

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ярославский государственный технический университет»
Кафедра «Цифровых систем»

Курсовая работы защищена
с оценкой _____
Руководитель,
ассистент
_____ Пашичев В.С.
«__» _____ 2024

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ НА SPRING

Расчетно-пояснительная записка к курсовой работе по дисциплине
«Технологии программирования»

ЯГТУ 09.03.02 – 000 КП

Нормоконтролер
Ассистент
_____ Пашичев В.С.
«__» _____ 2024

Проект выполнил
студент группы ЦИС-27
Бычков Е.А. _____
«__» _____ 2024

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационные системы»

ЗАДАНИЕ №1

по курсовому проектированию

Студенту: Бычкову Евгению Александровичу
Институт Цифровых систем **курс 2 группа** ЦИС-27

1. Тема проекта и исходные данные
Тема «Разработка Web-приложения на Spring»

Исходные данные: Разработка Веб-приложения на Spring Boot по заданной теме – веб приложение для подбора конфигурации компьютера.

Оглавление

Введение	6
Проектирование базы данных.....	8
Логическая модель.....	8
Модели.....	10
Класс Complect.....	10
Импортируемые библиотеки и аннотации.....	10
Аннотации на уровне класса	10
Поля класса	10
Класс Computer	11
Импортируемые библиотеки и аннотации.....	11
Аннотации на уровне класса	11
Поля класса	11
Взаимосвязи с другими сущностями.....	12
Класс User	12
Импортируемые библиотеки и аннотации.....	12
Аннотации на уровне класса	12
Поля класса	13
Взаимосвязи с другими сущностями.....	13
Реализация интерфейса UserDetails	13
Класс UserAuthority	14
Импортируемые библиотеки и интерфейсы.....	14
Перечисление прав	14
Реализация метода getAuthority	14
Класс UserRole	14
Импортируемые библиотеки и аннотации.....	14
Аннотации на уровне класса	15
Поля класса	15
Аннотации на уровне полей.....	15
Класс WishList.....	16
Импортируемые библиотеки и аннотации.....	16
Аннотации на уровне класса	16
Поля класса	16

Аннотации на уровне полей	17
Репозитории	18
ComplectRepository	18
ComputerRepository	18
UserRepository	18
UserRolesRepository	18
WishListRepository	18
Сервис	19
ComplectServiceImpl	19
Импортируемые библиотеки и аннотации	19
Аннотации на уровне класса	19
Поля класса	19
Методы класса	19
ComputerServiceImpl	20
Импортируемые библиотеки и аннотации	20
Аннотации на уровне класса	20
Поля класса	20
Методы класса	21
UserServiceImpl	21
Импортируемые библиотеки и аннотации	21
Аннотации на уровне класса	22
Поля класса	22
Методы класса	22
WishListServiceImpl	22
Импортируемые библиотеки и аннотации	22
Аннотации на уровне класса	23
Поля класса	23
Методы класса	23
Контроллеры	24
ComplectGetController	24
Импортируемые библиотеки и аннотации	24
Аннотации на уровне класса	24
Поля класса	24

ComplectPostController	25
ComputerController	26
Описание класса:	26
Поля класса:	26
Методы класса:	26
RegistrationController	27
Описание класса:	27
Поля класса:	27
Методы класса:	27
UserController	27
Описание класса:	27
Поля класса:	27
Методы класса:	28
Exceptions	29
ComplectNotFoundException	29
UsernameAlreadyExistsException	29
UsernameNotFoundException	29
Конфигурация	30
SpringSecurityConfiguration	30
Описание класса:	30
Методы класса:	30
Логирование	31
ControllerLogAspect	31
Описание класса:	31
Методы класса:	31
MyBeanPostProcessor	31
Описание класса:	31
Методы класса:	31
Заключение	34
Приложение:	35

Введение

Современные информационные технологии развиваются стремительными темпами, и с каждым годом растет потребность в высокопроизводительных и надежных компьютерных системах. В связи с этим задача подбора оптимальной конфигурации компьютера становится все более актуальной как для индивидуальных пользователей, так и для корпоративного сектора. Правильный выбор компонентов компьютера позволяет не только сэкономить средства, но и обеспечить высокую производительность и долговечность системы.

Целью данной курсовой работы является разработка веб-приложения для подбора конфигурации компьютера с использованием Java Spring. Веб-приложение позволит пользователям легко и быстро сформировать оптимальную конфигурацию компьютера, исходя из их индивидуальных потребностей и бюджета.

Использование Java Spring в разработке данного веб-приложения обусловлено его широкими возможностями и преимуществами, такими как высокая производительность, масштабируемость и гибкость. Spring Framework предоставляет богатый набор инструментов для создания надежных и устойчивых веб-приложений, что делает его идеальным выбором для реализации проекта данного типа.

В ходе работы будет рассмотрено архитектурное решение проекта, описаны основные этапы разработки и тестирования, а также проведен анализ результатов и перспективы дальнейшего развития веб-приложения.

Общая диаграмма классов представлена на Рисунке 1:



Рисунок 1 – Диаграмма классов в IDE

Общая модель БД представлена на Рисунке 2:

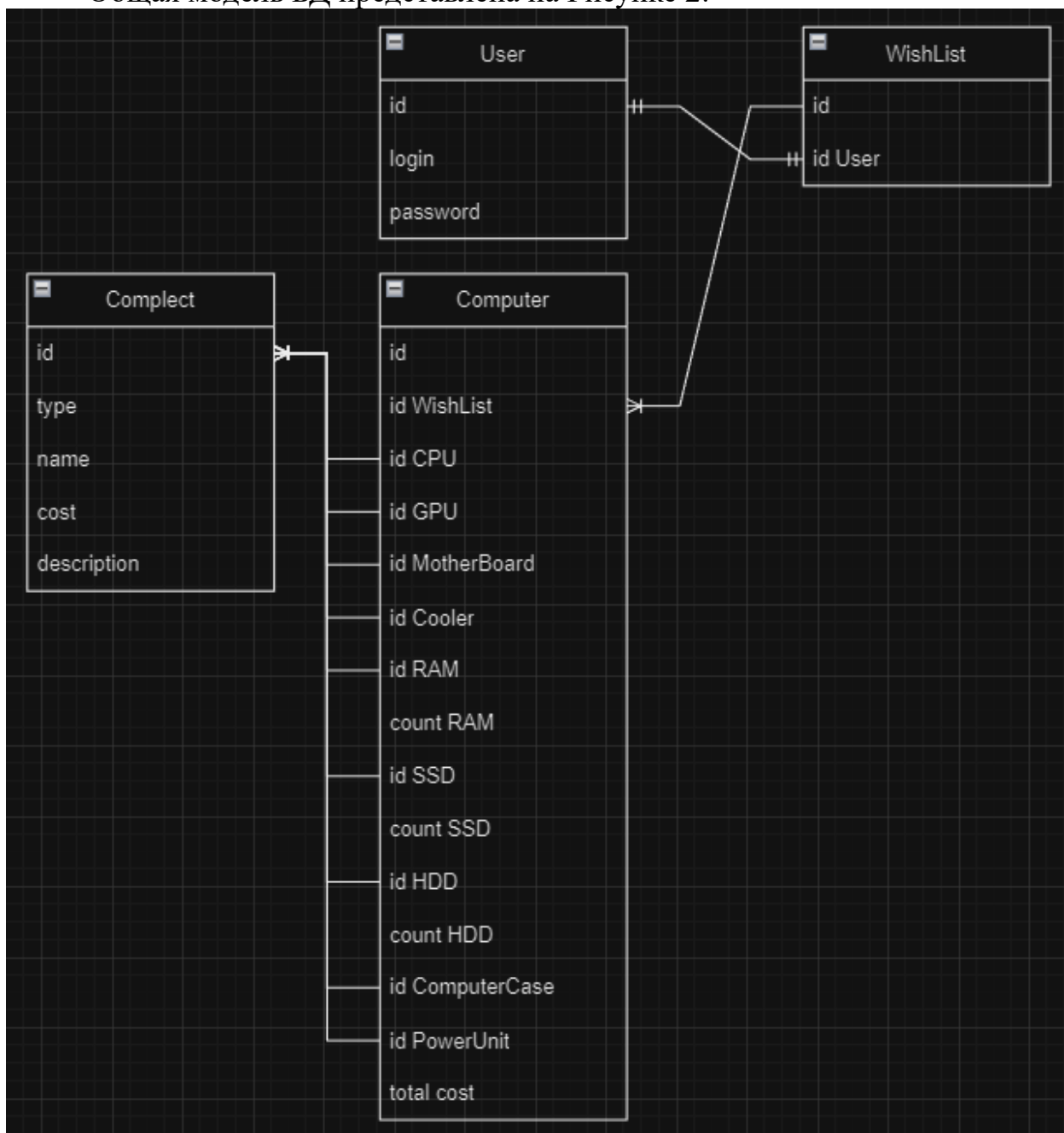


Рисунок 2 – Схема БД

Проектирование базы данных

Проектирование базы данных включает в себя разработку логической и физической модели данных.

Логическая модель

На логическом уровне определяются сущности, их атрибуты и взаимосвязи между ними. На основе предоставленной схемы определены следующие сущности:

- **User**
 - id (PRIMARY KEY)
 - login
 - password
- **WishList**
 - id (PRIMARY KEY)
 - idUser (FOREIGN KEY)
- **Computer**
 - id (PRIMARY KEY)
 - idWishList (FOREIGN KEY)
 - idCPU (FOREIGN KEY)
 - idGPU (FOREIGN KEY)
 - idMotherBoard (FOREIGN KEY)
 - idCooler (FOREIGN KEY)
 - idRAM (FOREIGN KEY)
 - countRAM
 - idSSD (FOREIGN KEY)
 - countSSD
 - idHDD (FOREIGN KEY)
 - countHDD
 - idComputerCase (FOREIGN KEY)
 - idPowerUnit (FOREIGN KEY)
 - totalCost
- **Complect**
 - id (PRIMARY KEY)
 - type
 - name
 - cost
 - description

Для проектирования была использована СУБД PostgreSQL

В данном случае схема соответствует третьей нормальной форме (3NF), так как: Все атрибуты атомарны, Все неключевые атрибуты зависят от первичного ключа, Нет транзитивных зависимостей.

Модели

Класс Complect

Класс Complect представляет сущность "Комплекующий" в системе подбора конфигурации компьютера. Он описывает отдельный компонент компьютера, такой как процессор, видеокарта, материнская плата и другие комплекующие. Класс содержит основные атрибуты комплекующего и взаимосвязи с другими классами, что позволяет реализовать необходимые функции для работы веб-приложения.

Импортируемые библиотеки и аннотации

- `com.fasterxml.jackson.annotation.JsonIgnore` и `com.fasterxml.jackson.annotation.JsonManagedReference`: Используются для управления сериализацией JSON, чтобы избежать циклических зависимостей и рекурсивных ссылок.
- `jakarta.persistence.*`: Набор аннотаций JPA (Java Persistence API) для маппинга класса на таблицу базы данных и указания различных аспектов поведения полей класса.
- `lombok.*`: Библиотека Lombok для автоматической генерации стандартных методов, таких как геттеры, сеттеры, конструкторы и другие.

Аннотации на уровне класса

- `@AllArgsConstructor`, `@NoArgsConstructor`: Lombok-аннотации для автоматического создания конструктора с аргументами и конструктора без аргументов соответственно.
- `@Getter`, `@Setter`: Lombok-аннотации для генерации геттеров и сеттеров для всех полей класса.
- `@Entity(name = "Complect")`: Указывает, что класс является сущностью JPA и должен быть маппирован на таблицу Complect в базе данных.
- `@Table(name = "Complect")`: Задаёт имя таблицы в базе данных, с которой ассоциируется данный класс.

Поля класса

- `id`: Уникальный идентификатор комплекующего, автоматически генерируемый с использованием последовательности.
- `type`: Тип комплекующего (например, "CPU", "GPU").
- `name`: Название комплекующего.
- `cost`: Стоимость комплекующего.
- `description`: Описание комплекующего.

Класс Computer

Класс Computer представляет сущность "Компьютер" в системе подбора конфигурации компьютера. Он описывает конфигурацию компьютера, включающую различные комплектующие, такие как процессор, видеокарта, оперативная память и другие компоненты. Класс содержит атрибуты, соответствующие этим компонентам, и обеспечивает их взаимосвязь с другими сущностями, такими как WishList и Complect.

Импортируемые библиотеки и аннотации

- `com.fasterxml.jackson.annotation.JsonBackReference`: Используется для управления сериализацией JSON и предотвращения циклических зависимостей.
- `jakarta.persistence.*`: Набор аннотаций JPA (Java Persistence API) для маппинга класса на таблицу базы данных и указания различных аспектов поведения полей класса.
- `lombok.*`: Библиотека Lombok для автоматической генерации стандартных методов, таких как геттеры, сеттеры, конструкторы и другие.

Аннотации на уровне класса

- `@AllArgsConstructor`, `@NoArgsConstructor`: Lombok-аннотации для автоматического создания конструктора с аргументами и конструктора без аргументов соответственно.
- `@Getter`, `@Setter`: Lombok-аннотации для генерации геттеров и сеттеров для всех полей класса.
- `@Entity(name = "Computer")`: Указывает, что класс является сущностью JPA и должен быть маппирован на таблицу Computer в базе данных.
- `@Table(name = "Computer")`: Задаёт имя таблицы в базе данных, с которой ассоциируется данный класс.

Поля класса

- `id`: Уникальный идентификатор конфигурации компьютера, автоматически генерируемый с использованием последовательности.
- `wishList`: Ссылка на сущность WishList, к которой относится данная конфигурация.
- `cpu`, `gpu`, `motherBoard`, `cooler`, `ram`, `ssd`, `hdd`, `computerCase`, `powerUnit`: Ссылки на сущность Complect, представляющие различные компоненты компьютера.
- `countRAM`, `CountSSD`, `CountHDD`: Количество соответствующих компонентов в конфигурации.

- `totalCost`: Общая стоимость конфигурации компьютера.

Взаимосвязи с другими сущностями

- Каждое поле, представляющее компонент компьютера, аннотировано `@ManyToOne`, что указывает на связь "многие-к-одному" с сущностью `Complect`. Это означает, что множество конфигураций могут содержать один и тот же компонент.
- `fetch = FetchType.EAGER`: Определяет стратегию загрузки данных. `FetchType.EAGER` означает, что связанные данные будут загружены вместе с основной сущностью.
- `@JoinColumn(name = "column_name")`: Указывает имя колонки в таблице базы данных, которая будет использоваться для хранения внешнего ключа.

Класс User

Класс `User` представляет пользователя в системе подбора конфигурации компьютера. Этот класс реализует интерфейс `UserDetails` из `Spring Security`, что позволяет использовать его для аутентификации и авторизации в приложении. Класс включает основные атрибуты пользователя, методы, необходимые для интеграции с `Spring Security`, и связи с другими сущностями, такими как `UserRole` и `WishList`.

Импортируемые библиотеки и аннотации

- `com.fasterxml.jackson.annotation.JsonIgnore`: Используется для исключения полей из сериализации JSON, чтобы избежать рекурсивных ссылок и утечки чувствительных данных.
- `jakarta.persistence.*`: Набор аннотаций JPA (Java Persistence API) для маппинга класса на таблицу базы данных и указания различных аспектов поведения полей класса.
- `lombok.*`: Библиотека Lombok для автоматической генерации стандартных методов, таких как геттеры, сеттеры, конструкторы и другие.
- `org.springframework.security.core.GrantedAuthority`, `org.springframework.security.core.userdetails.UserDetails`: Интерфейсы `Spring Security` для определения пользовательских данных, необходимых для аутентификации и авторизации.

Аннотации на уровне класса

- `@AllArgsConstructor`, `@NoArgsConstructor`: Lombok-аннотации для автоматического создания конструктора с аргументами и конструктора без аргументов соответственно.

- `@Getter`, `@Setter`: Lombok-аннотации для генерации геттеров и сеттеров для всех полей класса.
- `@ToString`: Lombok-аннотация для автоматического создания метода `toString()`.
- `@Accessors(chain = true)`: Lombok-аннотация для включения методов сеттера, возвращающих `this`, что позволяет использовать цепочку вызовов.
- `@Entity(name = "App_User")`: Указывает, что класс является сущностью JPA и должен быть маппирован на таблицу `App_User` в базе данных.
- `@Table(name = "App_User")`: Задаёт имя таблицы в базе данных, с которой ассоциируется данный класс.

Поля класса

- `id`: Уникальный идентификатор пользователя, автоматически генерируемый с использованием последовательности.
- `username`: Имя пользователя.
- `password`: Пароль пользователя.
- `expired`: Флаг, указывающий, истек ли срок действия учетной записи.
- `locked`: Флаг, указывающий, заблокирована ли учетная запись.
- `enabled`: Флаг, указывающий, включена ли учетная запись.

Взаимосвязи с другими сущностями

- `userRoles`: Список ролей пользователя, аннотированный `@OneToMany`. Связь указывает, что один пользователь может иметь несколько ролей.
- `wishList`: Связь с сущностью `WishList`, представляющей список желаемых конфигураций пользователя. Аннотация `@OneToOne` указывает, что один пользователь может иметь только один список желаемых конфигураций.

Реализация интерфейса `UserDetails`

- Класс `User` реализует методы интерфейса `UserDetails`, которые необходимы для интеграции с `Spring Security`:
 - `getAuthorities()`: Возвращает коллекцию прав (ролей) пользователя.
 - `getPassword()`: Возвращает пароль пользователя.
 - `getUsername()`: Возвращает имя пользователя.

- `isAccountNonExpired()`, `isAccountNonLocked()`, `isCredentialsNonExpired()`, `isEnabled()`: Методы, возвращающие статус учетной записи пользователя (активна/неактивна).

Класс `UserAuthority`

Класс `UserAuthority` представляет собой перечисление (enum), которое реализует интерфейс `GrantedAuthority` из `Spring Security`. Этот класс используется для определения прав (авторизаций), которые могут быть присвоены пользователю в системе. Права определяют, какие действия пользователь может выполнять в системе, такие как размещение заказов, управление заказами и полные права.

Импортируемые библиотеки и интерфейсы

- `org.springframework.security.core.GrantedAuthority`: Интерфейс `Spring Security`, который представляет собой право или авторизацию, присваиваемую пользователю.

Перечисление прав

- Перечисление `UserAuthority` содержит три значения, каждое из которых представляет определенный уровень доступа:
 - `PLACE_ORDERS`: Право на размещение заказов.
 - `MANAGE_ORDERS`: Право на управление заказами.
 - `FULL`: Полные права, которые могут включать в себя все доступные действия.

Реализация метода `getAuthority`

- Метод `getAuthority` из интерфейса `GrantedAuthority` возвращает строковое представление права. В данном случае он возвращает имя константы перечисления, которое совпадает с ее названием.

Класс `UserRole`

Класс `UserRole` представляет роль пользователя в системе подбора конфигурации компьютера. Он содержит информацию о конкретном праве (авторизации), которое присвоено пользователю. Класс включает связи с сущностями `User` и `UserAuthority` для реализации механизма ролей и авторизаций.

Импортируемые библиотеки и аннотации

- `com.fasterxml.jackson.annotation.JsonIgnore`: Используется для исключения полей из сериализации JSON, чтобы избежать рекурсивных ссылок и утечки чувствительных данных.
- `jakarta.persistence.*`: Набор аннотаций JPA (Java Persistence API) для маппинга класса на таблицу базы данных и указания различных аспектов поведения полей класса.
- `lombok.*`: Библиотека Lombok для автоматической генерации стандартных методов, таких как геттеры, сеттеры, конструкторы и другие.

Аннотации на уровне класса

- `@Data`: Lombok-аннотация для генерации всех стандартных методов, таких как геттеры, сеттеры, `toString()`, `equals()` и `hashCode()`.
- `@Table(name = "user_roles")`: Задаёт имя таблицы в базе данных, с которой ассоциируется данный класс.
- `@Entity(name = "user_roles")`: Указывает, что класс является сущностью JPA и должен быть маппирован на таблицу `user_roles` в базе данных.
- `@AllArgsConstructor`: Lombok-аннотация для автоматического создания конструктора с аргументами для всех полей.
- `@NoArgsConstructor`: Lombok-аннотация для автоматического создания конструктора без аргументов.

Поля класса

- `id`: Уникальный идентификатор роли пользователя, автоматически генерируемый с использованием последовательности.
- `userAuthority`: Перечисление `UserAuthority`, представляющее право или авторизацию пользователя.
- `user`: Связь с сущностью `User`, представляющей пользователя, которому принадлежит данная роль.

Аннотации на уровне полей

- `@Id`: Указывает, что поле `id` является первичным ключом.
- `@GeneratedValue(generator = "user_role_id_seq", strategy = GenerationType.SEQUENCE)`: Указывает, что значение для поля `id` будет автоматически генерироваться с использованием последовательности `user_role_id_seq`.
- `@SequenceGenerator(name = "user_role_id_seq", sequenceName = "user_role_id_seq", allocationSize = 1)`: Определяет генератор последовательности для поля `id`.
- `@Enumerated`: Указывает, что поле `userAuthority` должно быть маппировано как перечисление.

- `@JsonIgnore`: Указывает, что поле `user` не должно быть сериализовано в JSON.
- `@ManyToOne`: Указывает, что существует связь многие-к-одному с сущностью `User`.
- `@JoinColumn(name = "user_id")`: Указывает имя столбца в таблице `user_roles`, который хранит внешний ключ для связи с таблицей `User`.

Класс `WishList`

Класс `WishList` представляет собой сущность, которая хранит список желаемых конфигураций компьютеров для пользователя в системе подбора конфигурации. Он связывается с пользователем через отношение один-к-одному и содержит список компьютеров, связанных с этим списком желаемого.

Импортируемые библиотеки и аннотации

- `com.fasterxml.jackson.annotation.*`: Набор аннотаций для управления сериализацией и десериализацией JSON с использованием Jackson.
- `jakarta.persistence.*`: Набор аннотаций JPA для маппинга класса на таблицу базы данных и указания различных аспектов поведения полей класса.
- `lombok.*`: Библиотека Lombok для автоматической генерации стандартных методов, таких как геттеры, сеттеры, конструкторы и другие.

Аннотации на уровне класса

- `@Data`: Lombok-аннотация для генерации всех стандартных методов, таких как геттеры, сеттеры, `toString()`, `equals()` и `hashCode()`.
- `@AllArgsConstructor`: Lombok-аннотация для автоматического создания конструктора с аргументами для всех полей.
- `@NoArgsConstructor`: Lombok-аннотация для автоматического создания конструктора без аргументов.
- `@Getter` и `@Setter`: Lombok-аннотации для генерации геттеров и сеттеров для всех полей.
- `@Entity(name = "WishList")`: Указывает, что класс является сущностью JPA и должен быть маппирован на таблицу `WishList` в базе данных.
- `@Table(name = "WishList")`: Задаёт имя таблицы в базе данных, с которой ассоциируется данный класс.
- `@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")`: Используется для управления циклическими ссылками при сериализации JSON.

Поля класса

- `id`: Уникальный идентификатор списка желаемого, автоматически генерируемый с использованием последовательности.
- `user`: Связь с сущностью `User`, представляющей пользователя, которому принадлежит этот список желаемого.
- `computers`: Список конфигураций компьютеров, связанных с этим списком желаемого.

Аннотации на уровне полей

- `@Id`: Указывает, что поле `id` является первичным ключом.
- `@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "wishlist_seq_gen")`: Указывает, что значение для поля `id` будет автоматически генерироваться с использованием последовательности `wishlist_seq`.
- `@SequenceGenerator(name = "wishlist_seq_gen", sequenceName = "wishlist_seq", allocationSize = 1)`: Определяет генератор последовательности для поля `id`.
- `@Column(name = "id")`: Задаёт имя столбца в таблице базы данных для поля `id`.
- `@JsonIgnore`: Указывает, что поле не должно быть сериализовано в JSON, чтобы избежать рекурсивных ссылок.
- `@OneToOne` и `@JoinColumn(name = "user_id")`: Указывает отношение один-к-одному с таблицей `User`.
- `@OneToMany(mappedBy = "wishlist", fetch = FetchType.EAGER)`: Указывает отношение один-ко-многим с таблицей `Computer`.

Репозитории

В проекте по созданию веб-приложения для подбора конфигурации компьютера на базе Java Spring были использованы несколько репозиторий для взаимодействия с базой данных. Репозитории предоставляют абстракцию для операций CRUD (создание, чтение, обновление, удаление) и позволяют легко управлять данными сущностями. В данном проекте использовались следующие репозитории:

1. ComplectRepository
2. ComputerRepository
3. UserRepository
4. UserRolesRepository
5. WishListRepository

ComplectRepository

Репозиторий ComplectRepository используется для работы с сущностью Complect, которая представляет собой компонент компьютерной системы (CPU, GPU, RAM и т.д.).

ComputerRepository

Репозиторий ComputerRepository предназначен для работы с сущностью Computer, представляющей собой конфигурацию компьютера.

UserRepository

Репозиторий UserRepository используется для работы с сущностью User, которая представляет пользователя системы.

UserRolesRepository

Репозиторий UserRolesRepository предназначен для работы с сущностью UserRole, представляющей роли пользователя в системе.

WishListRepository

Репозиторий WishListRepository используется для работы с сущностью WishList, которая представляет список желаемых конфигураций компьютеров для пользователя.

Сервис

ComplectServiceImpl

Класс `ComplectServiceImpl` представляет собой сервисный слой приложения, отвечающий за выполнение операций, связанных с сущностью `Complect`. Этот сервис обеспечивает взаимодействие между контроллерами и репозиторием, выполняя бизнес-логику и операции CRUD (создание, чтение, обновление, удаление) для компонентов компьютерной конфигурации.

Импортируемые библиотеки и аннотации

- `com.example.CompConf.model.Complect`: Импортирует сущность `Complect`.
- `com.example.CompConf.repository.ComplectRepository`: Импортирует репозиторий для сущности `Complect`.
- `lombok.RequiredArgsConstructor`: Аннотация Lombok для автоматического создания конструктора с аргументами для всех финальных полей.
- `org.springframework.stereotype.Service`: Аннотация Spring для указания, что класс является сервисом.

Аннотации на уровне класса

- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса. Это позволяет внедрять зависимости через конструктор.
- `@Service`: Аннотация Spring, которая указывает, что данный класс является сервисным компонентом.

Поля класса

- `private final ComplectRepository complectRepository`: Поле, представляющее репозиторий для работы с сущностью `Complect`. Оно помечено как `final`, что означает, что оно должно быть инициализировано только через конструктор.

Методы класса

- `getComplectById(Long id)`: Метод для получения компонента по его идентификатору.
- `registerComplect(Complect complect)`: Метод для регистрации нового компонента в базе данных.

- `getCPU()`, `getGPU()`, `getMotherBoard()`, `getCooler()`, `getRAM()`, `getSSD()`, `getHDD()`, `getComputerCase()`, `getPowerUnit()`: Методы для получения компонентов по типам (CPU, GPU, MotherBoard и т.д.).

ComputerServiceImpl

Класс `ComputerServiceImpl` представляет собой сервисный слой, который выполняет операции, связанные с сущностью `Computer`. Этот сервис обеспечивает взаимодействие между контроллерами и репозиториями, выполняя бизнес-логику и операции CRUD для конфигураций компьютеров.

Импортируемые библиотеки и аннотации

- `com.example.CompConf.model.Complect`: Импортирует сущность `Complect`.
- `com.example.CompConf.model.Computer`: Импортирует сущность `Computer`.
- `com.example.CompConf.repository.ComplectRepository`: Импортирует репозиторий для сущности `Complect`.
- `com.example.CompConf.repository.ComputerRepository`: Импортирует репозиторий для сущности `Computer`.
- `lombok.RequiredArgsConstructor`: Аннотация Lombok для автоматического создания конструктора с аргументами для всех финальных полей.
- `org.springframework.stereotype.Service`: Аннотация Spring для указания, что класс является сервисом.

Аннотации на уровне класса

- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса. Это позволяет внедрять зависимости через конструктор.
- `@Service`: Аннотация Spring, которая указывает, что данный класс является сервисным компонентом.

Поля класса

- `private final ComputerRepository computerRepository`: Поле, представляющее репозиторий для работы с сущностью `Computer`. Оно помечено как `final`, что означает, что оно должно быть инициализировано только через конструктор.
- `private final ComplectRepository complectRepository`: Поле, представляющее репозиторий для работы с сущностью `Complect`.

Методы класса

- `createComputer(Computer computer)`: Метод для создания и сохранения новой конфигурации компьютера в базе данных.
- `deleteComputer(Long id)`: Метод для удаления конфигурации компьютера по его идентификатору.
- `updateComputer(Long id, Computer computerDetails)`: Метод для обновления конфигурации компьютера по его идентификатору с новыми данными.
- `getComputerById(Long id)`: Метод для получения конфигурации компьютера по его идентификатору.

UserServiceImpl

Класс `UserServiceImpl` реализует интерфейсы `UserService` и `UserDetailsService` и предоставляет методы для управления пользователями и их ролями. Этот сервис также обеспечивает интеграцию с Spring Security для аутентификации и авторизации пользователей.

Импортируемые библиотеки и аннотации

- `com.example.CompConf.exceptions.UsernameAlreadyExistsException`: Импортирует пользовательское исключение для обработки ситуации, когда имя пользователя уже существует.
- `com.example.CompConf.model.*`: Импортирует модели данных.
- `com.example.CompConf.repository.UserRepository`: Импортирует репозиторий для сущности `User`.
- `com.example.CompConf.repository.UserRolesRepository`: Импортирует репозиторий для сущности `UserRole`.
- `jakarta.transaction.Transactional`: Импортирует аннотацию для обеспечения транзакционности методов.
- `lombok.RequiredArgsConstructor`: Аннотация Lombok для автоматического создания конструктора с аргументами для всех финальных полей.
- `org.springframework.security.core.userdetails.UserDetails`: Импортирует интерфейс для предоставления основных данных о пользователе.
- `org.springframework.security.core.userdetails.UserDetailsService`: Импортирует интерфейс для загрузки данных о пользователе по имени пользователя.
- `org.springframework.security.core.userdetails.UsernameNotFoundException`: Импортирует исключение для обработки ситуации, когда пользователь не найден.

- `org.springframework.security.crypto.password.PasswordEncoder`: Импортирует интерфейс для кодирования паролей.
- `org.springframework.stereotype.Service`: Аннотация Spring для указания, что класс является сервисом.

Аннотации на уровне класса

- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса. Это позволяет внедрять зависимости через конструктор.
- `@Service`: Аннотация Spring, которая указывает, что данный класс является сервисным компонентом.

Поля класса

- `private final UserRolesRepository userRolesRepository`: Поле, представляющее репозиторий для работы с сущностью `UserRole`.
- `private final UserRepository userRepository`: Поле, представляющее репозиторий для работы с сущностью `User`.
- `private final PasswordEncoder passwordEncoder`: Поле для кодирования паролей.

Методы класса

- `registration(String username, String password)`: Метод для регистрации нового пользователя. Он проверяет, существует ли уже пользователь с таким именем, и если нет, создает нового пользователя и назначает ему роль `PLACE_ORDERS`.
- `loadUserByUsername(String username)`: Метод для загрузки пользователя по его имени пользователя. Если пользователь не найден, выбрасывается исключение `UsernameNotFoundException`.

WishListServiceImpl

Класс `WishListServiceImpl` реализует интерфейс `WishListService` и предоставляет методы для управления списками желаний пользователей. Этот сервис позволяет добавлять и удалять элементы из списка желаний, а также получать список компьютеров и списков желаний для конкретного пользователя.

Импортируемые библиотеки и аннотации

- `com.example.CompConf.exceptions.UsernameNotFoundException`: Импортирует пользовательское исключение для обработки ситуации, когда пользователь не найден.
- `com.example.CompConf.model.Computer`: Импортирует модель данных для компьютера.
- `com.example.CompConf.model.User`: Импортирует модель данных для пользователя.
- `com.example.CompConf.model.WishList`: Импортирует модель данных для списка желаний.
- `com.example.CompConf.repository.UserRepository`: Импортирует репозиторий для сущности User.
- `com.example.CompConf.repository.WishListRepository`: Импортирует репозиторий для сущности WishList.
- `lombok.RequiredArgsConstructor`: Аннотация Lombok для автоматического создания конструктора с аргументами для всех финальных полей.
- `org.springframework.stereotype.Service`: Аннотация Spring для указания, что класс является сервисом.

Аннотации на уровне класса

- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса. Это позволяет внедрять зависимости через конструктор.
- `@Service`: Аннотация Spring, которая указывает, что данный класс является сервисным компонентом.

Поля класса

- `private final WishListRepository wishListRepository`: Поле, представляющее репозиторий для работы с сущностью WishList.
- `private final UserRepository userRepository`: Поле, представляющее репозиторий для работы с сущностью User.

Методы класса

- `addToWishList(WishList wishList)`: Метод для добавления нового списка желаний.
- `getComputerByUserId(Long id)`: Метод для получения списка компьютеров по идентификатору пользователя.
- `getWishListByUserId(Long userId)`: Метод для получения списка желаний по идентификатору пользователя.
- `removeFromWishList(WishList wishList)`: Метод для удаления списка желаний.

Контроллеры

ComplectGetController

Класс ComplectGetController является контроллером Spring, который обрабатывает HTTP-запросы, связанные с компонентами компьютера (Complect). Этот контроллер предоставляет методы для получения компонентов по их типу и идентификатору.

Импортируемые библиотеки и аннотации

- `com.example.CompConf.model.Complect`: Импортирует модель данных для компонента.
- `com.example.CompConf.service.ComplectService`: Импортирует сервис для работы с компонентами.
- `lombok.RequiredArgsConstructor`: Аннотация Lombok для автоматического создания конструктора с аргументами для всех финальных полей.
- `lombok.extern.slf4j.Slf4j`: Аннотация Lombok для логирования.
- `org.springframework.http.HttpStatus`: Импортирует статусы HTTP-ответов.
- `org.springframework.http.ResponseEntity`: Импортирует класс для создания HTTP-ответов.
- `org.springframework.web.bind.annotation.GetMapping`: Аннотация для обработки GET-запросов.
- `org.springframework.web.bind.annotation.PathVariable`: Аннотация для извлечения переменных из URL.
- `org.springframework.web.bind.annotation.RequestMapping`: Аннотация для указания базового пути для контроллера.
- `org.springframework.web.bind.annotation.RestController`: Аннотация, указывающая, что класс является REST-контроллером.

Аннотации на уровне класса

- `@RestController`: Аннотация Spring, указывающая, что данный класс является REST-контроллером.
- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса.
- `@RequestMapping("/complect")`: Указывает базовый путь для всех методов контроллера.
- `@Slf4j`: Аннотация для логирования.

Поля класса

- `private final ComplectService complectService:` Поле, представляющее сервис для работы с компонентами.
- `getComplectById(Long id):` Метод для получения компонента по его идентификатору.

```
@GetMapping("/{id}")
public ResponseEntity<Complect> getComplectById(@PathVariable Long id)
{
    return complectService.getComplectById(id)
        .map(ResponseEntity::ok)
        .orElse(ResponseEntity.notFound().build());
}
```

`getCPU():` Метод для получения всех процессоров (CPU).

```
@GetMapping("/CPU")
public ResponseEntity<List<Complect>> getCPU() {
    List<Complect> complects = complectService.getCPU();
    return ResponseEntity.ok(complects);
}
```

Остальные методы работают по аналогии

ComplectPostController

Этот класс имеет следующую структуру:

REST-аннотации:

- `@RestController:` Указывает, что данный класс является REST-контроллером, который обрабатывает HTTP-запросы и возвращает данные в формате JSON.
- `@RequestMapping("/complect"):` Устанавливает базовый путь для всех методов контроллера.

Аннотации Lombok:

- `@RequiredArgsConstructor:` Генерирует конструктор с аргументами для всех финальных полей класса. В данном случае, это поле `complectService`.
- `@Slf4j:` Аннотация для использования логгера, который легко генерируется Lombok.

Поле класса:

- `private final ComplectService complectService`: Представляет сервис для работы с компонентами. Это финальное поле, и его значение инициализируется через конструктор.

ComputerController

Описание класса:

- `@RestController`: Указывает, что класс является REST-контроллером и ответы на его методы будут сериализованы в формат JSON.
- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.
- `@RequestMapping("/computer")`: Устанавливает базовый путь для всех методов контроллера.

Поля класса:

- `private final ComputerService computerService`;: Поле, представляющее сервис для работы с компьютерами.

Методы класса:

- `getComputerById`: Обработывает GET-запросы для получения информации о компьютере по его идентификатору.
- `createComputer`: Обработывает POST-запросы для создания нового компьютера.
- `updateComputer`: Обработывает PUT-запросы для обновления информации о компьютере по его идентификатору.
- `deleteComputer`: Обработывает DELETE-запросы для удаления компьютера по его идентификатору.

Каждый из этих методов выполняет соответствующую операцию на стороне сервера и возвращает соответствующий HTTP-ответ в зависимости от результата операции.

RegistrationController

Описание класса:

- `@RestController`: Указывает, что класс является REST-контроллером и ответы на его методы будут сериализованы в формат JSON.
- `@RequiredArgsConstructor`: Аннотация Lombok, которая автоматически генерирует конструктор с аргументами для всех финальных полей класса.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.
- `@RequestMapping("/registration")`: Устанавливает базовый путь для всех методов контроллера.

Поля класса:

- `private final UserService userService;` Поле, представляющее сервис для работы с пользователями.

Методы класса:

- `registration`: Обработывает POST-запросы для регистрации нового пользователя. Принимает параметры `username` и `password` через `@RequestParam`, вызывает соответствующий метод сервиса `userService.registration` для регистрации пользователя и возвращает ответ с кодом 200 (OK), если операция завершена успешно.

UserController

Описание класса:

- `@RestController`: Обозначает, что этот класс является REST-контроллером, который обрабатывает HTTP-запросы и возвращает данные в формате JSON.
- `@RequiredArgsConstructor`: Аннотация Lombok, генерирующая конструктор с аргументами для всех финальных полей класса.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.
- `@RequestMapping("/user")`: Устанавливает базовый путь для всех методов контроллера.

Поля класса:

- `private final WishListServiceImpl wishListService;` Поле, представляющее сервис для работы с списком желаемых товаров.

Методы класса:

- `addToWishList`: Обрабатывает POST-запросы для добавления товара в список желаемых. Принимает объект `WishList` в теле запроса, передает его сервису `wishListService` для добавления и возвращает ответ с кодом 200 (OK).
- `removeFromWishList`: Обрабатывает DELETE-запросы для удаления товара из списка желаемых. Принимает объект `WishList` в теле запроса, передает его сервису `wishListService` для удаления и возвращает ответ с кодом 200 (OK).
- `getComputersByUserId`: Обрабатывает GET-запросы для получения списка компьютеров пользователя по его идентификатору. Вызывает соответствующий метод сервиса для получения списка и возвращает ответ с кодом 200 (OK) и списком компьютеров.
- `getWishListByUserId`: Обрабатывает GET-запросы для получения списка желаемых товаров пользователя по его идентификатору. Вызывает соответствующий метод сервиса для получения списка и возвращает ответ с кодом 200 (OK) и списком желаемых товаров.

Exceptions

ComplectNotFoundException

- Этот класс представляет исключение, которое может возникнуть, если комплект (Complect) не найден.
- Он расширяет класс RuntimeException, что означает, что это непроверяемое исключение, которое может возникнуть во время выполнения программы.
- Используется для обработки ситуации, когда запрос на доступ к комплекту завершился неудачно из-за его отсутствия.

UsernameAlreadyExistsException

- Этот класс представляет исключение, которое возникает, если пользователь с заданным именем пользователя уже существует в системе.
- Также он расширяет класс RuntimeException.
- Это исключение используется для предотвращения создания пользователя с именем, которое уже используется другим пользователем в системе.

UsernameNotFoundException

- Этот класс представляет исключение, которое возникает, если пользователь с указанным именем пользователя не найден в системе.
- Также он расширяет класс RuntimeException.
- Это исключение используется для обработки ситуации, когда запрос на доступ к пользователю завершился неудачно из-за его отсутствия в системе.

Конфигурация

SpringSecurityConfiguration

Описание класса:

- `@Configuration`: Обозначает, что этот класс является конфигурационным классом Spring.
- `@EnableWebSecurity`: Аннотация, позволяющая Spring настроить глобальную безопасность веб-приложения.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.

Методы класса:

- `filterChain`: Этот метод настраивает цепочку фильтров безопасности Spring.
- `http.authorizeHttpRequests`: Настройка прав доступа к различным URL-адресам в зависимости от роли пользователя.
- `formLogin`: Настройка формы входа.
- `csrf`: Отключение защиты CSRF.
- `passwordEncoder`: Создает бин `PasswordEncoder` для хеширования паролей. Используется алгоритм хеширования BCrypt.

Логирование

ControllerLogAspect

Описание класса:

- `@Component`: Обозначает, что класс является компонентом Spring и должен быть обнаружен и сконфигурирован в контексте приложения.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.
- `@Aspect`: Обозначает, что этот класс является аспектом аспектно-ориентированного программирования (AOP).

Методы класса:

- `callController`: Это определение точки среза (pointcut), которая сопоставляется с любым публичным методом в любом классе в пакете `com.example.CompConf.controller`.
- `beforeCallController`: Этот метод выполняется перед вызовом любого метода, сопоставленного точкой среза `callController`. Он логирует информацию о вызываемом методе и переданных ему аргументах.
- `afterCallController`: Этот метод выполняется после возврата значения из любого метода, сопоставленного точкой среза `callController`. Он логирует информацию о вызываемом методе и возвращаемом им значении.

MyBeanPostProcessor

Описание класса:

- `@Component`: Обозначает, что этот класс является компонентом Spring и должен быть обнаружен и сконфигурирован в контексте приложения.
- `@Slf4j`: Аннотация для использования логгера, который легко генерируется Lombok.
- `@Configuration`: Обозначает, что класс является конфигурационным классом Spring.
- `@EnableAspectJAutoProxy`: Аннотация, которая включает поддержку аспектно-ориентированного программирования (AOP) с использованием AspectJ для созданных бинов.

Методы класса:

- `postProcessBeforeInitialization`: Метод, вызываемый Spring перед инициализацией бина. Здесь он логирует информацию о создаваемом бине и возвращает его без изменений.

- `postProcessAfterInitialization`: Метод, вызываемый Spring после инициализации бина. Здесь он также логирует информацию о созданном бине и возвращает его без изменений.

Эти методы могут быть использованы для выполнения дополнительных действий над бинами перед и после их инициализации.

Также были использованы changelog`и, структура которых представлена на Рисунке 3:

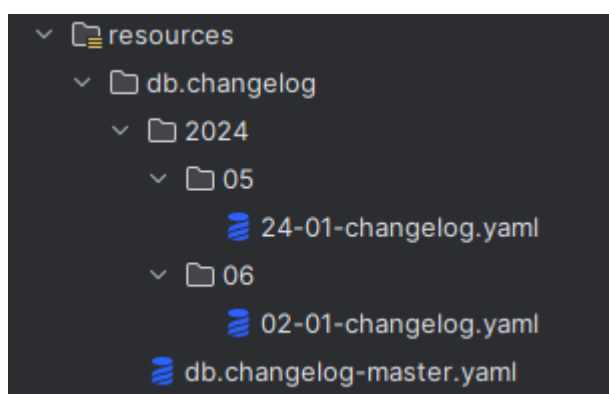


Рисунок 3 – Структура Changelog

Сами changelog описаны в приложении

Тесты имеют структуру, которая представлена на Рисунке 4:

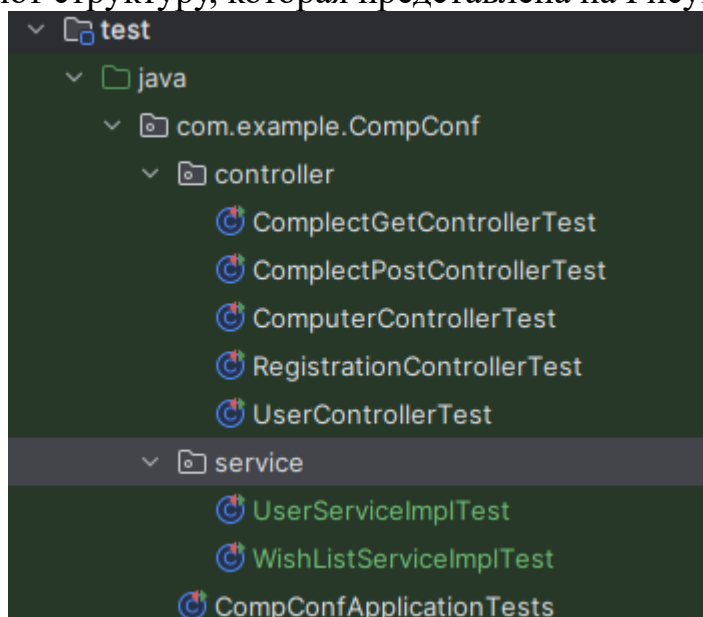


Рисунок 4 – Структура тестов

Список endpoint`ов представлен на Рисунке 5:





















CompConf.main		
	/complect/CPU [GET]	ComplectGetController
	/complect/ComputerCase [GET]	ComplectGetController
	/complect/Cooler [GET]	ComplectGetController
	/complect/GPU [GET]	ComplectGetController
	/complect/HDD [GET]	ComplectGetController
	/complect/MotherBoard [GET]	ComplectGetController
	/complect/PowerUnit [GET]	ComplectGetController
	/complect/RAM [GET]	ComplectGetController
	/complect/SSD [GET]	ComplectGetController
	/complect/{id} [GET]	ComplectGetController
	/complect [POST]	ComplectPostController
	/computer [POST]	ComputerController
	/computer/{id} [GET]	ComputerController
	/computer/{id} [PUT]	ComputerController
	/computer/{id} [DELETE]	ComputerController
	/registration [POST]	RegistrationController
	/user/wishlist [POST]	UserController
	/user/wishlist/{id} [DELETE]	UserController
	/user/{userId} [GET]	UserController
	/user/{userId}/wishlist [GET]	UserController

Рисунок 5 – Список endpoint'ов

Заключение

Целью настоящей курсовой работы было создание веб-приложения для подбора конфигурации компьютера, а также внедрение в него механизмов логирования и безопасности с использованием Spring Security. Для достижения этой цели были выполнены следующие задачи:

1. **Разработка функциональности подбора компьютерной конфигурации:** Реализованы различные компоненты приложения, позволяющие пользователям выбирать и настраивать компоненты компьютера, такие как процессор, видеокарта, оперативная память и другие.
2. **Управление компьютерными компонентами через REST API:** Созданы API-методы для добавления, удаления и редактирования компонентов, обеспечивая гибкость и удобство в управлении данными.
3. **Внедрение логирования:** Реализована система логирования с использованием библиотеки SLF4J, позволяющая отслеживать действия пользователей и выявлять проблемы в работе приложения.
4. **Обеспечение безопасности с помощью Spring Security:** Настроена система безопасности Spring Security для защиты данных и авторизации пользователей. Определены права доступа и настроены правила аутентификации для обеспечения безопасного доступа к приложению.
5. **Тестирование и обеспечение качества:** Проведено тестирование различных компонентов приложения, включая модульное, интеграционное и функциональное тестирование, чтобы обеспечить надежную работу приложения и предотвратить появление ошибок.
6. **Документирование и анализ результатов:** В ходе работы были составлены документы, описывающие архитектуру приложения, его функциональные возможности и особенности реализации. Проведен анализ результатов работы и выявлены дальнейшие направления развития приложения.

Результатом выполнения курсовой работы стало работоспособное веб-приложение, которое успешно реализует задачи подбора компьютерной конфигурации, обеспечивает безопасность данных пользователей и обладает надежной системой логирования для мониторинга и отладки приложения.

Полученный опыт в разработке веб-приложений с использованием Spring Framework и механизмов безопасности позволит эффективно применять полученные знания в будущих проектах и дальнейшем профессиональном развитии.

Приложение:

Ссылка на гит: <https://github.com/JDexMoment/CompConf>

Учебные материалы: <https://spring-projects.ru/guides/>