и архитектурного паттерна проектирования MVC [1]. Для систематизации работы и упрощения восприятия доклада процесс разработки был поделен на четыре основные части: 1. Изучение и выбор наиболее удобных и подходящих средств ведения разработки 2. Анализ предметной области 3. Разработка серверной части веб-приложения 4. Разработка клиенткого представления веб-приложения (графического интерфейса) Приложение, созданное в результате выполнения курсовой работы должно иметь широкий и завершенный функционал для взаимодействия со спроектированной системой, базовый пользовательский интерфейс (в данном случае это сайт с документацией, который позволяет отсылать запросы серверу по разработанному API), реализовывать серверную логику написанную на Java с помощью Spring Framework, работать с реляционной базой данных и соответствовать архитектуре MVC [2]. 1 ОБЩИЕ СВЕДЕНИЯ 1.1 Обоснование выбора средств ведения разработки Первым этапом любой разработки является выбор и установка соответсвующего программного обеспечения. Так в качестве среды разработки была выбрана IntelliJ IDEA Ultimate. Преимуществами IDEA является статус де факто стандарта при выборе IDE (Integrated Development Environment) в разработке на Java. В роли СУБД была выбрана PostgreSQL, так как она совершенно бесплатна, широко распространена и имеет продвинутый функционал. Для организации ORM (Object-Relational Mapping) и выполнения запросов к базе данных из среды Java был применён фреймворк Hibernate - самое популярное решение на данный момент. Версия JDK (Java Development Kit) выбрана 11, так как она стабильна и достаточно современна, включает в себя все самые важные и необходимые функции языка Java [3]. А где Java, там почти всегда и Spring [4] - стандарт при выборе фреймворка для разработки enterprise-приложений на Java. Развертываться приложение будет на Spring Boot, поскольку он обеспечивает простейшую "из коробки" интеграцию с остальным Java-приложением на Spring. В качестве системы сборки используется Maven, как наиболее популярный и гибко кастомизируемый инструмент. Для тестирования использовалась библиотека модульного тестирования JUnit и некоторые инструменты фреймворка Mockito [5], что опять же очень стандартно для разработки на Java. При разработке UI (User Interface) применялся Swagger [6] - Java-реализация спецификации OpenAPI 3.0, которая выполняет автоматическую генерацию документации на основе разработанного API. В результате предоставляя одностраничный сайт, написанный на HTML (HyperText Markup Language) и CSS (Cascading Style Sheets), на котором можно ознакомиться со всеми эндпоинтами проекта и их описанием, а так же отправить тестовые запросы серверу непосредственно из вкладки в браузере. Swagger был выбран из-за своего удобства и простоты, так как основной упор проекта - это бэкенд. 2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ Разработанное приложение несёт собой демонстративную, учебную цель. Оно подразумевается для использования в ознакомительных целях - в качестве прототипа, либо примера приложения на котором можно изучить то, как архитектурно и в деталях устроен современный программный продукт, а также то, из каких компонентов состоит серверная часть в общем цикле разработки распределенного клиент-серверного приложения [7]. Целевой аудиторией данного программного продукта могут быть начинающие backend-разработчики, желающие разобраться в устройстве полноценного RESTful веб-сервиса, либо frontend-разработчики, которым для создания своего pet-проекта может понадобиться внешний API, с которого можно брать и отсылать какие-либо данные. 3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ 3.1 Анализ предметной области Сбор информации был начат с изучения уже существующих на рынке аналогов. Для этого было скачано и

изучено популярное приложение для проката электросамокатов "Whoosh". На рисунке 1 представлен весь предоставляемый приложением функционал, основные элементы которого - выбор ближайшей точки проката и соответствующего самоката (рисунок 2), настройка профиля пользователя и просмотр истории поездок (рисунок 3) и возможность аренды как по подписке, так и по поминутной фиксированной ставке (рисунок 4), которые являются ядром всего приложения, на чем базируются все подобные сервисы. Рисунок 1 - Полный перечень опций, доступных в приложении "Whoosh" Рисунок 2 - Отображение ближайщих точек проката и заряда самокатов Рисунок 3 - Настройка профиля пользователя и просмотр истории поездок Рисунок 4 - Возможность покупки подписки или оплаты по тарифу Проанализировав возможности данного сервиса можно сделать вывод, что для конкурентноспособности разрабатываемого приложения его MVP (Minimum Viable Product) должен обладать уже перечисленными выше функциями, а именно: 1. Регистрация, авторизация, редактирование информации пользователя 2. Получение информации о ближайших точках проката и заряде самокатов на них 3. Возможность проката как по подписке, так и по поминутному тарифу 4. Просмотр истории завершенных поездок При этом такие "фичи" как промо коды, безлимит на один день, гайд по использованию, информация о приложении и политика конфиденциальности кажутся второстепенными или вовсе не относящимися к труду разработчика. 3.2 Разработка серверной части веб-приложения Данное приложение разрабатывалось по архитектуре MVC [8], в разработке бэкенда на Java это интерпретируется следующим образом (рисунок 1): все классы разделены в своем взаимодействии на 3 логических уровня - слой модели (это сущности, DAO (Data Access Object) или JpaRepository, и сервисы для работы над этими сущностями), слой контроллера (в современной разработке принято делать контроллеры максимально тонкими, чтобы они лишь маппили запрос с клиента на соответствующий метод сервиса) и слой представления (так как приложение представляет RESTful API, то им можно считать возвращаемые в формате JSON результаты запросов). Концептуально работа приложения выглядит так: запросы приходят по API на контроллеры, контроллеры передают их соответствующим методам сервисов, а сервисы оперируют над сущностями, полученными из базы данных через DAO [9]. Рисунок 5 - Структура пакетов проекта в соответствие с MVC 3.2.1 Проектирование базы данных Проведя анализ предметной области и выделив функциональные требования к проекту, следующим этапом необходимо провести выделение основных сущностей в будущей системе. Так, на рисунке 2 представлена разработанная схема базы данных, включающая в себя такие сущности как: геолокация, точка проката, модель самоката, экземпляр самоката, тариф, подписка, пользователь, поездка, а так же связи различные между ними. Рисунок 6 - Схема спроектированной базы данных База данных была создана в PostgreSQL с помощью двух скриптов - первый скрипт задает структуру таблиц с помощью DDL-команд, а второй - заполняет наобходимыми инициализирующими данными используя DML-команды в SQL [10]. На листинге 1 можно наблюдать отрывок из DML-скрипта, который вносит в таблицу данные о точках проката (настоящие географические объекты с соответствующими координатами на карте мира), о тарифах и подписках, а так же о моделях самокатов (их реальные характеристики были взяты с Яндекс.Маркета). Для удобства мониторинга базы данных использовался графический интерфейс для PostgreSQL - pgAdmin [11]. Так, на рисунке 3 продемонстрирована работоспособность созданной базы данных. Рисунок 7 - Представление базы данных в pgAdmin 3.2.1 3.2.2 Разработка слоя модели На данном этапе сущностям в SQL-таблицах было разработано отображение в виде POJO-объектов в Java (рисунок 4). Hibernate [12] в свою очередь берёт на себя управление запросами к БД из Java-окружения и контроль согласованности между этими данными. Рисунок 8 - Диаграмма сущностей разработанного приложения Все операции над сущностями для базы данных происходят через класса-посредника - DAO. Каждый из таких классов является бином в контексте Spring, и внедряет в себя зависимость экземпляра entityManager - объекта, который управляет транзакциями в Hibernate. DAO преимущественно используются для выполнения базовых CRUD операций (которые с помощью механизма наследования были вынесены в базовый абстракный класс AbstractDaoImpl и переиспользованы во всех дочерних классов воизбежание дублирования), но так же поддерживают и более сложные запросы с использованием динамических параметров или JOIN благодаря Criteria API. С реализацией DAO на примере класса RentalPointDaoImpl можно ознакомиться на листинге 2. В современных проектах от использования DAO всё чаще отказываются в пользу применения JpaRepository из SpringData, но в образовательных целях использования DAO всё еще уместно. Одним из дальнейших этапов развития проекта так же рассматривается миграция на Spring Data. Центральным местом всего приложения, реализующим основную бизнес-логику, является слой сервисов. Рассмотрим для примера листинг 3, на котором продемонстрированы 4 метода из сервиса поездок. Метод createRideWithTariff создает поездку по тарифу. Внутри этого метода у пользователя берётся текущий тариф, после чего вычисляется цена в минуту для предстоящей поездки. Далее идет проверка на корректность возможностей тарифа и статуса самоката, а затем создается объект поездки, сохраняется в базу данных и начинается 30-секундный отсчёт. Поскольку тарифом предусмотрена только базовая ставка, поверх которой могут накладываться дополнительная наценка, то пользователю предоставлен выбор - принять или не принимать поездку. Если пользователь не успеет принять поездку - она удаляется. За это отвечает метод startCountdownForDeletionOfPendingRidesOfTheUser, который создает отдельный поток в котором запускается таймер, и по истечению отсчета в асинхронном режиме удаляет ожидающую поездку пользователя. При этом всё это время пользователь может продолжать отсылать другие запросы на сервер. Помимо этого, в данном сервисе расположены методы начала и завершения поездки, которые фиксируют временные метки для точного расчета итоговой цены в зависимости от поминутной ставки и целого количества минут, проведённых в пути. Храня персональную информацию пользователей нельзя пройти мимо вопроса защиты данного приложения [13]. Для этих целей был применён модуль Spring Security [14] и библиотека JJWT, генерирующая JWT-токены [15]. На листинге 4 представлен отрывок из класса, предоставляющего и валидирующего токены. В данном приложении не задействован Authorization Server, применяется базовая внутренняя password grant аутентификация, при которой пользователей передает свои логин и пароль, а в ответ получает временный JWT access token, по которому система будет его идентифицировать. 3.2.3 Разработка слоя контроллера Контроллеры - компоненты системы, помеченные аннотацией @RestController и выполняющие связывающую функцию между эндпоинтами API и методами сервисов. В данном приложение представлены восемь контроллеров (рисунок 9). Рисунок 9 -

Диаграмма контроллеров разработанного приложения Шесть из них соответствуют работе с программнымим сущностями - пользователями, точками проката, самокатами, тарифами и подписками, а так же поездками. Оставшиеся два - DefaultController выполняет роль индикатора, свидетельствующего что запуск приложения был успешен, и LoginController - отвечает за аутентификацию пользователей. Все эндпоинты проектировались в соответствие с архитектурным стилем REST. Пример того, как они выглядят представлен на рисунке 10. Рисунок 10 - Отрывок из списка эндпоинтов разработанного API 3.2.4 Разработка слоя представления Поскольку проект представляет RESTful API и не отдает HTML-страницы непосредственно (что позволяет на его основе построить распределенную систему и уменьшает зацепление между компонентами), то можно сказать, что слой представления - это данные в формате JSON, которые получает клиент. Ответ в JSON формате формируется на основе DTO, которые с помощью соответсующих библиотек проходит процесс конвертации из/в POJO-объекты. Со списком разработанных DTO можно ознакомиться на рисунке 11. Рисунок 11 - DTO-классы проекта Обратим внимание на класс RentalPointDto (листинг 5). В нем мы явно указываем ту структуру, которую собираемся передать на клиент. Таким образом, при отравке запроса, ответ будет содержать именно те поля, которые были описаны в DTO. Пример такого запроса представлен на рисунке 12. Листинг 5 - RentalPointDto.class @Setter @Getter public class RentalPointDto { private Long id; private Integer currentNumberOfScooters; private Geolocation geolocation; private Double distanceToClientInKm; private List scooters; } Рисунок 12 - Отображение RentalPointDto в JSON-формате на клиенте 3.2.5 Разработка юнит тестов Важным этапом разработки любого программного продукта является тестирование [16]. Оно обеспечивает сохранность исполнения тех контрактов, которые закладывались в программные модули при их разработке, и в случае изменения каких-либо компонентов системы своей неисправностью сигнализируют о критических нарушениях логике работы программы. На данном этапе развития приложения реализованы только модульные тесты [17], покрывающие главным образом ключевые сценарии работы слоя сервисов. Все они исправны, в чем можно убедиться на рисунке 13. Рисунок 13 - Исправное функционирование тестов приложения 3