Geekbrains

**Разработка веб-приложения визитной карточки фотографа с системой онлайн-записи и отзывов**

Программа: Java-разработчик

Специалист

Сулагаев Дмитрий Викторович

г.Ульяновск

2025 год

# Содержание

Введение …………………………………………….…………………………….…1

Глава 1. Основы разработки веб-приложений …..………………………………...4

1.1. Что такое веб-приложение, его архитектура и особенности ………………..4

1.2. Основные этапы разработки веб-приложения: от идеи до развертывания…7

1.3. Жизненный цикл разработки: проектирование, кодирование, тестирование и поддержка…………………………………………………………………………10

1.4. Источники требований: что такое техническое задание и как разрабатывать веб-приложение при его отсутствии………………………………………………13

Глава 2. Подготовка к разработке веб-приложения ……………………………..17

2.1. Проектирование структуры приложения…………………………………….17

2.2. Разработка технического задания и спецификаций…………………………22

2.3. Выбор инструментов разработки………….………………...………………..26

2.4. Версионный контроль в разработке ....……….………………….……..…….29

Глава 3. Разработка веб-приложения визитной карточки фотографа с системой

онлайн записи ...……………………………………………………………………30

3.1. Подготовка и настройка проекта...…………………………………………...30

3.2. Проектирование и реализация удобного пользовательского интерфейса...32

3.3. Обеспечение безопасности веб-приложения и защита пользовательских данных………………………………………………………………………………35

3.4. Интеграция базы данных и работа с серверной логикой…………………...40

3.5. Разработка функциональности: система онлайн-записи и модуль отзывов……………………………………………………………………………...47  
3.6. Оптимизация производительности и обеспечение стабильности работы……………………………………………………………………………….52  
3.7. Подведение итогов…………………………………………………………….54

Заключение…………………………………………………………………………56

Список используемой литературы………………………………………………..58

Приложения………………………………………………………………………...59

# Введение

Современные технологии оказывают значительное влияние на все сферы деятельности, в том числе и на профессиональную деятельность фотографов. В условиях высокой конкуренции и стремительного развития цифровых решений фотографам необходимо не только владеть мастерством съемки, но и обеспечивать удобный и оперативный доступ клиентов к своим услугам. В связи с этим разработка веб-приложения - визитной карточки фотографа с системой онлайн-записи и отзывов является актуальной задачей.

Проект представляет собой веб-приложение, которое позволит фотографам эффективно презентовать свои услуги, организовывать взаимодействие с клиентами и управлять своим расписанием. Пользователи смогут ознакомиться с портфолио фотографа, примерами работ, стоимостью услуг, отправлять сообщения фотографу без регистрации, а также записаться на фотосессию онлайн, оставлять отзывы и просматривать мнения других клиентов. Это не только упрощает процесс взаимодействия между фотографом и клиентами, но и способствует улучшению качества обслуживания.

Актуальность темы обусловлена необходимостью цифровизации бизнеса в сфере фотоуслуг. В настоящее время многие фотографы сталкиваются с проблемой недостатка удобных инструментов для управления заказами, что ведет к потере потенциальных клиентов. Веб-приложение позволит решить эту проблему, предоставляя централизованную платформу для организации работы фотографа.

Основной целью проекта является разработка удобного, функционального и интуитивно понятного веб-приложения, которое обеспечит фотографам эффективный инструмент для продвижения и взаимодействия с клиентами. Для достижения этой цели планируется выполнение следующих задач:

* анализ существующих решений и выявление их преимуществ и недостатков;
* проектирование структуры и функционала веб-приложения;
* разработка пользовательского интерфейса;
* реализация системы онлайн-записи и отзывов;
* тестирование и отладка приложения;
* развертывание и финальная презентация готового продукта.

Данный проект направлен на решение проблемы автоматизации и оптимизации работы фотографов, что позволит им сосредоточиться на творческой деятельности, минимизируя временные затраты на административные задачи. Кроме того, приложение поможет потенциальным клиентам быстро находить нужного специалиста и взаимодействовать с ним в удобном формате.

Дипломный проект выполняется в рамках специализации веб-разработки. Опыт, полученный в процессе изучения программирования и разработки веб-приложений, является полезным для успешного выполнения данного проекта. В частности, ранее были реализованы проекты, связанные с созданием сайтов и веб-сервисов, что дало понимание принципов работы с фронтенд- и бэкенд-технологиями.

Для реализации проекта будут использоваться следующие инструменты:

* среда разработки IntelliJ IDEA 2024.1.4 (Ultimate Edition);
* системы контроля версий (Git/GitHub);
* инструменты для тестирования и отладки (Postman, браузерные DevTools);
* графические редакторы для оформления значков, картинок и фотографий (Paint, Adobe Photoshop).

В процессе разработки планируется использовать следующие технологии:

* HTML, CSS, JavaScript для создания пользовательского интерфейса;
* Thymeleaf шаблонизатор для динамического взаимодействия с пользователем;
* Spring Boot для серверной части приложения;
* Spring Security обеспечения безопасности веб-приложения, для аутентификации и авторизации пользователей сервиса;
* Spring Data JPA, H2 для удобного взаимодействия с базой данных, для хранения информации о клиентах, свободных днях, записях, отзывах и сообщений от клиентов.

Результатом выполнения проекта станет функциональное веб-приложение, способное облегчить процесс взаимодействия фотографа с клиентами и повысить уровень сервиса в данной сфере.

Проект выполнен в рамках индивидуальной работы. Вся разработка, включая проектирование, дизайн, программирование, тестирование и развертывание, осуществлена одним разработчиком.

Роли в проекте:

- Разработчик и архитектор – разработка структуры веб-приложения, выбор технологий, написание кода.

- Дизайнер – создание пользовательского интерфейса и проработка UX.

- Тестировщик – тестирование функционала, исправление багов.

- Менеджер проекта – планирование этапов разработки и контроль выполнения задач.

**Глава 1. Основы разработки веб-приложений**

**1.1. Что такое веб-приложение, его архитектура и особенности**

Веб-приложение – это клиент-серверное программное обеспечение, работающее через веб-браузер и использующее Интернет для передачи данных. Оно позволяет пользователям выполнять различные операции, такие как регистрация, авторизация, отправка данных, просмотр контента и взаимодействие с другими пользователями.

Ключевые характеристики веб-приложений:

1. **Доступность** – веб-приложения работают на различных устройствах (ПК, планшетах, смартфонах) без необходимости установки.
2. **Кроссплатформенность** – благодаря использованию браузеров, веб-приложения поддерживаются разными операционными системами (Windows, macOS, Linux, Android, iOS).
3. **Обновляемость** – обновления и исправления внедряются на сервере, поэтому пользователи автоматически получают актуальные версии без необходимости скачивания обновлений.
4. **Интерактивность** – современные веб-приложения позволяют пользователям взаимодействовать с интерфейсом так же, как и с настольными программами.
5. **Подключение к базе данных** – большинство веб-приложений взаимодействуют с сервером базы данных, что позволяет хранить и обрабатывать большие объёмы информации.

### Архитектура веб-приложений

Архитектура веб-приложения определяет его структуру, компоненты и способы взаимодействия между ними. Различают несколько основных типов архитектуры веб-приложений:

1. **Клиент-серверная архитектура** – базовая модель, в которой клиент (браузер) отправляет запросы серверу, а сервер обрабатывает их и отправляет ответы. Веб-приложение может состоять из:
   * **Клиентской части (Frontend)** – отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем. Обычно реализуется с использованием HTML, CSS, JavaScript и различных фреймворков.
   * **Серверной части (Backend)** – обрабатывает запросы клиентов, взаимодействует с базой данных, выполняет логику приложения. Реализуется с помощью языков программирования, таких как Java, Python, PHP, Node.js и других.
   * **Базы данных** – хранит информацию и обеспечивает доступ к данным для серверной части. Популярные системы управления базами данных (СУБД) включают MySQL, MongoDB, PostgreSQL.
2. **Одностраничные приложения (SPA – Single Page Application)** – архитектура, в которой большая часть обработки данных и рендеринга происходит на стороне клиента. Сервер отвечает только за передачу данных в формате JSON или XML. Это позволяет добиться высокой производительности и плавного пользовательского опыта. Примеры технологий для разработки SPA – React, Vue.js, Angular.
3. **Микросервисная архитектура** – структура, в которой веб-приложение разделено на небольшие независимые сервисы, каждый из которых отвечает за отдельную функциональность. Такой подход улучшает масштабируемость и упрощает поддержку.
4. **Прогрессивные веб-приложения (PWA – Progressive Web Application)** – гибридный подход, сочетающий возможности веб-приложений и нативных мобильных приложений. PWA могут работать в офлайн-режиме, отправлять push-уведомления и загружаться как мобильные приложения.

### Особенности веб-приложений

Веб-приложения обладают рядом особенностей, которые отличают их от традиционных программных решений:

1. **Гибкость и масштабируемость** – возможность адаптации под различные экраны и устройства, поддержка масштабируемости для увеличения нагрузки пользователей.
2. **Высокий уровень безопасности** – веб-приложения требуют защиты от атак (SQL-инъекций, межсайтового скриптинга XSS, CSRF-атак и других угроз).
3. **Сетевые зависимости** – работа веб-приложений зависит от качества интернет-соединения, что делает важным внедрение механизмов кэширования и офлайн-доступа.
4. **API и интеграции** – современные веб-приложения активно используют API (например, RESTful API) для связи с внешними сервисами и базами данных.

Веб-приложения играют ключевую роль в цифровом мире, обеспечивая удобный доступ к различным сервисам и функционалу через браузер. Они обладают высокой доступностью, кроссплатформенностью и возможностью обновления без необходимости установки. Важно учитывать их архитектуру, особенности и современные технологии при разработке, чтобы создать качественный и безопасный продукт. С развитием веб-технологий веб-приложения продолжают эволюционировать, предоставляя пользователям ещё больше возможностей и удобства.

**1.2. Основные этапы разработки веб-приложения: от идеи до развертывания**

Разработка начинается с определения цели веб-приложения и его основной концепции. На этом этапе проводится:

* Определение целевой аудитории и изучение её потребностей.
* Анализ рынка и конкурентных решений.
* Формулирование ключевых задач и функционала приложения.
* Создание технического задания (ТЗ), в котором фиксируются основные требования к проекту.

Этот этап критически важен, так как он закладывает фундамент для дальнейшей работы над приложением.

### Проектирование структуры и архитектуры

После определения требований начинается этап проектирования, который включает:

* Разработку архитектуры веб-приложения (клиент-серверная, микросервисная, SPA и т. д.).
* Проектирование базы данных и структуры хранения данных.
* Создание схем взаимодействия компонентов системы (API, серверная и клиентская часть).
* Подготовку прототипов пользовательского интерфейса (UI/UX-дизайн), что помогает визуализировать основные элементы приложения и их взаимодействие.

### Выбор технологий и инструментов

На основе анализа требований и архитектурного проектирования выбираются технологии для реализации веб-приложения. Они могут включать:

* **Фронтенд**: HTML, CSS, JavaScript, фреймворки (React, Vue.js, Angular).
* **Бэкенд**: Java, Python (Django, Flask), PHP (Laravel), Ruby on Rails.
* **База данных**: MySQL, PostgreSQL, MongoDB, Firebase.
* **Средства контроля версий**: Git (GitHub, GitLab, Bitbucket).
* **Средства тестирования и CI/CD**: Jest, Selenium, Docker, Jenkins.

Правильный выбор технологий влияет на производительность, безопасность и масштабируемость будущего приложения.

### Разработка веб-приложения

Основной этап — программирование, которое делится на две ключевые части:

* **Фронтенд-разработка** — создание интерфейса пользователя, работа с клиентской логикой, интеграция с серверной частью через API.
* **Бэкенд-разработка** — реализация бизнес-логики, настройка базы данных, управление аутентификацией и авторизацией пользователей.

Важной практикой на этом этапе является использование методологий Agile или Scrum, которые позволяют разбивать работу на небольшие итерации (спринты) и адаптироваться к изменениям требований.

### Тестирование и отладка

Перед развертыванием проводится тестирование веб-приложения, которое включает:

* **Функциональное тестирование** — проверка соответствия требованиям и корректной работы всех функций.
* **Тестирование безопасности** — выявление уязвимостей и защита данных пользователей.
* **Тестирование производительности** — проверка нагрузки на сервер и оптимизация кода.
* **Кросс-браузерное тестирование** — проверка отображения интерфейса в разных браузерах и устройствах.

Использование автоматизированного тестирования позволяет ускорить процесс выявления ошибок и повысить стабильность приложения.

### Развертывание и публикация

После успешного тестирования начинается процесс развертывания веб-приложения. Этот этап включает:

* Настройку серверного окружения и баз данных.
* Размещение приложения на хостинге или облачном сервисе (AWS, Heroku, DigitalOcean, Firebase).
* Настройку CI/CD для автоматического обновления и развертывания новой версии.
* Мониторинг состояния сервера и логирование работы приложения.

### Поддержка и развитие

Разработка веб-приложения не заканчивается на этапе развертывания. Важно обеспечивать его стабильную работу, анализировать поведение пользователей и вносить улучшения. Этот этап включает:

* Мониторинг ошибок и логирование событий.
* Сбор отзывов пользователей и их анализ.
* Разработку новых функций и оптимизацию существующих модулей.

Разработка веб-приложения — это сложный процесс, требующий грамотного подхода на каждом этапе. От формирования идеи до поддержки и развития, каждое звено играет важную роль в создании качественного и успешного продукта. Применение современных технологий, инструментов и методологий позволяет эффективно организовать процесс разработки, обеспечивая высокую производительность, безопасность и удобство использования веб-приложения.

# 1.3. Жизненный цикл разработки: проектирование, кодирование, тестирование и поддержка

Жизненный цикл разработки программного обеспечения представляет собой последовательность этапов, которые проходят при создании веб-приложения. Каждый этап играет важную роль в обеспечении качества, функциональности и надёжности программного продукта. Рассмотрим основные фазы жизненного цикла разработки: проектирование, кодирование, тестирование и поддержка.

### Проектирование

Проектирование — это первый и один из самых важных этапов разработки, который определяет структуру будущего веб-приложения. Основные задачи на этом этапе:

* Определение требований к приложению.
* Разработка архитектуры системы.
* Проектирование базы данных.
* Создание прототипов пользовательского интерфейса (UI/UX).

Правильное проектирование помогает избежать множества ошибок на последующих этапах и закладывает прочную основу для разработки.

### Кодирование

После завершения проектирования начинается процесс написания кода. Этот этап включает:

* **Фронтенд-разработку** – создание пользовательского интерфейса с использованием HTML, CSS, JavaScript и различных фреймворков (React, Vue.js, Angular).
* **Бэкенд-разработку** – реализация серверной логики, обработка запросов, управление базами данных. Популярные технологии: Java(Spring Framework), Python (Django, Flask), PHP (Laravel), Ruby on Rails.
* **Интеграцию API** – подключение сторонних сервисов и внутренних API для расширения функционала веб-приложения.

Важно применять современные методологии разработки, такие как Agile и DevOps, для повышения эффективности работы команды и ускорения процесса создания продукта.

### Тестирование

Перед развертыванием веб-приложения необходимо провести его тестирование, чтобы выявить ошибки и убедиться в правильности работы системы. Основные виды тестирования:

* **Функциональное тестирование** – проверка соответствия работы приложения требованиям.
* **Тестирование безопасности** – выявление уязвимостей и защита данных пользователей.
* **Тестирование производительности** – проверка скорости работы и устойчивости к нагрузкам.
* **Кросс-браузерное тестирование** – проверка отображения интерфейса в различных браузерах и на разных устройствах.

Использование автоматизированного тестирования позволяет значительно ускорить процесс выявления и исправления ошибок.

### Поддержка и развитие

После успешного развертывания веб-приложения начинается этап поддержки и дальнейшего развития. Этот процесс включает:

* Мониторинг производительности и логирование ошибок.
* Сбор отзывов пользователей и анализ метрик использования приложения.
* Разработка новых функций и улучшение существующих возможностей.
* Обновление системы безопасности и устранение найденных уязвимостей.

Поддержка и регулярное обновление веб-приложения позволяют обеспечивать его актуальность, стабильность и безопасность в долгосрочной перспективе.

Жизненный цикл разработки веб-приложения включает несколько важных этапов: проектирование, кодирование, тестирование и поддержку. Каждый из них играет ключевую роль в создании качественного и надёжного программного продукта. Грамотный подход к каждому этапу разработки, использование современных технологий и методологий помогают создавать эффективные и востребованные веб-приложения, соответствующие требованиям пользователей и бизнеса.

**1.4. Источники требований: что такое техническое задание и как разрабатывать веб-приложение при его отсутствии**

Разработка веб-приложений требует четкого понимания и формулировки требований, которые будут лежать в основе работы системы. Важно понять, какие задачи веб-приложение должно решать, каковы его функциональные и нефункциональные требования. Одним из важнейших документов, определяющих требования к проекту, является техническое задание (ТЗ). Однако в практике разработки иногда встречаются ситуации, когда ТЗ отсутствует или его создание затруднено. В таких случаях разработчики сталкиваются с необходимостью гибко подходить к процессу разработки, разрабатывая систему по мере работы, опираясь на понимание нужд клиента и пользователя.

### Техническое задание: что это и зачем оно нужно

Техническое задание — это документ, который служит основой для разработки системы. Он описывает требования к функциональности, интерфейсу, безопасности, производительности и другим аспектам веб-приложения. Задача ТЗ — дать всем участникам проекта (заказчикам, разработчикам, тестировщикам и дизайнерам) четкое представление о том, что должно быть разработано, какие функции необходимо реализовать, и какие ограничения нужно учесть.

ТЗ является важным элементом успешной реализации проекта, поскольку:

1. **Четко определяет требования**. В ТЗ описаны все функции, которые должен выполнять продукт, а также ожидаемые результаты.
2. **Управляет ожиданиями**. В нем фиксируются договоренности с клиентом относительно сроков, бюджета и конечного продукта.
3. **Используется как руководство для команды разработки**. Разработчики, дизайнеры и тестировщики могут опираться на него для того, чтобы двигаться в одном направлении и не отклоняться от курса.
4. **Предотвращает недоразумения**. ТЗ помогает снизить риск возникновения конфликтов между заказчиком и исполнителями, так как оно служит договором о том, что будет сделано.

### Проблемы отсутствия технического задания

При отсутствии ТЗ проект может столкнуться с рядом проблем. Недостаток четких требований может привести к недопониманию между заказчиком и командой разработчиков. Разработчики могут потратить много времени на создание функционала, который в итоге не удовлетворяет потребности заказчика. Внешний вид и взаимодействие с пользователем могут быть не такими, как было задумано, а функциональность — не соответствовать ожиданиям.

В таких ситуациях проект может столкнуться с риском:

* **Невыполнения требований заказчика**.
* **Излишних затрат** на доработку и переписывание функционала.
* **Сложностей с контролем качества**.
* **Долгими сроками реализации** из-за неопределенности на старте.

### Разработка веб-приложения без технического задания

Когда техническое задание отсутствует, важно выстроить процесс разработки таким образом, чтобы максимально снизить риски и обеспечить выполнение всех ключевых требований.

1. **Постоянное общение с клиентом**. При отсутствии ТЗ особенно важно регулярное взаимодействие с заказчиком. Это может быть в виде встреч, звонков, обсуждений и тестирования прототипов. Регулярные отзывы от заказчика позволяют корректировать проект в процессе его создания, выявляя несоответствия и уточняя требования.
2. **Использование гибких методологий разработки**. В таких случаях хорошей практикой является применение гибких методологий, таких как Agile или Scrum. Эти подходы ориентированы на выполнение работы малыми итерациями, с регулярными пересмотрами требований и фокусом на потребности заказчика. С помощью таких методологий можно быстрее реагировать на изменения и дополнения в проекте.
3. **Создание прототипа или MVP**. При отсутствии полноценного ТЗ можно начать с разработки минимально жизнеспособного продукта (MVP). Это позволит на ранних стадиях увидеть, как веб-приложение будет выглядеть и работать, а также тестировать его основные функции. Прототип также может служить наглядным инструментом для обсуждения требований с заказчиком.
4. **Анализ потребностей целевой аудитории**. Для того чтобы избежать значительных переделок в дальнейшем, необходимо заранее проанализировать целевую аудиторию веб-приложения. Понимание того, какие проблемы должны решаться пользователями (в данном случае — клиентами фотографа), помогает выделить самые важные функции и сосредоточиться на них в первую очередь.
5. **Разработка пользовательских историй**. В случаях, когда нет ТЗ, вместо его стандартных пунктов можно использовать концепцию пользовательских историй (user stories). Каждая история представляет собой описание того, что пользователь должен быть в состоянии делать с веб-приложением. Это помогает сформулировать требования, которые будут четко понятны как заказчику, так и разработчику.
6. **Итеративная разработка**. Важно разбить проект на небольшие части и разрабатывать их поэтапно, начиная с самых приоритетных функций. Таким образом, в процессе реализации можно будет уточнять требования и адаптировать проект под изменяющиеся условия.

Разработка веб-приложений без технического задания — сложная задача, требующая гибкости, четкого взаимодействия с заказчиком и использования адаптивных методологий. Важно помнить, что при отсутствии ТЗ задача разработки не теряет своей значимости, но необходимо тщательно прорабатывать каждый этап работы, минимизируя риски и внимательно следя за изменяющимися требованиями. Основной фокус следует делать на итеративности, обратной связи и постоянной проверке получаемых результатов.

**Глава 2. Подготовка к разработке веб-приложения**

**2.1. Проектирование структуры приложения**

Процесс проектирования структуры веб-приложения является одним из ключевых этапов в разработке системы. Он включает в себя определение логической организации компонентов приложения, а также их взаимодействие. В данном проекте целью является создание веб-приложения визитной карточки для фотографа с функциональностью онлайн-записи и системой отзывов. Для обеспечения удобства пользователей и надежности работы системы, необходимо тщательно продумать как структуру интерфейса, так и архитектуру приложения в целом.

### Основные цели проектирования

При проектировании структуры приложения важно, прежде всего, определить основные цели и задачи, которые должны быть решены. В данном случае, приложение должно предоставить пользователю простую и интуитивно понятную навигацию, где каждый элемент системы имеет свое четкое назначение.

Основной задачей является создание удобного интерфейса для записи на услуги фотографа, отображения актуальных отзывов и предоставления информации о фотографе. Важной частью проектирования является и защита данных пользователей, поскольку будет храниться информация о клиентах, их персональные данные, подробные данные о записях, информация о днях работы фотографа, публикуемая самим фотографом, информация о свободных/занятых интервалах времени, а также отзывы.

### Разделение приложения на функциональные блоки

Для удобства разработки и дальнейшего сопровождения веб-приложения, оно будет разделено на несколько логических блоков, каждый из которых будет отвечать за выполнение конкретной задачи. Основные блоки приложения включают:

#### Главная страница приложения

Главная страница является первым элементом, с которым пользователь сталкивается при входе в приложение. Именно она должна заинтересовать пользователя и побудить его воспользоваться услугами фотографа. На этой странице будет представлена информация о фотографе, а также шапка (блок в верхней части страницы сайта, который виден на всех страницах сайта) с размещенными на ней навигационными элементами, которые будут включать разделы с примерами работ, стоимостью и описанием услуг, контактные данные и формы связи, а также блок для регистрации и авторизации в системе.

Особое внимание будет уделено простоте и легкости восприятия страницы, так как важно, чтобы пользователь не терялся в интерфейсе и мог легко найти нужную информацию.

#### Система онлайн-записи

Одним из ключевых функциональных элементов приложения является система онлайн-записи на съемки. Этот компонент позволит пользователю выбрать удобное время и дату для записи на услугу, заполнить форму с предпочтениями (например, тип съемки, локацию и т.п.), а также оставить дополнительные комментарии.

Для реализации системы записи будет использоваться таблица со свободными днями по убыванию, чтобы пользователь мог видеть доступные временные слоты и выбирать свободные даты. Важно, чтобы интерфейс был максимально понятным и не требовал от пользователя дополнительных усилий для выбора времени. В целях безопасности, доступ к онлайн записи будет открыт только для авторизованных в системе пользователей. После авторизации будут доступны дополнительные элементы навигации, такие как информация о текущей записи, статусе, который указывает, подтвердил ли фотограф запись, а также историю заказов и персональные данные с указанием уровня скидки в зависимости от активности пользователя.

#### Система отзывов и рейтингов

Отзывы играют важную роль в принятии решения о сотрудничестве с фотографом. Веб-приложение будет включать функционал, позволяющий пользователям оставлять свои отзывы о проделанной работе, а также ставить рейтинг. Это даст будущим клиентам возможность узнать мнение других людей, а фотографу — получать обратную связь для улучшения качества услуг.

Система отзывов будет включать фильтрацию по дате и средний рейтинг, что позволит пользователям быстрее найти актуальные и наиболее полезные отклики. Возможность оставить отзыв будет реализована только для авторизованных пользователей, неавторизованные смогут лишь ознакомиться с другими отзывами и средним рейтингом фотографа.

#### Административная панель

Для эффективного управления приложением предусмотрена административная панель, которая позволит фотографу контролировать все аспекты работы с пользователями. Для доступа к административной панели необходимо будет авторизоваться при помощи логина и пароля администратора. Через панель будет возможно просматривать и редактировать информацию о заказах, подтверждать или отменять новые записи, завершать записи после успешного оказания услуги, а также управлять просматривать сообщения, оставленные через форму обратной связи.

В административной панели также будут собраны статистические данные об общем количестве заказов, о заказах в текущем и предыдущем месяце. Это поможет фотографу анализировать работу и принимать решения по улучшению функциональности приложения и качества предоставляемых услуг.

#### Авторизация и регистрация пользователей

Для того чтобы предоставить персонализированный опыт использования приложения, предусмотрена система регистрации и авторизации пользователей. Зарегистрированные пользователи смогут сохранять свои данные для быстрого оформления записи, а также отслеживать историю заказов и статус текущей записи.

Авторизация будет обеспечена через стандартные формы ввода логина и пароля. В дальнейшем может быть добавлена поддержка авторизации через социальные сети, что упростит процесс для пользователей.

### Технологические аспекты проектирования

Проектирование структуры приложения включает в себя не только определение функциональных блоков, но и выбор технологий для их реализации. Важно, чтобы выбранные технологии обеспечивали высокую производительность, безопасность и простоту в обслуживании приложения.

В рамках данного проекта будет использована архитектура с разделением на клиентскую и серверную части. Серверная часть будет отвечать за обработку запросов и управление данными, а клиентская — за отображение информации и взаимодействие с пользователем. Для создания интерфейса будет применяться HTML, CSS и JavaScript, а серверная логика будет реализована с использованием современных технологий, таких как Java и фреймворк Spring Boot с использованием шаблонизатора Thymeleaf.

Для обеспечения безопасности данных пользователей будут использоваться методы шифрования паролей, а также механизмы защиты от SQL-инъекций и других атак. Высокий уровень безопасности будет достигаться за счет использования фреймворка Spring Security.

Проектирование структуры веб-приложения — это многоэтапный процесс, который требует внимательности к каждой детали. Важно, чтобы приложение было не только функциональным, но и удобным в использовании, а также надежным в плане безопасности данных. Разделение приложения на логические блоки и четкое определение их задач поможет обеспечить эффективность работы системы и удовлетворение потребностей пользователей.

**2.2. Разработка технического задания и спецификаций**

#### Введение в процесс разработки технического задания

Техническое задание (ТЗ) является основой для разработки любого веб-приложения, так как в нем описаны все требования, задачи и условия, которые должны быть выполнены в процессе создания системы. Для разработки веб-приложения визитной карточки фотографа с системой онлайн-записи и отзывов важно составить четкое и подробное ТЗ, которое будет направлять разработчиков на всех этапах работы. Важно, чтобы ТЗ было максимально понятным, чтобы избежать недоразумений и ошибок в процессе разработки.

#### Основные цели и задачи

Основной целью разработки веб-приложения является создание удобной и функциональной платформы, которая позволит потенциальным клиентам фотографа легко находить информацию о фотографе, его стоимости услуг, записываться на фотосессии и оставлять отзывы. Задачи, которые необходимо решить при разработке ТЗ для такого веб-приложения:

* Разработка структуры сайта и его функциональных возможностей.
* Создание системы онлайн-записи с выбором даты, времени и пожеланиями.
* Интеграция системы отзывов для клиентов, чтобы они могли делиться опытом с другими пользователями.
* Обеспечение безопасности и конфиденциальности данных пользователей.

#### Требования к функционалу

Важной частью технического задания является описание функциональных требований к веб-приложению. Для этого можно выделить несколько ключевых блоков:

**1.Главная страница:**

* Отображение информации о фотографе, его услугах, ценах и примерах работ.
* Возможность ознакомиться с отзывами, а также связь с фотографом через формы обратной связи.

**2. Система онлайн-записи:**

* Таблица с возможностью выбора доступных дат и времени.
* Данные клиента заполняются автоматически исходя из данных при регистрации (имя, контактная информация, комментарии).
* Отображение статуса записи.

**3. Система отзывов:**

* Оставлять отзывы могут зарегистрированные пользователи.
* Возможность рейтинга (например, от 1 до 5 звезд).
* Подсчет среднего рейтинга на основании всех оценок

**6. Безопасность:**

* Шифрование пароля пользователей.
* Защита от несанкционированного доступа и использование современных технологий безопасности для обработки личных данных.

#### Технические требования

В этой части необходимо указать требования к использованным технологиям и платформам для разработки веб-приложения. Рекомендуемые компоненты и подходы:

* **Языки программирования:** HTML, CSS, JavaScript для фронтенда; Java, PHP или Node.js для бэкенда.
* **Фреймворки:** Thymeleaf для создания динамичного интерфейса; Spring Framework для сервера.
* **База данных:** H2 Database, PostgreSQL или MongoDB для хранения данных пользователей, записей и отзывов.
* **Хостинг:** Облачные решения, такие как AWS или DigitalOcean, для обеспечения надежности и масштабируемости.
* **Интеграция с платежными системами:** если предполагается возможность оплаты через сайт, то необходима интеграция с платежными шлюзами.

#### Спецификации интерфейса

Спецификации интерфейса определяют, как будет выглядеть и работать каждый элемент на сайте. Это важная часть технического задания, так как она описывает взаимодействие пользователей с системой. Спецификации интерфейса могут включать:

* Описание главных экранов сайта с их элементами (меню, кнопки, формы).
* Процесс регистрации пользователя и входа в систему.
* Процесс оформления онлайн-записи, с пошаговыми инструкциями.
* Отображение отзывов и рейтингов на странице каждого вида услуги.

#### Оценка рисков и ограничений

Каждое техническое задание должно включать оценку потенциальных рисков и ограничений, с которыми могут столкнуться разработчики. В данном случае к рискам можно отнести:

* **Неопределенность в требованиях пользователей.** Потребности клиентов могут изменяться, что потребует дополнительных доработок и улучшений.
* **Сложности с интеграцией системы онлайн-записи.** Возможно, потребуется использовать сторонние сервисы или API для синхронизации с календарями, что может увеличить сложность.
* **Безопасность данных.** Важно обеспечить надежную защиту данных клиентов, что потребует использования современных методов шифрования и соблюдения стандартов безопасности.

Разработка технического задания и спецификаций для веб-приложения визитной карточки фотографа с системой онлайн-записи и отзывов является ключевым этапом в процессе его создания. Тщательно составленное ТЗ позволяет избежать множества проблем в процессе разработки, а также помогает разработчикам и заказчикам на всех этапах работы по проекту. Спецификации интерфейса, функциональные и технические требования обеспечивают создание высококачественного, удобного и безопасного веб-приложения, которое будет отвечать потребностям пользователей и удовлетворять цели фотографа.

### 2.3. Выбор инструментов разработки

Выбор инструментов разработки играет ключевую роль в создании качественного, производительного и удобного веб-приложения. Подбор технологий зависит от функциональных требований, удобства в разработке и последующей поддержки, а также от совместимости с современными веб-стандартами.

Для реализации веб-приложения визитной карточки фотографа с системой онлайн-записи и отзывов необходимо определить инструменты для фронтенд- и бэкенд-разработки, базу данных и вспомогательные технологии.

#### Выбор технологий для фронтенда

Фронтенд – это клиентская часть веб-приложения, отвечающая за интерфейс и взаимодействие пользователя с системой. Он должен быть удобным, адаптивным и интуитивно понятным.

Для реализации фронтенда выбраны следующие технологии:

* **HTML5 и CSS3** – основные технологии для разметки и стилизации страниц. Они обеспечивают кроссбраузерную совместимость и адаптивность.
* **JavaScript (ES6+)** – язык программирования, необходимый для создания интерактивных элементов.
* **Thymeleaf** — современный серверный механизм Java-шаблонов для веб- и автономных сред, способный обрабатывать HTML, XML, JavaScript, CSS и даже простой текст.

 Шаблоны Thymeleaf удобны тем, что при простом открытии в браузере они выглядят как обычные HTML-страницы и их можно использовать как статический прототип приложения.

#### Выбор технологий для бэкенда

Бэкенд отвечает за обработку запросов пользователей, управление базой данных и бизнес-логику веб-приложения. Он должен быть надежным, безопасным и легко масштабируемым.

Для реализации серверной части выбран Spring Framevork, включающий в себя следующие технологии:

* **Spring MVC** – это часть фреймворка Spring, предназначенная для создания веб-приложений по паттерну **MVC**.
* **Spring**  **Security** - это мощный модуль Spring для аутентификации и авторизации, предусматривающий цепочку фильтров для защиты от атак (CSRF, XSS, SQL-инъекции), поддержку различных методов аутентификации, простую интеграцию с ролевыми моделями

Выбор Spring Framework обусловлен его высокой производительностью, возможностью быстрой разработки REST API и удобной интеграцией с фронтендом.

#### Выбор базы данных

Для хранения информации о пользователях, записях, свободных днях, сообщениях и отзывах необходима надежная база данных. В данном проекте используется:

* **Spring Data JPA** – это удобная библиотека для работы с базами данных с использованием **JPA (Java Persistence API)**. Она упрощает работу с SQL-запросами и ORM (Object-Relational Mapping).
* **H2 Database** — это компактная реляционная база данных, разработанная на Java. В Spring-приложениях она широко применяется для тестирования и разработки, так как проста в настройке и поддерживает режим работы в оперативной памяти (in-memory).

H2 Database выбрана из-за ее быстроты и легковесности, она идеально подходит для работы в Spring Framework. Т.к. объем данных, сохраняемых в БД, относительно невелик, H2 Database настроена на работу с файловой БД, благодаря чему отсутствует необходимость развертывания отдельного сервера с базой данных.

#### Вспомогательные инструменты

Дополнительно используются:

* **Git и GitHub** – для управления версиями кода и совместной работы.
* **Postman** – для тестирования API.
* **Adobe Photoshop, Paint** – для работы со значками, картинками и фотографиями, используемыми в графическом интерфейсе сервиса.

Выбранные технологии и инструменты позволяют создать удобное, надежное и производительное веб-приложение, соответствующее современным требованиям. Они обеспечивают быструю разработку, гибкость в масштабировании и удобство поддержки проекта.

**2.4. Версионный контроль в разработке**

Разработка веб-приложения визитной карточки фотографа требует эффективного подхода к управлению кодом и процессом выполнения задач. В данном разделе рассмотрены методологии и инструменты, применяемые для контроля версий проекта.

### Версионный контроль

Версионный контроль является неотъемлемой частью разработки программного обеспечения. Он позволяет отслеживать изменения в коде, управлять различными версиями приложения и обеспечивать совместную работу команды разработчиков. В рамках данного проекта использовалась система контроля версий **Git**, а в качестве удаленного репозитория — **GitHub**.

Использование Git обеспечивает:

* Отслеживание всех изменений в коде, что позволяет при необходимости вернуться к предыдущим версиям.
* Ветвление (branching) для организации параллельной работы над различными функциональными модулями.
* Возможность совместной работы нескольких разработчиков без риска потери данных.
* Автоматизацию развертывания через системы CI/CD.

**Глава 3. Разработка веб-приложения визитной карточки фотографа с системой онлайн-записи**

**3.1. Подготовка и настройка проекта**

Перед началом активной разработки веб-приложения необходимо провести ряд подготовительных работ, включая настройку среды разработки, выбор технологий и инструментов, а также базовую конфигурацию проекта. Этот этап обеспечивает удобство и эффективность дальнейшей разработки.

Разработка проекта реализована в среде разработке IntelliJ IDEA 2024. Cоздан проект Spring Boot c соответствующими деталями проекта (метаданные, язык программирования, версия Java и т.п.).

В качестве сборщика в проекте использован фреймворк Apache Maven. Для подключения всех необходимых для разработки модулей, таких как Spring Boot, Spring MVC, Spring Data JPA, H2 Database, Spring Security, в конфигурационный файл проекта **pom.xml** добавлены следующие зависимости:

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <optional>true</optional>  
 </dependency>  
</dependencies>

Выполнение данных этапов позволяет подготовить стабильную и функциональную основу для дальнейшей разработки и оптимизируют процесс работы команды.

**3.2. Проектирование и реализация удобного пользовательского интерфейса**

Проектирование и реализация пользовательского интерфейса (UI) — важнейший этап разработки веб-приложения, поскольку от этого напрямую зависит удобство взаимодействия пользователя с системой, а также его общее восприятие приложения. В данном разделе рассматриваются ключевые принципы, подходы и технологии, использованные для создания интуитивно понятного и функционального интерфейса для веб-приложения.

#### ****Пользовательский опыт (UX) и его значение****

Основой разработки удобного интерфейса является принцип ориентированности на пользователя. UX-дизайн (User Experience) заключается в создании интерфейса, который обеспечивает комфортное и эффективное взаимодействие с веб-приложением. В процессе проектирования UI учитывались следующие аспекты:

* **Простота и интуитивность**: интерфейс должен быть понятным, даже для пользователей без технической подготовки. Важно, чтобы пользователь мог быстро понять, как использовать приложение без необходимости изучать сложные инструкции.
* **Эффективность навигации**: структура приложения должна быть логичной и удобной, чтобы пользователь мог без труда находить необходимые функции.
* **Минимализм**: избегание перегрузки интерфейса лишними элементами. Каждый элемент должен выполнять конкретную роль, не отвлекая внимание от основной задачи.

#### 

#### ****Создание прототипа и дизайна****

Процесс проектирования пользовательского интерфейса начался с создания шаблона веб-страниц. Для этого также использовалась среда программирования IntelliJ IDEA.

Прототипирование включало следующие этапы:

* **Разработка структуры страниц**: на основе бизнес-требований были выделены основные страницы приложения, такие как главная страница, страница профиля, страница с примерами работ, прайсом, отзывами и системой онлайн-записи.
* **Создание wireframe'ов** (каркасных макетов): на этом этапе разрабатывались упрощенные версии страниц без детализированного дизайна, что позволяло сосредоточиться на функциональности и логике навигации.
* **Дизайн интерфейса**: на основе утвержденных прототипов был разработан визуальный стиль приложения, включающий выбор цветовой схемы, типографики, иконок и элементов управления. Цветовая палитра была выбрана с учетом предпочтений целевой аудитории и психологического воздействия на пользователей.

#### ****Реализация интерфейса****

Для реализации интерфейса использовались современные веб-технологии, включая HTML5, CSS3 и JavaScript, а также шаблонизатор Thymeleaf.

* **HTML5** обеспечил семантическую разметку и поддержку мультимедийных элементов.
* **Thymeleaf шаблонизатор** обеспечил динамическое обновление контента на страницах, а также интерактивные элементы, такие как модальные окна, формы для записи на услуги и отображение отзывов.

#### ****Тестирование и улучшение интерфейса****

После реализации интерфейса были проведены тесты на различных браузерах для обеспечения совместимости и выявления потенциальных проблем. В ходе тестирования учитывались следующие аспекты:

* **Проверка на пользовательские ошибки**: тестировалась логика взаимодействия пользователя с интерфейсом, чтобы минимизировать количество ошибок при его использовании.
* **Обратная связь пользователей**: тестирование с реальными пользователями позволило выявить нюансы, которые могли быть упущены на стадии разработки.

#### ****Интеграция с функциональными модулями****

Интерфейс был интегрирован с основными функциональными модулями приложения такими как система онлайн-записи, форма подачи отзывов и взаимодействие пользователей с администратором (фотографом). Взаимосвязь пользователей с различными модулями организована при помощи компонентов Controller фреймворка Spring Boot, «слушающих» запросы пользователя и отправляющих динамические шаблоны страниц в ответ, с соответствующей логикой. Удобное и понятное взаимодействие с этими модулями обеспечивает высокую эффективность использования приложения и удовлетворенность пользователей.

Проектирование и реализация пользовательского интерфейса являются ключевыми этапами разработки веб-приложения, от которых зависит успешность всего продукта. В результате, приложению был придан современный и привлекательный вид, с обеспечением максимальной удобности и функциональности для пользователей. Интуитивно понятный интерфейс, адаптивный дизайн и продуманные элементы взаимодействия позволяют пользователям без труда использовать все возможности веб-приложения, что, в свою очередь, способствует высокой эффективности работы и положительным отзывам пользователей.

**3.3. Обеспечение безопасности веб-приложения и защита пользовательских данных**

Современные веб-приложения становятся не только все более функциональными и удобными, но и более уязвимыми для различных типов атак, что делает обеспечение их безопасности важнейшей частью разработки. Защита пользовательских данных, безопасность авторизации и аутентификации, а также предотвращение атак на систему — все это требует комплексного подхода, который эффективно решается с использованием современных фреймворков, таких как **Spring Security**. Этот мощный инструмент для обеспечения безопасности в экосистеме Spring предоставляет разработчикам широкий набор функций для защиты веб-приложений.

#### ****Значение безопасности в веб-приложениях****

Веб-приложения зачастую обрабатывают чувствительную информацию пользователей, такую как личные данные, платежные реквизиты и историю действий. Потеря или утечка таких данных может привести к серьезным юридическим и финансовым последствиям для компании, а также к утрате доверия со стороны пользователей. Следовательно, обеспечение безопасности должно быть встроено в саму архитектуру приложения.

Основные угрозы, с которыми сталкиваются веб-приложения, включают:

* **Неавторизованный доступ**: попытки злоумышленников получить доступ к защищенным частям приложения без надлежащей аутентификации.
* **Утечка данных**: неконтролируемая утечка пользовательской информации, которая может произойти из-за ошибок в коде или недостаточной защиты.
* **Атаки с использованием уязвимостей**: такие как SQL-инъекции, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), и другие.

Для защиты от этих угроз необходимо реализовать эффективные механизмы аутентификации, авторизации, шифрования и контроля доступа.

#### ****Spring Security как решение для обеспечения безопасности****

**Spring Security** — это широко используемый фреймворк для обеспечения безопасности приложений, который является частью экосистемы Spring. Он предоставляет гибкие и масштабируемые механизмы для аутентификации и авторизации, защиты от атак, а также для мониторинга безопасности приложения.

Основные особенности Spring Security:

* **Аутентификация**: определяет, кто именно использует приложение, и предоставляет механизмы для реализации различных схем аутентификации, включая формы логина, OAuth, JWT и другие.
* **Авторизация**: определяет, какие права имеет аутентифицированный пользователь, и позволяет настроить доступ к различным частям приложения в зависимости от его ролей и полномочий.
* **Защита от атак**: включает защиту от таких угроз, как CSRF, XSS, а также фильтрацию запросов, чтобы предотвратить несанкционированный доступ.

#### ****Реализация аутентификации с помощью Spring Security****

Аутентификация — это процесс проверки личности пользователя, который вступает в систему. В Spring Security аутентификация может быть реализована с использованием различных методов, таких как:

* **Форма входа (Form-based Authentication)**: это традиционный способ аутентификации, при котором пользователь вводит логин и пароль на веб-форме. Spring Security поддерживает настройку стандартной страницы входа, а также возможность настройки пользовательских форм.
* **Basic Authentication**: механизм HTTP-аутентификации, при котором логин и пароль передаются в заголовке каждого запроса. Этот способ обычно используется в API.
* **OAuth2 и JWT**: для более сложных систем используется аутентификация через OAuth2, что позволяет интегрировать внешние системы аутентификации, такие как Google, Facebook, или другие сервисы. В этом случае Spring Security интегрируется с авторизационными серверами, используя протокол OAuth2.

После аутентификации важно реализовать механизмы авторизации, которые будут контролировать доступ к различным частям веб-приложения в зависимости от роли или прав пользователя. Spring Security предоставляет возможность гибкой настройки доступа через аннотации и выражения:

* **Роли и разрешения**: можно определить различные роли, такие как **ROLE\_USER**, **ROLE\_ADMIN**, и настроить доступ в зависимости от роли пользователя.
* **Аннотация @PreAuthorize**: позволяет ограничить доступ к методам на уровне кода, указывая выражение доступа, например, @PreAuthorize("hasRole('ADMIN')").

В проекте, для аутентификации пользователей, использована форма входа. Все пользователи, которые регистрируются в системе, получают роль USER, и перенаправляются на пользовательскую панель. Аккаунт с ролью ADMIN для администратора/фотографа создан и внесен в БД заранее, при каждом запуске сервера эта запись проверяется. Ниже представлена конфигурация аутентификации и авторизации проекта:

@Bean  
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(AbstractHttpConfigurer::disable)  
 .cors(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(registry -> {  
 registry.requestMatchers("/index","/login","/contacts","/price","/gallery",  
 "/registration","/reviews","/entrance",

"/online-record","/success",  
 "/errorRegistration", "/",

"/send-success","/logout","/images/\*\*",  
 "/h2-console/\*\*").permitAll();  
 registry.requestMatchers("/admin/\*\*").hasAuthority("ADMIN");   
 registry.requestMatchers("/user/\*\*").hasAuthority("USER");   
 registry.anyRequest().permitAll()  
 })  
 .formLogin(form -> form  
 .loginPage("/login")  
 .loginProcessingUrl("/login") .successHandler(customSuccessHandler)   
 .permitAll()  
 )  
 .logout(logout -> logout  
 .logoutUrl("/logout")  
 .logoutSuccessUrl("/")  
 .permitAll()  
 )  
 .build();  
}

#### ****Защита от CSRF атак****

**Cross-Site Request Forgery (CSRF)** — это тип атаки, при котором злоумышленник заставляет пользователя выполнить нежелательное действие на веб-приложении, в которое он уже аутентифицирован.

**CORS (Cross-Origin Resource Sharing)** — это механизм, позволяющий веб-страницам запрашивать ресурсы с других доменов. Однако он может быть использован злоумышленниками для выполнения запросов с поддельных источников, что создаёт угрозу безопасности. Spring Security предоставляет механизм для настройки **CORS** в веб-приложении, позволяя контролировать, какие домены могут отправлять запросы к серверу. Это достигается через конфигурацию, которая ограничивает доступ только для доверенных источников. Активируя CORS фильтр в Spring Security, можно ограничить или разрешить доступ к ресурсам только с определенных доменов, тем самым предотвращая возможные атаки, связанные с неправильной настройкой CORS.

Spring Security включает защиту от CSRF и **CORS** атак по умолчанию. Он генерирует уникальный токен для каждого запроса, который должен быть включен в форму на стороне клиента. Сервер проверяет этот токен при получении запроса, чтобы убедиться, что он был отправлен с доверенного источника. В проекте также подключена защита от данных атак при помощи дополнительных методов в бине securityFilterChain класса SecurityConfig:

.csrf(AbstractHttpConfigurer::disable)  
.cors(AbstractHttpConfigurer::disable)

#### ****Шифрование данных и защита паролей****

Важно обеспечить безопасность хранения паролей пользователей. Для этого Spring Security поддерживает интеграцию с алгоритмами шифрования, такими как **BCrypt**. Это позволяет безопасно хранить пароли в базе данных, используя «соль» и многократные итерации хеширования. Кроме того, можно реализовать собственную методику шифрования паролей, примитивный пример которой, исключительно в демонстрационных целях, реализован в проекте:

@Component  
public class MyPasswordEncoder implements PasswordEncoder {  
 @Override  
 public String encode(CharSequence rawPassword) {  
 return "hashed\_" + rawPassword;  
 }  
  
 @Override  
 public boolean matches(CharSequence rawPassword, String encodedPassword) {  
 return Objects.*equals*(encode(rawPassword), encodedPassword);  
 }

Использование Spring Security для обеспечения безопасности веб-приложений предоставляет разработчикам мощный набор инструментов для защиты от различных угроз. Аутентификация и авторизация, защита от атак, таких как CSRF и XSS, а также безопасное хранение пользовательских данных — все эти функции значительно повышают уровень безопасности приложения. Важно учитывать эти аспекты на этапе проектирования и интегрировать механизмы безопасности с самого начала разработки, чтобы минимизировать риски и обеспечить надежную защиту пользовательской информации.

**3.4. Интеграция базы данных и работа с серверной логикой**

Интеграция базы данных и реализация серверной логики являются неотъемлемой частью создания веб-приложений, которые обрабатывают динамические данные. В данном разделе рассматривается процесс подключения и взаимодействия с базой данных в рамках разработки веб-приложения с использованием фреймворка Spring, а также описание реализации серверной логики, которая обрабатывает запросы пользователей, взаимодействует с базой данных и формирует ответ.

#### ****Выбор и настройка базы данных****

Одним из первых шагов при разработке веб-приложения является выбор подходящей системы управления базами данных (СУБД). В зависимости от задач проекта может быть выбран как реляционный подход, так и NoSQL-решение. Для данного веб-приложения в качестве СУБД была выбрана **H2 Database**, которая является одной из наиболее популярных реляционных баз данных, поддерживающих примитивные запросы, запись данных в файл и высокую производительность.

Для подключения Spring-приложения к базе данных используется **JPA** (Java Persistence API), что позволяет значительно упростить взаимодействие с базой данных, устраняя необходимость вручную писать SQL-запросы для большинства операций. Spring Data JPA автоматически генерирует необходимые запросы, исходя из имен методов интерфейса репозитория.

Ниже представлены настройки для БД в файле application.properties, прописанные в проекте:

spring.datasource.url=jdbc:h2:file:./database/testdb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

Этот файл конфигурации определяет настройки подключения к базе данных H2, включая URL с местоположением файла БД. Параметр ddl-auto=update указывает Hibernate автоматически обновлять схему базы данных, основываясь на моделях сущностей. Также включена возможность доступа к БД через веб-интерфейс.

**Проектирование сущностей и таблиц**

После того как установлено соединение с базой данных, следующим шагом является проектирование сущностей, которые будут отображаться в таблицах базы данных. Важно, чтобы структура данных была логичной и соответствовала требованиям приложения.

В рамках данного веб-приложения основной сущностью является **Пользователь**, который может быть зарегистрирован в системе и оставлять отзывы о предоставленных услугах. Кроме того, создаются сущности для таких объектов, как **Сообщение**, **Свободный день**, **Отзыв**, и **Запись на услугу**. Каждая из этих сущностей будет представлять отдельную таблицу в базе данных. Ниже, в качестве примера, представлен класс - сущность User:

@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String login;  
 private String password;  
 private String firstName;  
 private String lastName;  
 private String role;  
 private int saleLevel; //уровень скидки в процентах  
 private String email;  
 private String phoneNumber;  
 private boolean limitReview = false; //указывает, исчерпан ли лимит по отзывам (доступен один отзыв)  
 private boolean limitActiveOrder = false; //указывает, исчерпан ли лимит по заказам (доступен один активный заказ)

//геттеры + сеттеры

Cущность **User** описывает таблицу пользователей, где каждый пользователь имеет уникальный идентификатор, имя пользователя, пароль, электронную почту, имя и фамилию, роль (уровень доступа), уровень скидки, номер телефона и флаги - лимиты по отзывам(можно оставить только один) и заказам (можно иметь только один текущий заказ).

Для других сущностей, таких как **FreeDay** (Свободный день), **Review** (Отзыв), **Message** (Сообщение) и **Order** (заказ/запись) процесс создания будет аналогичным. Все сущности аннотируются с помощью **@Entity** и **@Table**, что позволяет Spring автоматически создавать соответствующие таблицы в базе данных.

#### ****Реализация репозиториев с использованием Spring Data JPA****

Для взаимодействия с базой данных используется **Spring Data JPA**. Этот фреймворк предоставляет удобные репозитории для работы с сущностями, позволяя избежать написания большого количества SQL-кода. В частности, репозитории создаются как интерфейсы, которые наследуют интерфейс JpaRepository или CrudRepository. Ниже представлен интерфейс репозитория для сущностей User:

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
 Optional<User> findByLogin(String login);  
}

Репозиторий предоставляет методы для поиска пользователя по имени. **JpaRepository** автоматически генерирует реализацию всех базовых операций с базой данных, таких как сохранение, удаление, обновление и поиск.

Аналогично создаются репозитории остальных сущностей, что позволяет эффективно работать с данными, не обращаясь к SQL-запросам напрямую.

#### ****Реализация серверной логики****

Серверная логика приложения обрабатывает запросы, полученные от клиента, выполняет соответствующие операции с данными и формирует ответ. В веб-приложении это реализуется с помощью контроллеров, сервисов и репозиториев.

Контроллеры в Spring MVC отвечают за обработку HTTP-запросов и передачу их соответствующим сервисам, которые выполняют основную бизнес-логику. Сервисы, в свою очередь, используют репозитории для работы с базой данных и обработки данных, полученных от клиента. Ниже представлен фрагмент кода из проекта класса MainController:

@Controller  
@RequestMapping("/")  
public class MainController {  
 private final MessageService messageService;  
 private final UserService service;  
 private final ReviewService reviewService;  
  
 public MainController(MessageService messageService, UserService service, ReviewService reviewService) {  
 this.messageService = messageService;  
 this.service = service;  
 this.reviewService = reviewService;  
 }  
  
 @GetMapping("/")  
 public String index(Model model){  
 model.addAttribute("title", "Фотограф Наталья Сулагаева: Главная страница");  
 return "index";  
 }  
 @GetMapping("/gallery")  
 public String galerry(Model model){  
 return "gallery.html";  
 }  
 @GetMapping("/price")  
 public String price(Model model){  
 return "price.html";  
 }  
 @GetMapping("/online-record")  
 public String entry(Model model){  
 return "online-record.html";  
 }  
  
 @GetMapping("/contacts")  
 public String contact(Model model){  
 return "contacts.html";  
 }  
  
 @PostMapping("/contacts")  
 public String sendMessage(@ModelAttribute("message") Message message){  
 messageService.saveMessage(message);  
 return "redirect:/send-success";  
 }  
  
 @GetMapping("/reviews")  
 public String reviews(Model model){  
 model.addAttribute("averageRate", reviewService.calcAverageRating());  
 model.addAttribute("reviews", reviewService.findAllReview());  
 return "reviews.html";  
 }  
  
 @GetMapping("/login")  
 public String login(){  
 return "login-form.html";  
 }  
  
 @GetMapping("/registration")  
 public String registration(Model model){  
 model.addAttribute("user", new User());  
 return "registration.html";}  
  
 @PostMapping("/registration")  
 public String registerUser(@ModelAttribute("user") User user){  
 System.*out*.println(user.getLogin() + "успешно зарегистрирован в системе");  
 return service.registerUser(user);  
 }  
  
 @GetMapping("/success")  
 public String successRegistration(){  
 return "success.html";}  
  
 @GetMapping("/errorRegistration")  
 public String errorRegistration(){  
 return "errorRegistration.html";  
 }  
  
 @GetMapping("/send-success")  
 public String successSendMessage(){  
 return "send-success.html";}  
}

В данном примере **MainController** обрабатывает запросы неавторизованных пользователей. Контроллер вызывает методы сервисов, которые занимаются бизнес-логикой, а сам контроллер просто возвращает ответ в виде HTTP-ответа.

Ниже представлен фрагмент кода проекта, класса UserService:

@Service  
public class UserService {  
 private final UserRepository repository;  
 private final MyPasswordEncoder encoder;  
  
 public UserService(UserRepository repository, MyPasswordEncoder encoder) {  
 this.repository = repository;  
 this.encoder = encoder;  
 }  
  
 public String registerUser(User user){  
 for (User u : repository.findAll()) {  
 if (user.getLogin().equals(u.getLogin())) return "redirect:/errorRegistration";  
 if (user.getEmail().equals(u.getEmail())) return "redirect:/errorRegistration";  
 }  
 user.setRole("USER");  
 user.setPassword(encoder.encode(user.getPassword()));  
 user.setSaleLevel(5);  
 repository.save(user);  
 return "redirect:/success";  
 }  
  
 public void updateLimitOrderUser(String login, boolean blocked){  
 User user = repository.findByLogin(login).get();  
 user.setLimitActiveOrder(blocked);  
 repository.save(user);  
 }  
  
 public void updateLimitReviewUser(String login, boolean blocked){  
 User user = repository.findByLogin(login).get();  
 user.setLimitReview(blocked);  
 repository.save(user);  
 }  
  
 public User getUser(String login){  
 return repository.findByLogin(login).get();  
 }  
  
 public User getUserById(Long id){  
 return repository.findById(id).get();  
 }  
  
 public List<User> getAllUsers(){  
 return repository.findAll();  
 }  
  
 public void levelUpUser(Long id){  
 User user = repository.findById(id).get();  
 user.setSaleLevel(10);  
 repository.save(user);  
 }  
}

Сервис выполняет регистрацию пользователя, поиск пользователя по идентификатору и другие полезные функции, необходимые для полноценной работы веб-приложения.

Интеграция базы данных и реализация серверной логики — это ключевые этапы в разработке любого веб-приложения. Использование технологий, таких как **Spring Data JPA** и **Spring MVC**, позволяет значительно упростить взаимодействие с базой данных и разработку серверной логики, минимизируя количество необходимого кода и обеспечивая высокую производительность. Внедрение правильной структуры и архитектуры приложения позволяет эффективно управлять данными, обработкой запросов и исключений, что в итоге приводит к созданию стабильного и безопасного веб-приложения.

**3.5. Разработка функциональности: система онлайн-записи и модуль отзывов**

Разработка функциональности системы онлайн-записи и модуля отзывов для веб-приложения фотографа представляет собой ключевые аспекты для создания удобного и эффективного интерфейса для пользователей. Система онлайн-записи позволяет пользователю легко и быстро записаться на фотосессию, а модуль отзывов способствует повышению доверия к предоставляемым услугам, позволяя клиентам оставлять мнения о качестве работы. В этом разделе рассматривается проектирование и реализация этих компонентов, их интеграция в общую архитектуру приложения, а также особенности реализации с использованием современных технологий.

#### ****Система онлайн-записи****

Система онлайн-записи — это одна из важнейших частей веб-приложения, которая позволяет пользователю выбрать удобное время и записаться на фотосессию с фотографом. Это существенно упрощает процесс взаимодействия между фотографом и клиентом, исключая необходимость телефонных звонков и личных встреч для уточнения доступности времени.

##### 1.1. ****Проектирование базы данных для системы онлайн-записи****

Для реализации системы онлайн-записи требуется создание соответствующих сущностей в базе данных. Основные элементы этой системы включают **заказ** (например, фотосессия), **клиента** (пользователя веб-приложения) и **свободный день** (дни, когда фотограф работает, с указанием свободных интервалов времени). Для хранения информации о записи создана сущность Order, которая будет связывать клиента с конкретной датой и временем записи:

@Entity  
@Table(name = "orders")  
public class Order {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private Long customerId;  
 private String fullName;  
 private String phoneNumber;  
 private LocalDate dateOrder = LocalDate.*now*();  
 private LocalDate date;  
 private String time;  
 private String location;  
 private String wishes; //примечания/пожелания со стороны клиента  
 private String status = "Зарегистрирована, не подтверждена";

//геттеры + сеттеры

После того как сущности базы данных определены, следующим шагом является реализация бизнес-логики для управления процессом записи. Важным моментом является возможность выбора времени записи, которое не должно пересекаться с уже существующими записями для данной услуги.

Для этого в приложении реализуется изменение поля свободного дня, отвечающего за конкретное время:

FreeDay freeDay = freeDayService.findFreeDay(id);  
freeDay.updateTimeFromString(time, false);  
Order order1 = new Order();  
order1.setTime(convertTime(time));  
order1.setDate(freeDay.getDate());  
order1.setCustomerId(userService.getUser(user.getName()).getId());  
order1.setFullName(userService.getUser(user.getName()).getFirstName() + " "  
 + userService.getUser(user.getName()).getLastName());  
order1.setPhoneNumber(order.getPhoneNumber());  
order1.setLocation(order.getLocation());  
order1.setWishes((order.getWishes() == null)?" ":order.getWishes());  
freeDayService.updateFreeDay(freeDay);  
userService.updateLimitOrderUser(user.getName(), true);  
orderService.createOrder(order1);

Если администратор (фотограф) отменяет запись, то выполняется следующий метод, делающий определенное поле с интервалом времени свободного дня снова свободным:

public void liberationDayOfOrderId(Long orderId){  
 LocalDate date = orderRepository.findById(orderId).get().getDate();  
 String time = orderRepository.findById(orderId).get().getTime();  
 for (FreeDay day : freeDayRepository.findAll()){  
 if (day.getDate().equals(date)) {  
 day.updateTimeFromNumber(time, true);  
 }  
 }  
}

#### ****Модуль отзывов****

Модуль отзывов является не менее важной частью веб-приложения, поскольку он позволяет пользователям оставлять мнения о качестве услуг, а также помогает потенциальным клиентам сформировать представление о работе фотографа. Отзывы помогают повысить доверие к услугам, предоставляемым через веб-приложение, и способствуют популяризации бренда.

##### ****Проектирование базы данных для модуля отзывов****

Для реализации модуля отзывов создана сущность Review, которая связана с пользователем и записью. Сущность Review хранит текст отзыва, рейтинг и дату создания отзыва:

@Entity  
@Table(name = "reviews")  
public class Review {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*SEQUENCE*)  
 private Long id;  
 private String name;  
 private int exellent;  
 private LocalDate publicDate = LocalDate.*now*();  
 private String comment;

Для модуля отзывов важно, чтобы система позволяла пользователю оставлять только один отзыв. Бизнес-логика будет обеспечивать это ограничение и после размещения одного отзыва система будет отображать оповещение - «Вы уже оставляли отзыв». Реализовано это на html странице при помощи шаблонизатора, с использованием условного оператора if:

<div th:if="${limitReview}">  
 <div class="centered">  
 <p>Вы уже оставляли отзыв</p>  
 </div>  
</div>

Разработка функциональности системы онлайн-записи и модуля отзывов играет ключевую роль в улучшении пользовательского опыта и повышении удовлетворенности клиентов. Реализация этих компонентов требует внимательного подхода к проектированию базы данных, созданию бизнес-логики и разработке удобных интерфейсов. Использование современных технологий, таких как **Spring Boot, Spring MVC, Thymeleaf**, позволяет создавать динамичные, удобные и высокопроизводительные системы, которые отвечают потребностям пользователей и обеспечивают успешную работу веб-приложения.

**3.6. Оптимизация производительности и обеспечение стабильности работы**

#### Оптимизация производительности и обеспечение стабильности работы веб-приложения являются важными аспектами его разработки, особенно когда речь идет о приложениях, которые обслуживают большое количество пользователей или выполняют сложные операции. Важно, чтобы приложение не только предоставляло функциональные возможности, но и работало быстро, эффективно и стабильно в условиях реальной эксплуатации.

#### ****Обеспечение стабильности с помощью тестирования****

Для достижения высокой стабильности работы веб-приложения необходимо проводить регулярное **нагрузочное тестирование**, чтобы убедиться в его способности обрабатывать большое количество запросов без деградации производительности. Также следует проводить **юнит-тестирование** и **интеграционное тестирование**, чтобы убедиться в корректности работы различных компонентов системы.

Использование инструментов для автоматизированного тестирования, таких как **JUnit** и **Mockito**, помогает предотвратить ошибки и баги, которые могут привести к сбоям в работе приложения.

Оптимизация производительности и обеспечение стабильности работы веб-приложения являются важнейшими задачами в процессе разработки. Применение эффективных методов работы с базой данных, кэширования, балансировки нагрузки и асинхронной обработки запросов позволяет значительно улучшить производительность системы. Кроме того, внедрение систем мониторинга, логирования и тестирования помогает поддерживать стабильную работу приложения и быстро реагировать на возникающие проблемы.

**3.7. Подведение итогов**

Разработка веб-приложения визитной карточки фотографа с системой онлайн-записи и отзывов охватывала множество этапов, начиная от проектирования пользовательского интерфейса и серверной логики, заканчивая оптимизацией производительности и обеспечением безопасности. В данном разделе подведем итоги выполненной работы, проанализируем достигнутые результаты и рассмотрим ключевые аспекты реализации проекта.

#### ****Достижения в разработке веб-приложения****

В ходе работы над проектом была успешно реализована **система онлайн-записи**, обеспечивающая удобный и интуитивно понятный процесс выбора услуг и времени для клиентов. Благодаря использованию **Spring Boot** и **Spring Data JPA**, удалось создать гибкую и масштабируемую серверную архитектуру, а интеграция с базой данных позволила эффективно управлять бронированиями и контролировать доступные временные интервалы.

Кроме того, разработанный **модуль отзывов** позволил клиентам оставлять обратную связь о предоставленных услугах, что способствует росту доверия и репутации фотографа. Интерактивный пользовательский интерфейс обеспечивает удобное взаимодействие с функционалом веб-приложения.

#### ****Обеспечение безопасности и надежности****

Одним из ключевых аспектов разработки веб-приложения стало обеспечение **защиты пользовательских данных**. Внедрение **Spring Security** позволило реализовать надежную систему аутентификации и авторизации, защиту от атак, таких как **SQL-инъекции, XSS и CSRF**. Кроме того, были применены механизмы шифрования паролей и безопасного хранения данных пользователей, что повышает уровень конфиденциальности информации.

Особое внимание уделялось защите от **CORS-атак**, что предотвращает несанкционированные запросы из других доменов и гарантирует корректную работу API. Это позволило минимизировать риски утечки данных и создать безопасную среду для пользователей веб-приложения.

#### ****Выводы и перспективы развития****

Результатом проведенной работы стало создание полнофункционального веб-приложения, которое обладает удобным интерфейсом, высокой производительностью и надежной системой безопасности. Реализованные функциональные модули соответствуют современным требованиям веб-разработки и позволяют фотографу эффективно взаимодействовать с клиентами.

Перспективы развития проекта включают в себя **дальнейшее расширение функциональности**, добавление интеграции с популярными платежными системами для онлайн-оплаты услуг, а также внедрение автоматических напоминаний о записях через **SMS и email-рассылку**. Возможна также разработка мобильной версии веб-приложения для удобного использования с мобильных устройств.

В целом, выполненная работа демонстрирует, что веб-приложение может значительно упростить процесс взаимодействия фотографа с клиентами, повысить уровень удобства и автоматизировать ключевые бизнес-процессы, что делает его полезным инструментом для ведения профессиональной деятельности.

**Заключение**

В ходе выполнения дипломного проекта была разработана веб-визитка фотографа с системой онлайн-записи и модулем отзывов, обеспечивающая удобное взаимодействие между фотографом и его клиентами. Проект включал в себя полное проектирование, разработку и тестирование веб-приложения, что позволило создать современный, функциональный и безопасный инструмент для ведения профессиональной деятельности в сфере фотографии.

Разработка велась с использованием **Spring Boot** для серверной части и **HTML** и **Thymeleaf** для клиентской стороны, что обеспечило высокую производительность и удобство работы с приложением. База данных была реализована на основе **H2 Database**, что позволило организовать надежное хранение информации о записях, пользователях и отзывах. Интеграция системы безопасности с помощью **Spring Security** обеспечила защиту пользовательских данных.

Основными функциональными модулями стали:

* **Система онлайн-записи**, позволяющая клиентам в удобном формате бронировать услуги фотографа;
* **Модуль отзывов**, который дает возможность пользователям оставлять и просматривать отзывы о предоставленных услугах;
* **Система безопасности**, включающая защиту от атак и несанкционированного доступа.

Результатом работы стало создание веб-приложения, которое решает поставленные задачи и соответствует современным требованиям веб-разработки. Система удобна в использовании, надежна и безопасна, что делает её актуальной для фотографов и других специалистов, работающих с клиентами по записи.

Перспективы дальнейшего развития проекта включают:

* **Расширение функциональности**, например, интеграцию онлайн-оплаты за услуги;
* **Автоматизацию уведомлений**, включая напоминания о записях через email и SMS;
* **Разработку мобильного приложения** на основе существующего веб-приложения.

В целом, выполненная работа подтверждает актуальность разработки подобного рода веб-приложений и демонстрирует, что использование современных технологий позволяет создавать удобные, эффективные и защищенные решения, облегчающие работу бизнеса и взаимодействие с клиентами.

**Список используемой литературы**

1. Бауэр К., Кинг Г. **Spring в действии. 6-е издание**. – СПб.: Питер, 2022. – 704 с.
2. Фримен Э., Робсон Э. **Spring и Spring Boot. Простое и наглядное руководство**. – СПб.: Питер, 2021. – 368 с.
3. Фаулер М. **Проектирование программного обеспечения. Архитектурные стили и примеры для разработчиков и архитекторов**. – СПб.: Питер, 2020. – 448 с.
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. **Приемы объектно-ориентированного проектирования. Паттерны проектирования**. – СПб.: Питер, 2021. – 368 с.
5. Фленов М. **Java. Полное руководство**. – СПб.: Питер, 2023. – 1280 с.

### ****Официальная документация и онлайн-ресурсы****

Spring Framework Documentation – <https://docs.spring.io/spring-framework/docs/current/reference/html/>

Spring Boot Documentation – <https://docs.spring.io/spring-boot/docs/current/reference/html/>

Spring Security Documentation – <https://docs.spring.io/spring-security/reference/>

Hibernate ORM Documentation – <https://hibernate.org/orm/documentation/>

Spring Data JPA – <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

H2 Database – https://www.h2database.com/html/main.html

Thymeleaf – https://www.thymeleaf.org/documentation.html

**Приложения**

Ссылка на проект: https://github.com/sulagaevdim/photograph\_project\_diplom.git