Geekbrains

Создание веб-приложения для взаимодействия фотографов и клиентов.

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Гаврилов Валентин Сергеевич

Москва

2025

# Введение

Современный рынок услуг фотографов требует более удобных и эффективных решений для взаимодействия клиентов и исполнителей. Веб-приложение, разработанное с использованием Java, будет служить платформой для предоставления услуг, автоматизации процессов и повышения удобства как для фотографов, так и для клиентов.   
 Веб-технологии в последнее время стали неотъемлемой частью повседневной жизни. Наблюдается стремительный рост интернет-рынка, что обуславливает необходимость создания современных, функциональных и удобных веб-приложений. Количество пользователей интернета постоянно растет, что создаёт огромные возможности и вызовы для разработчиков. В условиях такой динамичной среды ключевым аспектом успешной веб-разработки является использование эффективных инструментов и технологий.  
Java — один из самых распространенных языков программирования, обладающий множеством преимуществ, таких как кроссплатформенность, высокая производительность и широкие возможности для разработки серверных приложений. Широкая экосистема Java включает в себя фреймворки, такие как Spring и Hibernate, которые значительно упрощают процесс разработки, обеспечивают гибкость и дают возможность легко интегрироваться с другими системами. Эти технологии позволяют создавать современные веб-приложения, которые могут обрабатывать большое количество запросов и выполнять сложные бизнес-логики.  
Цель данной дипломной работы заключается в разработке полноценного веб-приложения с использованием языка Java и соответствующих фреймворков. В рамках выполнения работы будут решены следующие задачи:  
- Проектирование архитектуры веб-приложения. В этом разделе будет определена структура приложения, его основные компоненты и совокупность взаимодействий между ними.  
  
- Реализация функционала. В процессе разработки будут использованы подходы, удобные для создания RESTful API, а также интеграция с базами данных с помощью ORM.  
  
 В работе будет рассмотрен вопрос безопасности веб-приложений, который является критически важным в условиях растущих угроз кибербезопасности. Уделим внимание таким аспектам, как аутентификация, авторизация и шифрование данных.  
Таким образом, данная дипломная работа будет полезна как с точки зрения практической реализации, так и с точки зрения углубленного понимания архитектуры и технологий веб-разработки. Ожидается, что результаты исследования смогут служить основой для будущих профессиональных проектов в области веб-разработки.

Весь проект планирую реализовать сам. Клиентскую часть буду реализовывать схематично, в целях тестирования функционала.

Глава 1. Веб-приложение

1.1 Архитектура веб-приложения

Архитектура веб-приложения включает в себя организацию компонентов, их взаимодействие и технологию, используемую для создания системы. Основные слои архитектуры веб-приложения можно разделить на несколько категорий:

1.1.1 Клиентская часть (Frontend)  
Frontend — это клиентская часть веб-приложения, которая отвечает за взаимодействие пользователя с приложением через браузер.

Основные технологии  
- HTML (HyperText Markup Language): Язык разметки, который используется для структуры веб-страниц. Определяет различные элементы, такие как заголовки, параграфы, ссылки, изображения и т.д.  
  
- CSS (Cascading Style Sheets): Используется для оформления HTML-документов. Позволяет изменять шрифты, цвета, отступы, размеры и другие визуальные аспекты элементов.  
  
- JavaScript: Язык программирования, который добавляет интерактивность на веб-страницы. Позволяет реагировать на действия пользователя, такие как клики, наведение мыши и отправка форм.

1.1.2. Серверная часть (Backend)  
Отвечает за обработку логики приложения и управление данными. Основные аспекты серверной части:  
- Язык программирования: Java, Python, Node.js, Ruby и другие.  
- Фреймворки: Spring Boot для Java, Django для Python, Express.js для Node.js и другие.

Spring Boot — это популярный фреймворк для разработки веб-приложений на Java, который упрощает создание, развертывание и управление приложениями Spring. Он создан для того, чтобы минимизировать количество конфигураций и позволить разработчикам сосредоточиться на бизнес-логике, а не на рутинных задачах настройки.  
Основные особенности Spring Boot  
- Автоконфигурация: Spring Boot использует умные настройки для автоматической конфигурации приложения на основе добавленных зависимостей. Это означает, что если вы импортируете библиотеку, например, для работы с базой данных, Spring Boot автоматически настроит соединение с базой данных.  
  
- Организация проекта: Spring Boot создает стандартную структуру проекта, что позволяет легко организовать код. Он использует принципы модульности, инверсии управления и аспектно-ориентированного программирования (AOP).  
  
- Встроенные серверы: Spring Boot позволяет вам запускать приложение как самостоятельное Java-приложение, включая встроенный сервер, такой как Tomcat или Jetty. Это убирает необходимость установки сервера отдельно.  
  
- Легкость в развертывании: Приложения можно упаковывать в исполняемые JAR-файлы, которые можно развертывать на любом сервере с установленной Java.  
  
- Шаблоны проекта: Spring Boot предоставляет множество начальных шаблонов (Starters), которые позволяют быстро подключать необходимые зависимости и библиотеки.  
  
- Зависимости и управление версиями: Spring Boot использует BOM (Bill of Materials) для управления версиями зависимостей, что упрощает обновление библиотек.  
  
Основные аннотации  
- @SpringBootApplication: Основная аннотация для приложения Spring Boot, которая объединяет три другие аннотации: @Configuration, @EnableAutoConfiguration и @ComponentScan.  
  
- @RestController: Аннотация для определения контроллера, который обрабатывает HTTP-запросы и возвращает данные в формате JSON.  
  
- @RequestMapping: Используется для сопоставления HTTP-запросов с методами контроллера. Можно задавать путь, метод (GET, POST и т.д.) и другие параметры.  
  
- @Autowired: Позволяет автоматически внедрять зависимости в классы, что помогает управлять компонентами приложения.  
  
- @Entity: Обозначает сущность JPA, которая будет отображаться на таблицу в базе данных.  
  
Рбота с базой данных  
Spring Boot поддерживает работу с разными базами данных через Spring Data JPA. Можно легко подключаться к реляционным и NoSQL базам данных. Основные шаги включают:  
- Добавление зависимостей: Включите необходимые библиотеки в файл pom.xml (для Maven) или build.gradle (для Gradle).  
  
- Создание сущностей: Определите классы, помеченные аннотацией @Entity, которые будут представлять таблицы в базе данных.  
  
- Создание репозиториев: Создайте интерфейсы, которые расширяют JpaRepository или CrudRepository для выполнения операций CRUD.  
  
- Конфигурация подключения: Настройте параметры подключения к базе данных в [application.properties](https://web.telegram.org/a/application.properties) или application.yml.  
  
Поддержка микросервисов  
Spring Boot часто используется для создания микросервисов. В сочетании с другими проектами Spring (такими как Spring Cloud) он позволяет разрабатывать распределенные системы с помощью подхода микросервисной архитектуры.

- API: RESTful или GraphQL интерфейсы для взаимодействия с клиентом.

1.1.3. База данных

Используется для хранения и управления данными приложения. Может быть:  
- Реляционная: MySQL, PostgreSQL, Oracle и др.  
- NoSQL: MongoDB, Cassandra, Firebase и др.

1.1.4. Архитектурные стили  
Наиболее распространенные архитектурные подходы:  
- Монолитная архитектура: Все компоненты приложения объединены в одном коде и развертываются как единое целое.  
- Микросервисная архитектура: Приложение разбивается на небольшие, независимые сервисы, каждый из которых выполняет определенную функцию.

1.1.5. Протоколы и технологии  
- HTTP/HTTPS: Протоколы передачи данных в Интернете.  
- WebSocket: Протокол для обмена данными в реальном времени.  
- REST/GraphQL: Стандарты для создания API.

1.1.5.1 RESTful API — это интерфейс, который позволяет взаимодействовать с веб-приложениями и сервисами через стандартные HTTP-запросы. REST (Representational State Transfer) представляет собой архитектурный стиль для работы с распределенными системами, такими как веб-приложения. Он призван упростить создание и поддержку сетевых API.  
Основные принципы REST  
- Статус без сохранения состояния: Взаимодействия между клиентом и сервером не содержат информации о предыдущих запросах. Каждое взаимодействие является независимым, и клиент должен отправлять всю необходимую информацию с каждым запросом.  
- Клиент-серверная архитектура: Клиент и сервер разделены, что позволяет каждому из них развиваться независимо. Клиент отвечает за пользовательский интерфейс, а сервер — за логику обработки данных.  
- Унифицированный интерфейс: REST использует стандартные HTTP-методы (GET, POST, PUT, DELETE) для выполнения операций над ресурсами:  
- GET: Получение данных от сервера.  
- POST: Создание нового ресурса.  
- PUT: Обновление существующего ресурса.  
- DELETE: Удаление ресурса.  
  
- Идентификация ресурсов: Ресурсы (например, пользователи, статьи) идентифицируются с помощью уникальных URI (Uniform Resource Identifier).

- База данных:  
- Реляционные базы данных (например, PostgreSQL или MySQL), что позволит эффективно хранить и управлять данными о пользователях, фотографиях и заказах.

1.1.6. Безопасность  
Включает механизмы аутентификации (например, OAuth, JWT) и защиты от атак (например, CSRF, XSS).

Глава 2. Проектирование веб-приложения

При проектировании веб-приложения важно учитывать архитектуру клиент-сервер. В нашем случае:  
- Клиентская часть: в рамках дипломной работы не будет проводиться разработка интерфейса пользователя, функционал будет представляться в виде html странички с основным функционалом и полями, предназначен будет только для тестирования.  
  
- Серверная часть:  
- Реализация на Java с использованием Spring Boot для создания RESTful API.  
- Важно использовать MVC (Model-View-Controller) паттерн для разделения логики приложения.

Основные функции приложения  
Приложение должно включать следующие ключевые функции:  
- Регистрация и аутентификация: Удобный процесс создания учетных записей для клиентов и фотографов с использованием OAuth2 или JWT.  
  
- Профили пользователей: Возможность фотографов загружать свой портфолио, устанавливать расценки и описания услуг, вести свой календарь фотосессий, доступный всем пользователям, для планирования даты съемок(при заказе фотосессии).  
  
- Поиск и фильтрация: Функциональность, позволяющая клиентам находить фотографов по различным критериям (стиль, цена, местоположение, дата съемки).  
  
- Создание заказов: Клиенты могут оформлять заказы с указанием всех деталей, а фотографы – подтверждать или отклонять их.  
  
- Отзывы и рейтинги: Система обратной связи, позволяющая клиентам оставлять отзывы о работе фотографов, тем самым формируя их репутацию.  
  
Безопасность веб-приложения  
Безопасность является критически важной частью любого веб-приложения. Вам следует учесть:  
- Аутентификация: Реализация безопасной аутентификации с помощью JWT и шифрование паролей (например, с использованием bcrypt).  
  
- Шифрование данных: Обеспечение передачи данных по HTTPS для защиты информации пользователей.  
  
- Защита от атак: Введение дополнительных мер для защиты от SQL-инъекций, XSS и CSRF атак.  
  
Заключение  
Разработка веб-приложения для взаимодействия фотографов и клиентов на Java имеет значительный потенциал для улучшения качества обслуживаемых клиентов и расширения возможностей фотографов. Этот проект не только решает актуальные проблемы, присутствующие на рынке, но и предлагает интуитивно понятные решения для конечных пользователей, тем самым способствуя развитию бизнес-экосистемы в сфере фотографии.  
Эта теоретическая часть предоставляет основу для дальнейшей реализации проекта и подчеркивает его значимость в современных условиях.