Geekbrains

**Создание интернет-магазина с системой рекомендаций на основе предпочтений пользователей «Магазин аквариумных растений» на Java Spring.**

Программа обучения: «Разработчик»

Специализация: «Программист»

Технологическая специализация: «Веб разработка на Java»

Грибов Сергей Александрович

Москва

2025

**Дипломный проект**

**«Создание** **интернет-магазина с системой рекомендаций на основе предпочтений пользователей «Магазин аквариумных растений» на Java Spring»**

Оглавление

[Глава 1. Введение 3](#_Toc203501215)

[Глава 2. Предпроектная стадия 5](#_Toc203501216)

[2.1. Порядок проведения работ 5](#_Toc203501217)

[2.2. Определение требований к приложению 6](#_Toc203501218)

[2.3. Выбор стека технологий 8](#_Toc203501219)

[2.4. Архитектура приложения 9](#_Toc203501220)

[Глава 3. Проектная стадия 12](#_Toc203501221)

[3.1. Проектирование архитектуры приложения 12](#_Toc203501222)

[3.2. Проектирование WEB – интерфейса приложения 28](#_Toc203501223)

[3.3. Проектирование базы данных 36](#_Toc203501224)

[3.4. Безопасность приложения 40](#_Toc203501225)

[3.5. Docker-контейнер 53](#_Toc203501226)

[Глава 4. Заключение 54](#_Toc203501227)

[Глава 5. Список литературы 55](#_Toc203501228)

# Введение

Настоящий дипломный проект это разработанное ПО - модель интернет-магазина, особенностью которого является статистическая составляющая, определяющая рейтинг товаров.

Направление деятельности, для которого создано данное ПО – это зообизнес.

ОКВЭД 47.76 «Торговля розничная цветами и другими растениями, семенами, удобрениями, домашними животными и кормами для домашних животных в специализированных магазинах».

Тематика, выбрана не случайно, так как аквариумистика – уже многие годы является хобби автора.

Основная функциональность приложения включает в себя:

**Регистрация и аутентификация. Необходимая функциональность любого WEB приложения с обратной связью.**

**Каталог растений**. Пользователь может ознакомиться с ассортиментом магазина. Выполнить выбор растения по типу, аквариумисты водные растения условно разделяют на типы (растения, почвопокровные, длинностебельные, мхи, папоротники, водоросли). Выполнить выбор по направлению (морское или пресное), по уровню сложности выращивания, и по названию биномена (имя рода + имя вида).

**Моя корзина**. Пользователь может выбрать растение и приступить к формированию заказа - положить товар в свою корзину. Так же может удалить товар из корзины. Посмотреть итоговую стоимость. Может сделать заказ.

**Мои заказы**. Это форма, где отображаются все заказы текущего пользователя. В каждом заказе есть номер корзины, итоговая стоимость, время создания и время закрытия. Пользователь может детально посмотреть состав каждого заказа.

**Просмотр всех пользователей (роль администратора)**. Функциональность доступна только администратору. Администратор сможет посмотреть контактные данные интересующего пользователя.

**Просмотр всех заказов (роль администратора)**. Функциональность доступна только администратору. Администратор сможет посмотреть все заказы всех пользователей. Может детально посмотреть каждый заказ. Может закрыть любой заказ.

**Рейтинг**. Как было упомянуто выше, в приложении реализована очень полезная функциональность, статистическая составляющая - рейтинг товара.

Как это работает? Если пользователь добавит товар в корзину (косвенно пользователь проявил интерес к растению) рейтинг товара увеличится.

Данное приложение было разработано студентом Грибовым С.А.

# Предпроектная стадия

## Порядок проведения работ

В ходе создания проекта были проведены следующие виды работ:

### Планирование проекта:

• Определение требований к приложению.

• Выбор используемых технологий.

• Проектирование архитектуры приложения.

### Настройка средств разработки:

Установка и настройка.

• JavaDevelopmentKit (JDK),

• ApacheMaven (для сборки проекта).

• SpringBoot (для создания и запуска приложения).

### Создание проекта:

• Создан новый проект с использованием Spring Initializr.

• Выбраны необходимые зависимости, такие как:

Spring Web, Spring Data JPA, Lombok, Spring Security, Thymeleaf, H2 Database SQL, PostgreSQL Driver.

### Разработка модели данных, репозиториев, сервисов, контроллеров.

### Разработка пользовательского интерфейса:

• Создание HTML-шаблонов с использованием шаблонизатора Thymeleaf , и фреймворка Bootstrap для отображения данных и взаимодействия с пользователем.

### Отладка и оптимизация:

• Отладка кода для исправления ошибок и улучшения производительности.

• Оптимизация кода и конфигурации приложения для улучшения скорости работы и эффективности использования ресурсов.

### Документирование:

• Создание документации по проекту, включая описание архитектуры, инструкции по установке и запуску, описание API (если применимо) и другие справочные материалы.

## Определение требований к приложению

Для разработки приложения были определены следующие требования:

### Функциональные требования:

* Пользователи должны иметь возможность просмотра каталога растений.
* Пользователи должны иметь возможность формировать заказ - добавить (или удалить) выбранный товар в корзину.
* Приложение должно предоставлять возможность создать заказ.
* Администраторы должны иметь возможность детального просмотра всех заказов.
* Администраторы должны иметь возможность закрыть заказ.
* Администраторы должны иметь возможность просмотра списка всех пользователей.
* Пользователям и Администраторам должна быть доступна актуальная информация о количестве товара.

### Нефункциональные требования:

* Удобство использования: Интерфейс должен быть интуитивно понятным и привлекательным для пользователя.
* Безопасность: Данные пользователей должны храниться и передаваться в зашифрованном виде.
* Производительность: Приложение должно обеспечивать быстрый доступ к данным и отзывчивый интерфейс.
* Масштабируемость: Возможность расширения функциональности и увеличения числа пользователей без значительного ухудшения производительности.

### Требования к системе рекомендаций:

* Должен быть реализован механизм, определяющий рейтинг товара на основе предпочтений пользователей.

### Требования к безопасности:

* Аутентификация и авторизация пользователей.
* Защита данных пользователей от несанкционированного доступа.

### Требования к производительности:

* Быстрый доступ к данным и отзывчивый интерфейс приложения.

### Требования к масштабируемости:

* Возможность добавления новых функций и поддержка роста числа пользователей.

### Требования к поддержке и обновлениям:

* Поддержка приложения и обновления для улучшения функциональности и безопасности.

## Выбор стека технологий

### ****Язык программирования****:

Для разработки серверной части приложения используется Java, так как этот язык хорошо подходит для создания масштабируемых и надежных веб-приложений.

### ****Фреймворк для веб-разработки****:

Для быстрой и удобной разработки серверной части приложения используется фреймворк Spring Boot с модулями Spring MVC и Spring Security.

* Spring Security обеспечит безопасность приложения.
* Spring MVC позволит создать удобный интерфейс.

### ****База данных****:

Для хранения данных в рабочем режиме приложения используется СУБД PostgreSQL. Для демонстрационного и отладочного режима используется H2 Database.

### ****Frontend****:

Для создания интерфейса приложения используется язык разметки HTML. Для упрощения разработки интерфейса используется фреймворк Bootstrap.

### ****Шаблонизатор****:

В качестве шаблонизатора используется Thymeleaf.

Thymeleaf интегрируется со Spring и позволяет передавать/получать данные от сервера к frontend.

### ****Аутентификация и авторизация****:

Для обеспечения безопасности приложения, включая аутентификацию пользователей и управление доступом к ресурсам используется SpringSecurity.

### ****Среда разработки:****

Для разработки серверной части приложения используется IntelliJ IDEA 2025.1.3.

## Архитектура приложения

Архитектура приложения в рамках JavaSpring основывается на паттерне проектирования MVC (Model-View-Controller), который позволяет разделить приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Наше приложение состоит из данных составных частей:

### Модель (Model):

* Модель представляет собой структуру данных приложения.
* Включает классы, описывающие сущности приложения ("Гидробионт", "Покупка", "Заказ", "Пользователь", "Роль").

### Представление (View):

* Представление отвечает за отображение данных.
* В веб-приложениях на JavaSpring представление часто реализуется с использованием шаблонизатора Thymeleaf, который позволяет обмениваться данными между контроллером и HTML-шаблоном.

### Контроллер (Controller):

* Контроллер обрабатывает запросы от пользователя и взаимодействует с моделью и представлением.
* В JavaSpring контроллеры представлены классами, аннотированными **@Controller** или **@RestController**.
* Контроллеры содержат методы для обработки различных типов запросов (GET, POST, PUT, DELETE) и вызывают соответствующие сервисы для выполнения бизнес-логики.

### Репозиторий (Repository):

* Репозиторий отвечает за доступ к данным из базы данных.
* В JavaSpring репозитории обычно реализуются с помощью интерфейсов, расширяющих **JpaRepository**, которые предоставляют базовые методы для работы с данными.
* Репозитории используются сервисами для получения, сохранения, обновления и удаления данных.

### Сервис (Service):

* Сервис содержит бизнес-логику приложения.
* Он обычно используется для выполнения операций с данными, координирования действий между различными компонентами приложения и обеспечения единого доступа к функциональности приложения.

### Сервер (Server):

* Сервер представляет собой окружение, в котором работает приложение.
* В JavaSpring приложение обычно запускается на встроенном сервере, таком как Tomcat или Jetty, который обрабатывает HTTP-запросы и управляет жизненным циклом приложения.

Каждый из этих компонентов играет важную роль в построении архитектуры приложения на JavaSpring, обеспечивая его модульность, гибкость и масштабируемость. Они позволяют разделить различные аспекты приложения и обеспечить их независимую разработку и тестирование.

# Проектная стадия

## Проектирование архитектуры приложения

### Проектирование моделей

Начальный этап проектирования приложения - это проектирование основных классов – моделей, сущностей. Определяются основные характеристики каждой модели – поля класса.

Аннотация @Entity это аннотация JPA, используемая для обозначения класса как сущности базы данных. Это сопоставляет класс Java с соответствующей таблицей в базе данных и обеспечивает такие функциональные возможности JPA, как выполнение запросов.

**Класс Hydrobiont** представляет собой объект содержащий информацию о товаре (растение, мох, папоротник, водоросль). Содержит следующие поля:

* id – идентификатор
* direction – направление аквариумистики (морское, пресное)
* type – тип (растение, мох, папоротник, почвопокровка и.т.д)
* name generic – название рода
* name species – название вида
* quantity – количество на складе
* price – цена
* photo – путь до фото (все фото в resource/static)
* level – уровень сложности
* rating – рейтин растения (определяется количеством совершённых покупок, требуется для прогнозирования оптовых закупок)

**Класс Buy** – объект содержит информацию о покупке.

* id – идентификатор
* hydrobiontId – идентификатор товара (гидробионта)
* basketNum – номер корзины (характеристика, объединяющая покупки)

**Класс Order** – содержит информацию о заказе

* id – идентификатор
* basketNum – номер корзины товаров
* userId – идентификатор пользователя
* totalPrice – итоговая стоимость заказа
* timestamp – время создания заказа
* returnedTimestamp – время закрытия заказа

**Класс User** – содержит информацию о пользователе

* id – идентификатор
* name – имя пользователя, хранимое как (имя и фамилия)
* email – адрес эл. почты
* password – пароль хранимый в зашифрованном виде
* nowBasketNum – номер используемой корзины (номер выдаётся пользователю при регистрации и каждый раз после совершения заказа, номер является уникальным).

**Класс Role** – содержит информацию характеризующую тип пользователя

* id – идентификатор
* name – наименование роли (user или admin)

**Класс UserDto** - это объект передачи информации о пользователе между слоями приложения, это “сырой” пользователь, получаемый формой регистрации (фамилия, имя, незашифрованный пароль).

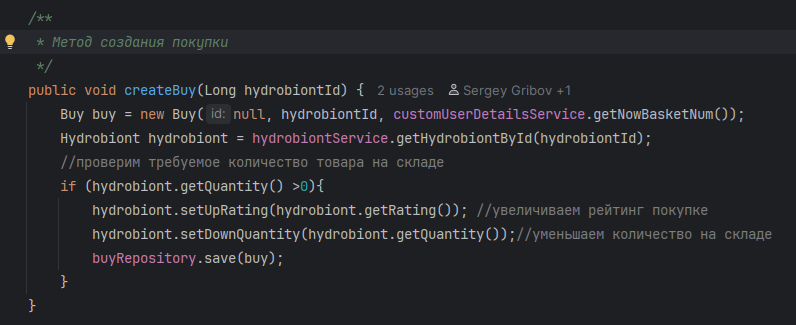
### Проектирование сервисов

Классы сервисы – это классы, которые классов, которые выполняют бизнес-логику, обычно связанную с обработкой данных.

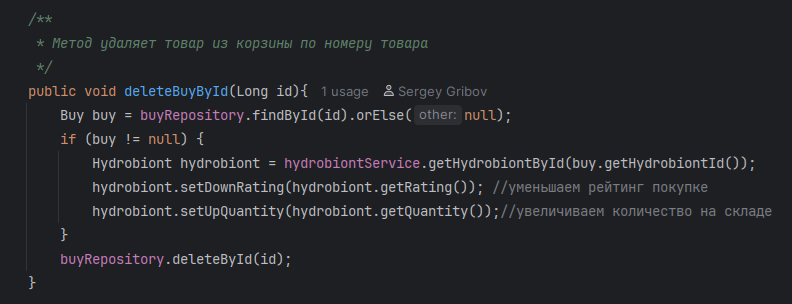
Аннотация @Service объявляет, что класс является компонентом (bin) SpringFramework .

**Класс BuyServise** - вся логика работы приложения с покупками пользователей. Класс включает в себя следующие методы

* Метод создания покупки. Метод добавляет покупку в корзину. Прежде чем создать покупку и сохранить её в репозитории, метод проверяет требуемое количество на складе, увеличивает рейтинг товара, уменьшает количество выбранного товара на складе.



* Метод удаления покупки. Метод удаляет покупку из корзины. Прежде чем удалить покупку, метод проверяет, что требуемая покупка есть в корзине, уменьшает рейтинг товару, увеличивает его количество на складе.



Кроме основных методов, есть служебные

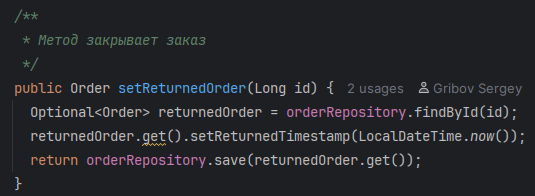
* Метод возвращает список покупок по номеру корзины basketNum
* Метод возвращает стоимость всех покупок по номеру корзины basketNum
* Метод возвращает количество купленных гидробионтов по id гидробионта.

**Класс OrderService** – вся логика работы приложения с заказами пользователей.

* Метод создания заказа для текущего пользователя. Метод создаёт заказ для текущего (авторизованного в данный момент) пользователя. Прежде чем создать заказ, метод проверяет, созданы покупки, сгруппированные по номеру корзины авторизованного пользователя. Существует ли пользователь с данным id.



* Метод закрытия заказа. Метод используя setter объекта Order устанавливает значение текущего времени в поле returnedTimestamp.



Кроме основных методов, есть служебные

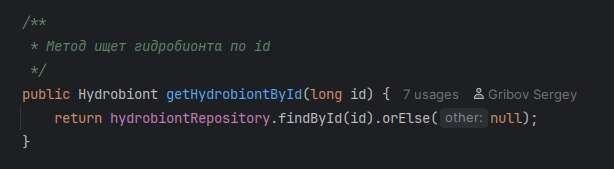
* Метод возвращает список всех заказов
* Метод ищет заказ в репозитории по id.
* Метод ищет заказ в репозитории по id пользователя.
* Метод возвращает список заказов для текущего пользователя.

**Класс HydrobiontService** – вся логика работы с гидробионтами

* Метод возвращает все гидробионты, используется сортировка по полю id.



* Метод возвращает гидробионта по id

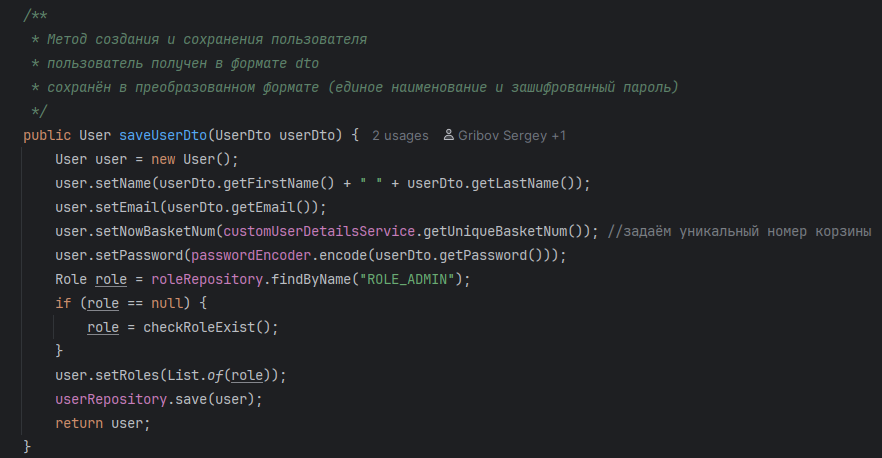


Служебные методы (будут задействованы в будущем, в более широком функционале администратора магазина)

* Метод добавляет гидробионта
* Метод обновляет гидробионта
* Метод удаляет гидробионта

**Класс UserService** – в этом классе находятся методы работы с пользователями и репозиторием пользователей.

* Метод создания и сохранения пользователя. Пользователь получен в формате dto, сохранён в преобразованном формате (единое имя и зашифрованный пароль), задан уникальный номер корзины для покупок. Назначена роль администратора (в рамках опытной тестовой эксплуатации).

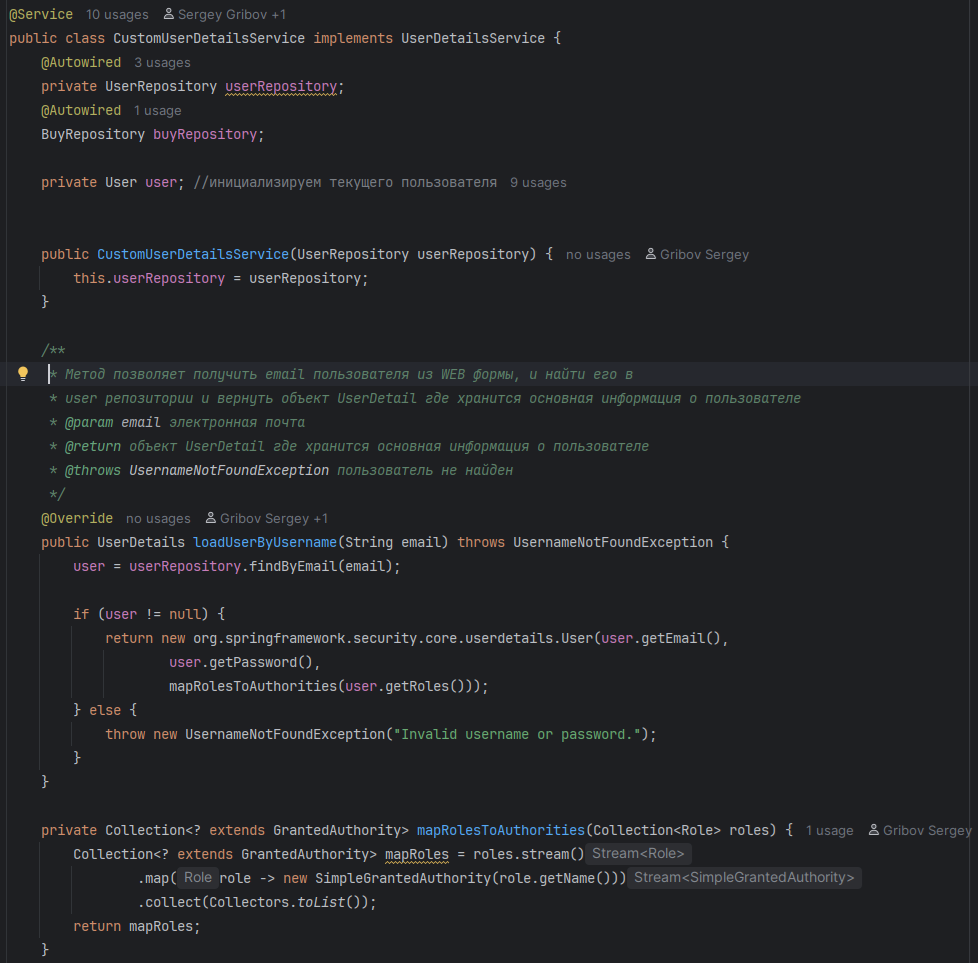


Служебные методы.

* Метод возвращает пользователя по id.
* Метод возвращает пользователя по email.
* Метод возвращает список всех пользователей (в формате dto).
* Метод конвертирует пользователя в формат dto.
* Метод добавляет роль пользователю.
* Метод удаляет пользователя.

**Класс CustomUserDetailsService** – в этом классе логика работы с пользователями, применительно к системе безопасности. Класс имплементирует интерфейс UserDetailsService. Данный класс я переместил в package security.

* В данном классе реализуется метод loadUserByUsername. Метод позволяет получить email пользователя из WEB формы, и найти его в user репозитории и вернуть объект UserDetail где хранится основная информация о пользователе требуемая Spring Security.



В классе есть служебные методы, необходимые другим сервисам

* Метод возвращает номер корзины текущего (авторизованного) пользователя.
* Метод возвращает id текущего пользователя.
* Метод задаёт пользователю новый уникальный номер корзины, требуется при создании нового заказа.
* Метод генерации нового уникального номера корзины.

### Проектирование контроллеров

Классы контроллеры является компонентами веб-приложения, отвечающим за взаимодействие с пользователем и обработку запросов.

Методы контроллера обрабатывают различные HTTP-запросы, например, GET и POST, для отображения страниц, получения и обновления данных.

Аннотация @Controller объявляет, что класс является компонентом (bin) SpringFramework .

Контроллеры данного приложения расположены в package api и package controller.

Контроллеры package api реализуют API (программно аппаратный интерфейс) приложения, и будут востребованы разработчиками собственного frontend и прочих приложений (напр. Приложения для Android).

Контроллеры package controller взаимодействуют с frontend, разработанного в рамках этого проекта, рассмотрим их подробнее.

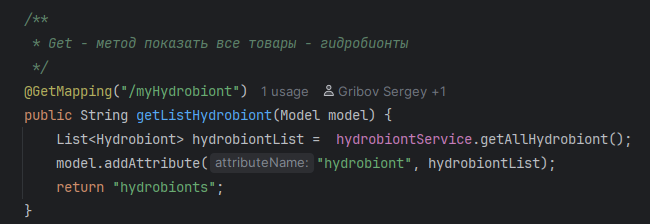
**Класс MyController** - взаимодействует с html формами, расположенными в каталоге resources. На данный момент в классе всего один метод, вызывающий главную страницу.

* Get - метод отображает главную страницу



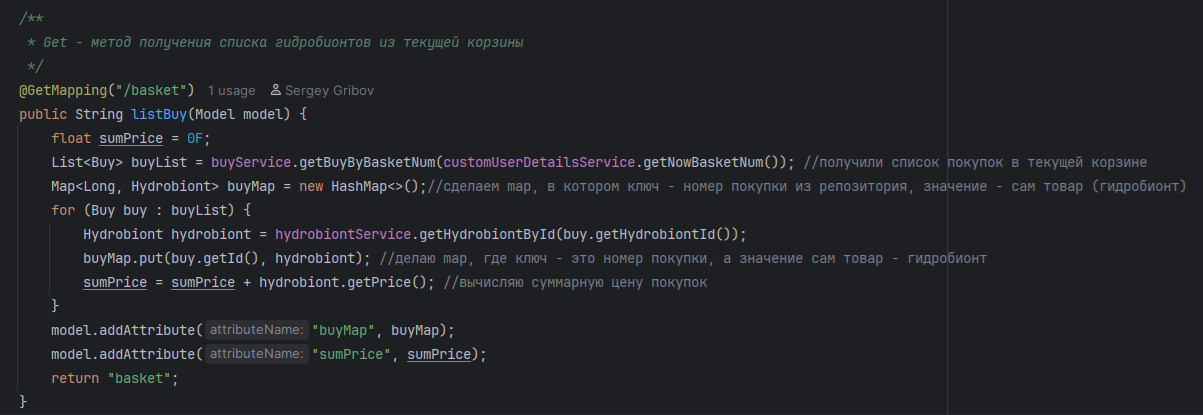
**Класс MyHydrobiontController** – взаимодействует с формами, отображающими информацию о гидробионтах (товарах).

* Get - метод показать все товары – гидробионты. Метод вызывает соответствующий метод в классе HydrobiontService и передаёт в html форму список гидробионтов hydrobiontList, после чего выводит в браузере саму форму hydrobionts.

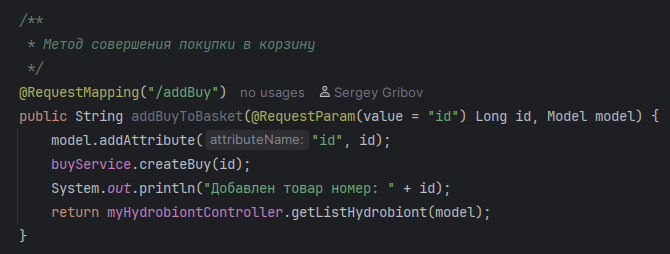


**Класс MyBuyController** – взаимодействует с html формами basket, basketNum. Это формы отображающие покупки пользователей, сгруппированные в корзины.

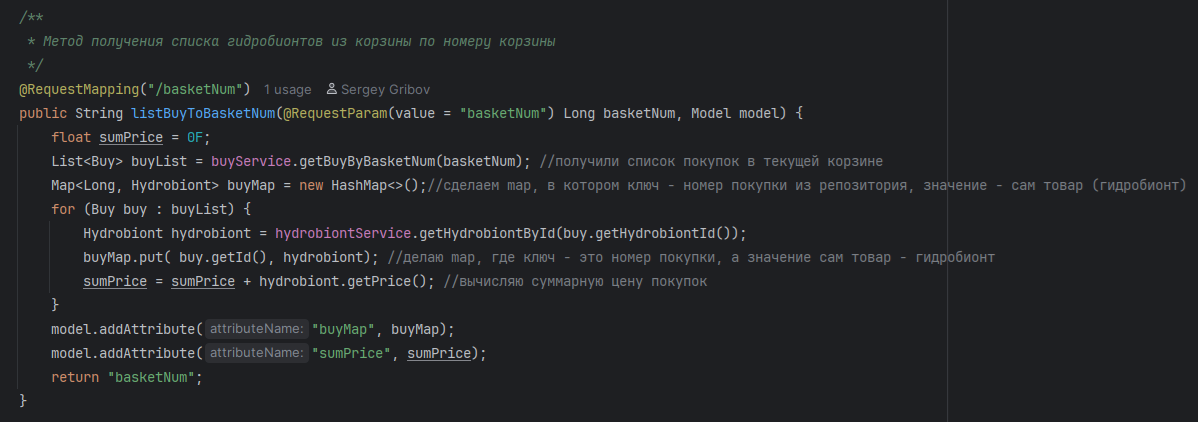
* Get - метод получения списка гидробионтов из текущей корзины. Метод передаёт в форму basket список map покупок, суммарную стоимость sumPrice, выводит в браузере саму форму basket.



* Метод совершения покупки (добавляет покупку в корзину). Вызывает соответствующий метод и выводит требуемую форму (в данном случае форму hydrobionts.



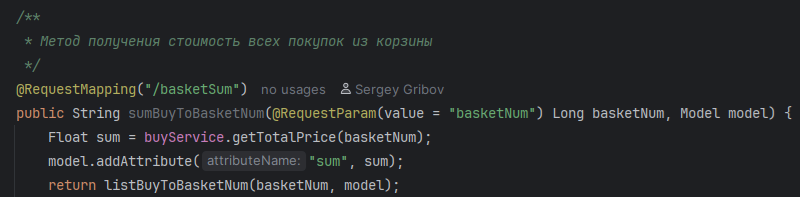
* Метод получения списка гидробионтов из корзины по номеру корзины



* Метод удаления покупки из корзины. Выводит форму basket, вызывая рассмотренный выше метод.

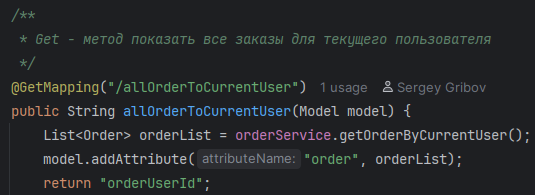


* Метод получения стоимости всех покупок из корзины. Выводит форму basketNum, вызывая рассмотренный выше метод.

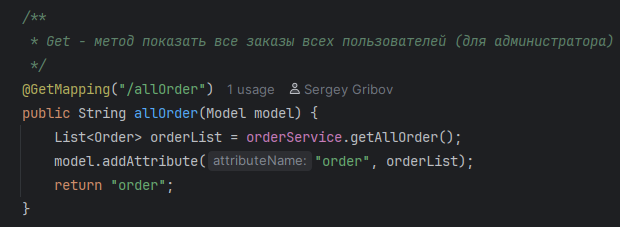


**Класс MyOrderController** – взаимодействует с html формами order, orderUserId. Это формы отображающие заказы пользователей.

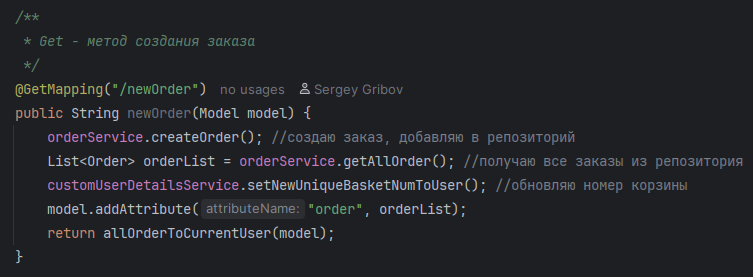
* Get - метод показать все заказы для текущего (авторизованного) пользователя. Метод вызывает соответствующий метод у класса orderService, получает список orderList. Выводит в браузер форму orderUserId, передав ей список orderList.



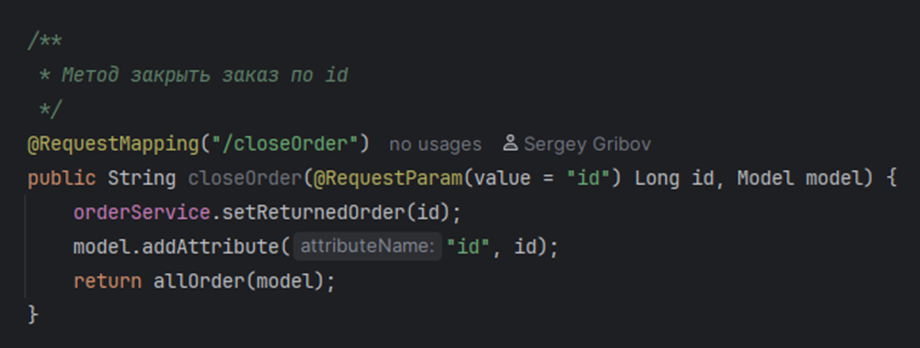
* Get - метод показать все заказы всех пользователей. Работает аналогично методу рассмотренному выше.



* Get - метод создания заказа. Метод вызывает соответствующий метод создания заказа в классе orderService, обновляет номер корзины польхователя (выдаём пользователю новую корзину для покупок). Выводит в браузер форму orderUserId, через метод рассмотренный выше.

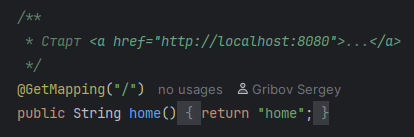


* Метод - закрыть заказ по id. Выводит в браузер форму order через метод рассмотренный выше.

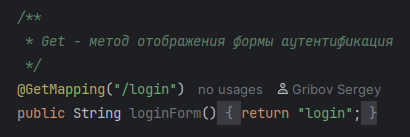


**Класс MyUserController** – взаимодействует с html формами login, register, users.

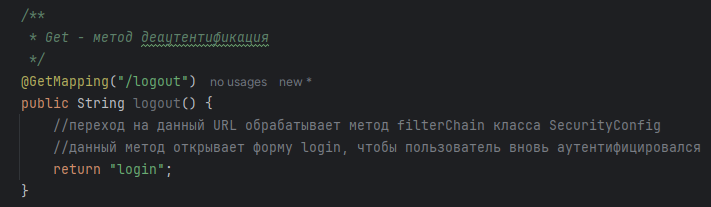
* Метод отображения главной формы



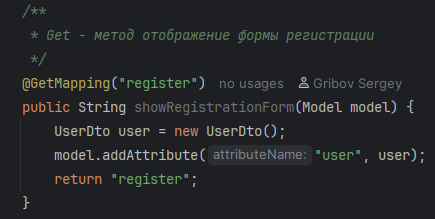
* Get - метод отображения формы аутентификация



• Get - метод деаутентификации



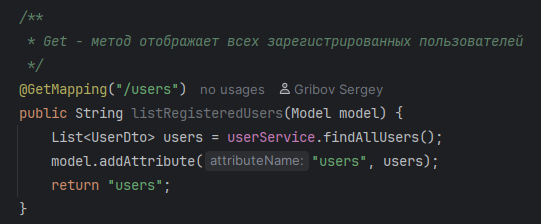
* Get - метод отображение формы регистрации



* Post - метод регистрации пользователя. Метод проверяет наличие пользователя в репозитории по email, если пользователь отсутствует, происходит регистрация и пользователь будет добавлен в репозиторий, через соответствующий метод в классе userService.

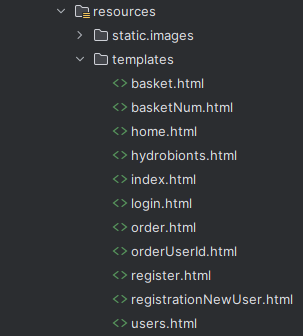


* Get - метод отображает всех зарегистрированных пользователей



## Проектирование WEB – интерфейса приложения

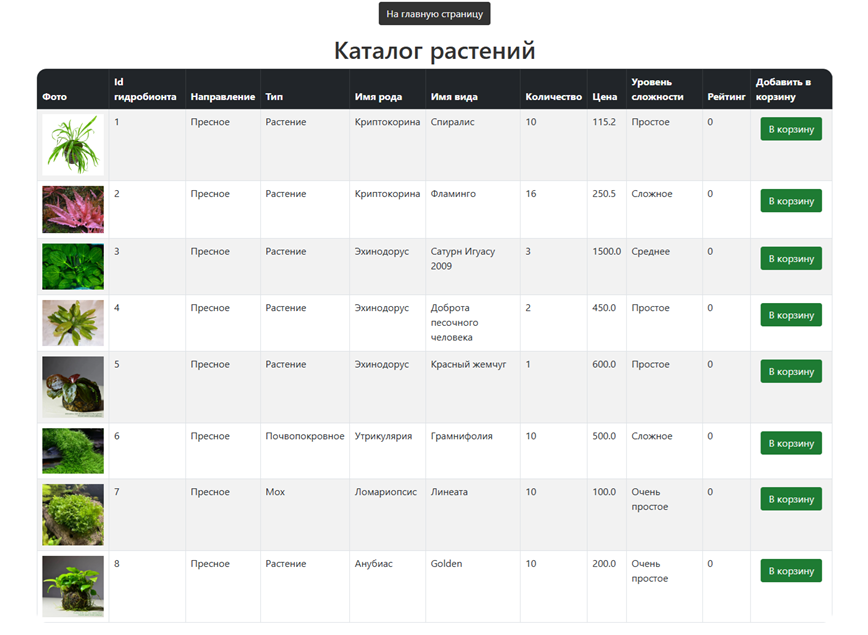
В ходе проектирования WEB – интерфейса приложения были созданы html формы.



* home – главная форма, отображает полный функционал приложения



* hydrobionts – форма, отображает каталог гидробионтов

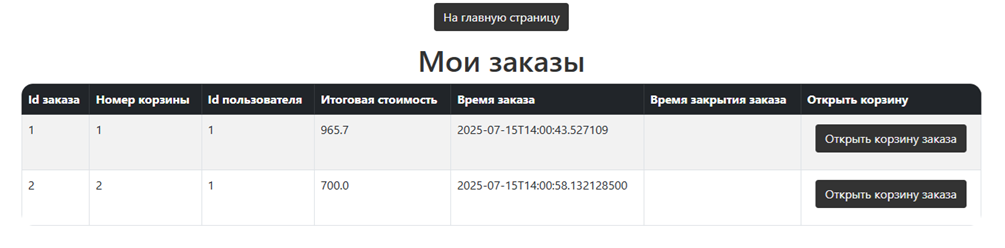


* basket – форма отображения покупок пользователей, сгруппированных в корзину.

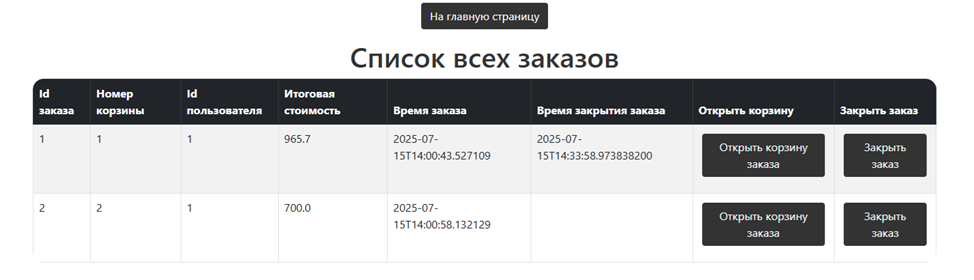


basketNum – форма позволяющая открыть корзину любого заказа, по сути эот форма basket, но без функциональной кнопки “Удалить”

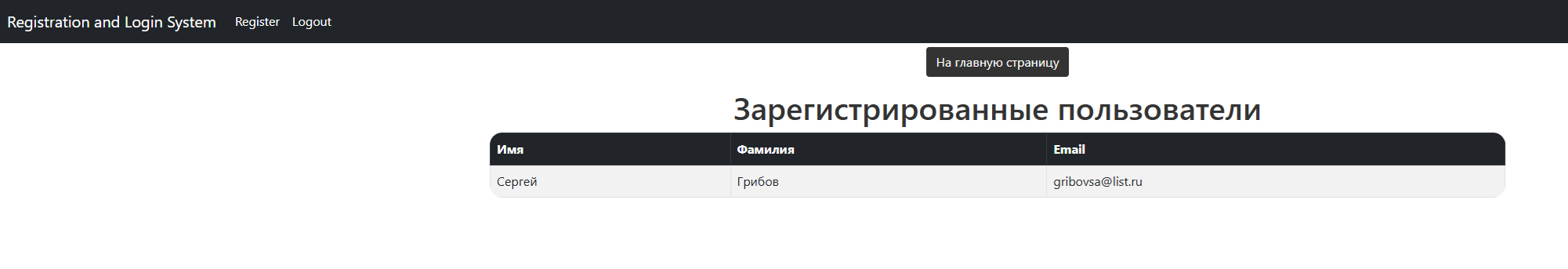
* orderUserId – форма, отображает заказы определённого пользователя



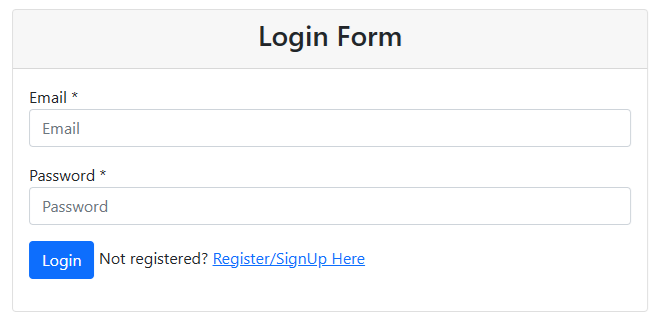
• order – форма, отображает перечень всех заказов



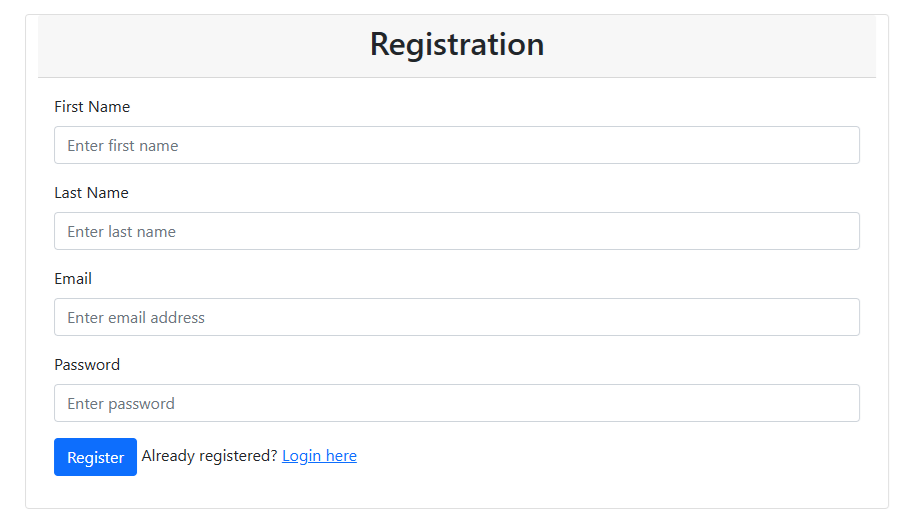
* users – форма отображает список всех зарегистрированных пользователей



* login – форма аутентификации



* register – форма регистрации пользователя

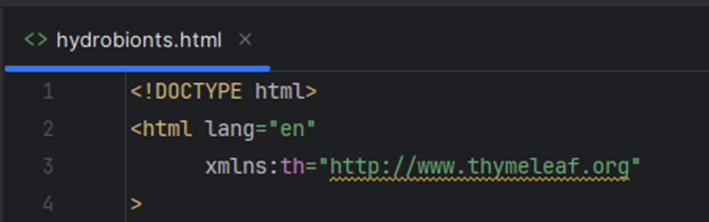


Рассмотрим на примере формы hidrobionts.html структуру .

Данный HTML-код представляет собой шаблон для веб-страницы, использующей Thymeleaf в качестве шаблонизатора.

Thymeleaf позволяет создавать динамические веб-страницы, вставляя данные из модели Java в HTML-шаблон.

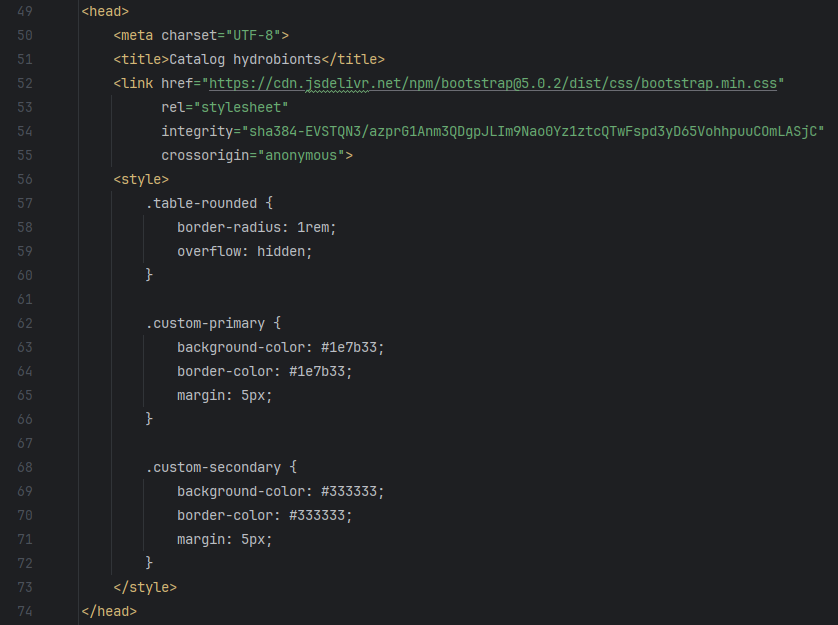
Давайте разберем основные элементы данного шаблона:



<!DOCTYPEhtml>: Определяет тип документа и его версию (HTML5).

<htmlxmlns:th="http://www.thymeleaf.org">: Указывает на использование Thymeleaf в качестве шаблонизатора и определяет пространство имен для Thymeleaf.

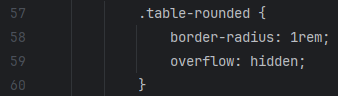
<head>: Содержит метаданные страницы, такие как заголовок (<title>).

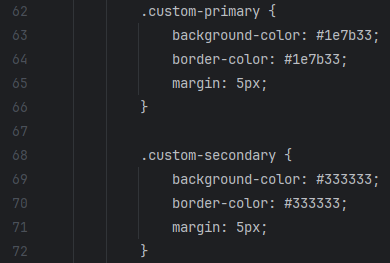


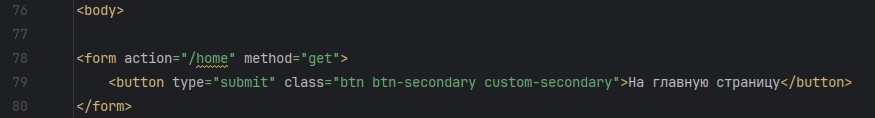
<link>: Ссылается и подключает framework Bootstrap.

Bootstrap предоставляет набор стилей и механизмов для оформления таблиц, элементов, кнопок полей итд.

<style>: Секция стилей для определения внешнего вида элементов.

 полукруглые углы у таблицы

 стили кнопок в таблице (зеленая и черная)



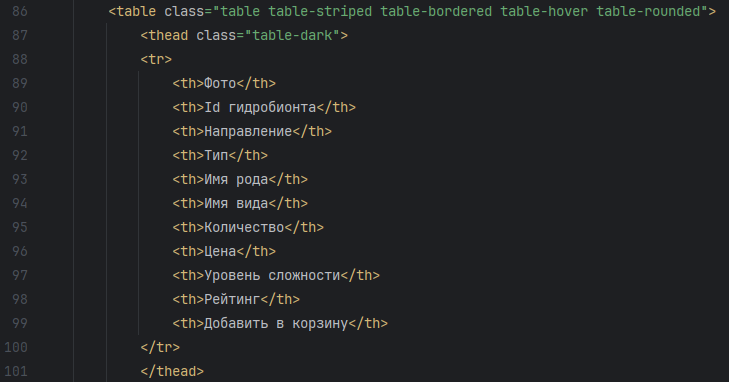
<body>: Основная часть страницы, содержащая контент, видимый пользователю.

Формы (<form>): используются для создания кнопок, которые выполняют определенные действия при нажатии. Атрибут action указывает на адрес, куда будет отправлен запрос при отправке формы, а method определяет HTTP-метод (GET или POST).

Кнопки (<button>): Содержатся внутри форм и используются для выполнения определенных действий на сервере при их нажатии.



Текстовый контент (<h1>, <h2>, <p>): Отображают заголовки и текстовую информацию на странице.



<table>: Таблица для отображения данных. <tr> - это строки, <th> - это значение в столбце. В данном случае это шапка <thead> нашей таблицы гидробионтов.

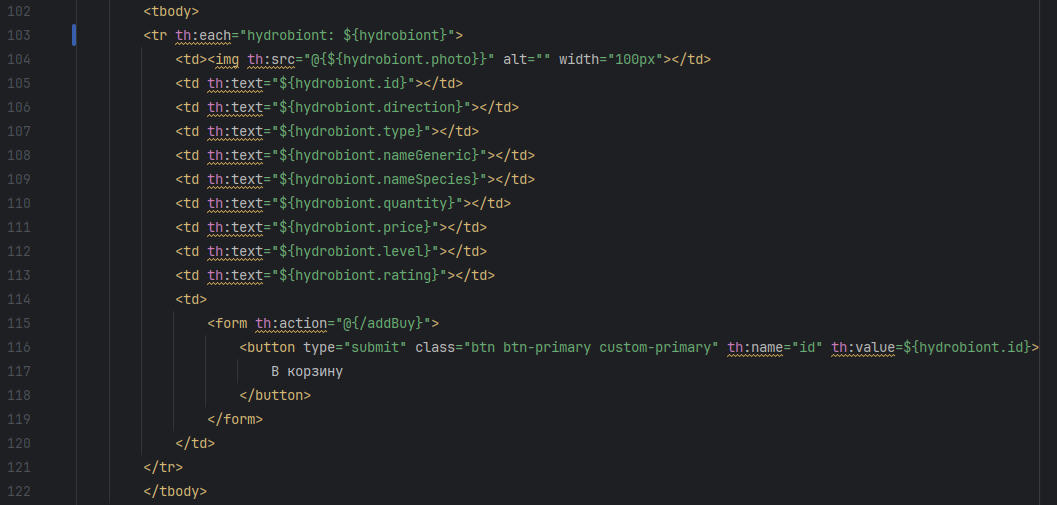


Строки 86,87 – это внешний вид таблицы, определяемый фреймворком Bootstrap.

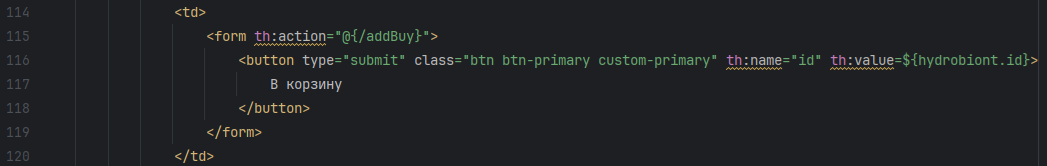
Далее идёт тело нашей таблицы - <tbody>, причём тело генерируется динамически (выполняется цикл for each):



Метод контроллера передает в форму атрибут ${hydrobiont}, это список объектов - hidrobiont. Цикл создаёт новые строки в таблице и раскладывает значения полей объекта hidrobiont по ячейкам в виде текстовых полей.



Исключение – последняя ячейка в строке, в этой ячейке расположена кнопка



th:action="@{/addBuy}" – вызов метода addBuy в классе BuyController при нажатии на кнопку

type="submit" – тип кнопки

class="btn btn-primary custom-primary" – внешний вид кнопки, см выше, в области <style> определены два стиля кнопок (зеленая и черная)

th:name="id" – название атрибута, через который значение будем передавать в метод

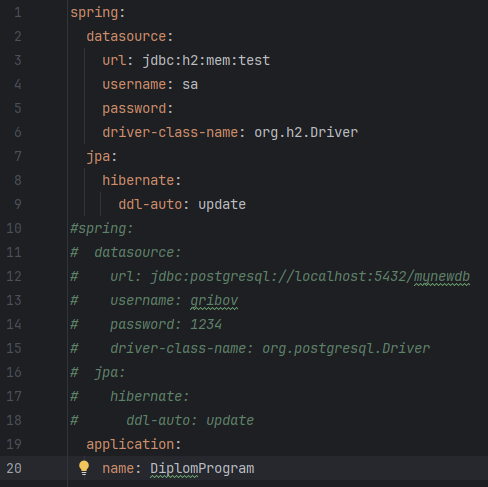
th:value=${hydrobiont.id} – значение которое будем передавать принажатии на кнопку, в нашем случае id гидробионта, чтобы совершить покупку.

## Проектирование базы данных

В режиме опытной эксплуатации, чтобы протестировать работу приложения, в качестве СУБД этого проекта используется H2 database.

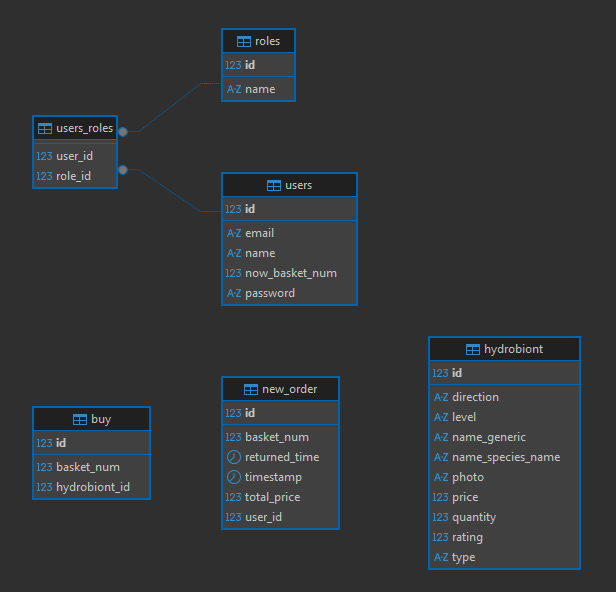
В рабочем режиме будет использоваться PostgreSQL.

В файле application.ymlуказаны следующие настройки.

****

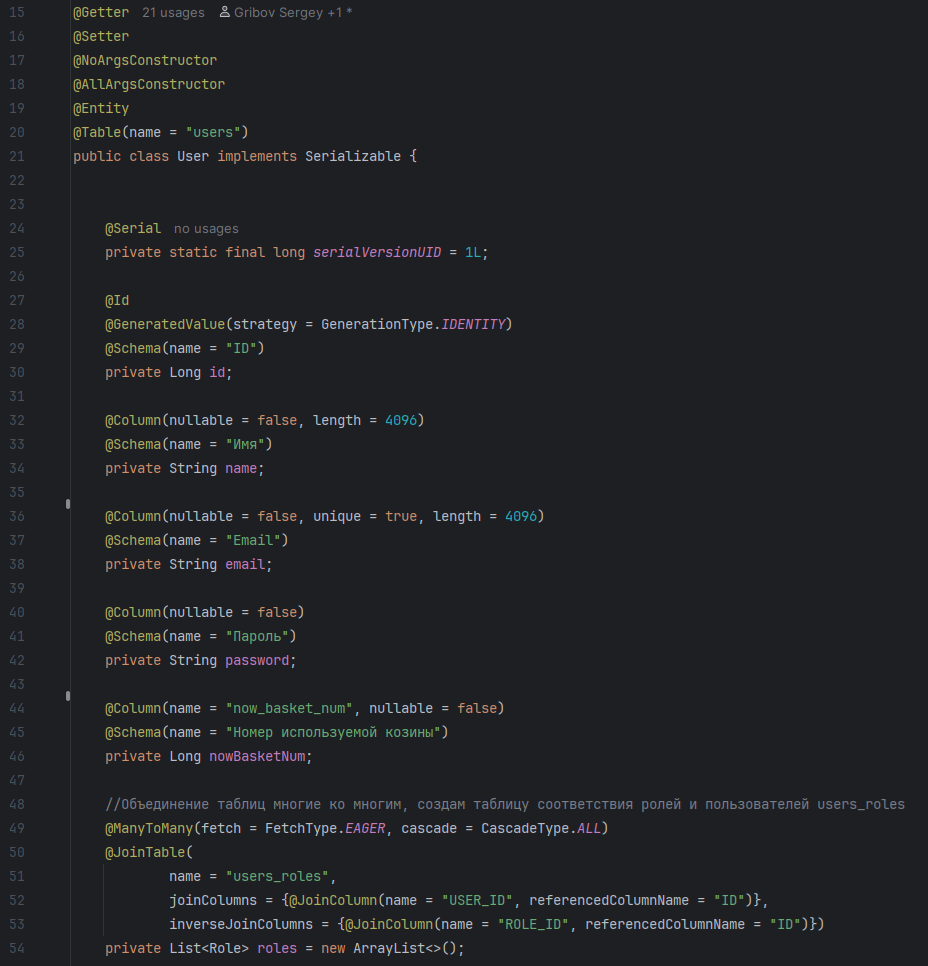
* datasource: содержит настройки для подключения к базе данных.
* driver-class-name указывает класс драйвера.
* url содержит URL-адрес для подключения к базе данных.
* username и password содержат учетные данные пользователя базы данных.
* jpa: содержит настройки JPA и Hibernate.
* hibernate.ddl-auto определяет стратегию автоматического создания таблиц в базе данных. Значение update указывает Hibernate на автоматическое обновление схемы базы данных при изменении сущностей JPA.

Структура базы данных имеет вид:



Jakarta Persistence API (JPA) — это технология, которая упрощает взаимодействие между приложениями Java и реляционными базами данных. В основе JPA лежат аннотации @Entity и @Table, которые позволяют разработчикам сопоставлять объекты Java с таблицами базы данных.

Рассмотрим класс User. Этот класс представляет собой сущность (Entity) JPA для хранения информации о пользователях (User) в базе данных.



Разберем каждую аннотацию и поле:

* @Entity: обозначает, что класс является сущностью JPA.
* @Id: обозначает поле, которое является первичным ключом в базе данных.
* @GeneratedValue: указывает, как будет генерироваться значение для первичного ключа. В данном случае GenerationType.IDENTITY используется для автоматической генерации значения идентификатора базой данных.
* @Column - определяет атрибуты столбца в базе данных, такие как: имя столбца (name), обязательность (nullable) и длину (length).
* private Long id - поле для хранения идентификатора пользователя.
* private String name - поле для хранения имени пользователя.
* private String email - поле для хранения адреса электронной почты пользователя. Уникальность адреса почты обеспечивается атрибутом unique = true.
* private String password - поле для хранения пароля пользователя.
* @ManyToMany - определяет отношение многие ко многим между пользователями и ролями (Role). Каждый пользователь может иметь несколько ролей.
* @JoinTable - определяет таблицу, которая используется для хранения связей между пользователями и ролями.

## Безопасность приложения

Для обеспечения безопасности приложения был использован Spring Security.

Spring Security - это фреймворк аутентификации и авторизации для приложений на платформе Java. Он обеспечивает набор функций для защиты приложений, включая аутентификацию пользователей, управление их правами доступа, защиту от атак, таких как CSRF (межсайтовая подделка запроса) и сессионные атаки, а также интеграцию с другими инфраструктурными службами, такими как LDAP, OAuth и другими.

Spring Security позволяет настраивать правила безопасности с использованием аннотаций или конфигурационных классов Java. Он также интегрируется с другими модулями Spring, такими как Spring MVC, что облегчает создание безопасных и надежных приложений на платформе Java.   
Основные концепции и функции Spring Security:

1. **Аутентификация**. Это процесс проверки подлинности пользователя. Spring Security поддерживает различные механизмы аутентификации, такие как форма, HTTP Basic, OAuth и другие.
2. **Авторизация**. Это процесс определения разрешений пользователя на доступ к определенным ресурсам или выполнение определенных действий. Spring Security использует роли и разрешения для управления доступом.
3. **Фильтры безопасности**. Spring Security использует цепочку фильтров для обработки запросов и применения правил безопасности. Эти фильтры обеспечивают аутентификацию, авторизацию, защиту от CSRF-атак, управление сессиями и другие функции.
4. **Защита от CSRF-атак**. CSRF (межсайтовая подделка запросов) - это атака, при которой злоумышленник заставляет пользователя совершить нежелательное действие на веб-сайте, на который он авторизован. Spring Security предоставляет защиту от таких атак.
5. **Управление сессиями**: Spring Security предоставляет возможности управления сессиями, включая защиту от смены сессии, контроль одновременных сессий и настройку таймаута сессии.
6. **Шифрование паролей**: Spring Security советует хранить пароли в зашифрованном виде, предоставляя возможность использовать алгоритмы шифрования, такие как BCrypt, SCrypt и PBKDF2.
7. **Интеграция с Spring Framework**: Spring Security интегрируется плавно с Spring Framework, что упрощает обеспечение безопасности в приложениях на основе Spring.
8. **Настройка и расширяемость**: Spring Security предоставляет множество возможностей для настройки и расширения. Можно настраивать логику аутентификации и авторизации, создавать собственные фильтры безопасности и интегрироваться с сторонними сервисами аутентификации.

**Класс SecurityConfig.**

Рассмотрим класс, в котором сосредоточена основная логика системы безопасности нашего приложения.



* **@Configuration** и **@EnableWebSecurity**: Эти аннотации указывают, что этот класс является конфигурационным классом Spring и активирует поддержку веб-безопасности.
* **SecurityFilterChain**: Этот метод создает фильтр цепочки безопасности, который определяет правила доступа к различным URL-адресам. В данном случае, он позволяет доступ к **/register/\*\*** и **/index** всем пользователям, а к **/users** и всем остальным URL-ам только пользователям с ролью "ADMIN".
* **formLogin**: Этот метод настраивает аутентификацию через форму. Он указывает, что страница логина находится по адресу **/login**, обработка логина выполняется по адресу **/login**, а после успешной аутентификации пользователь перенаправляется на страницу **/home**.
* **logout**: Этот метод настраивает логаут, позволяя пользователям разлогиниваться по адресу **/logout**.
* **PasswordEncoder**: Этот бин создает экземпляр BCryptPasswordEncoder, который используется для хэширования паролей пользователей.
* **configureGlobal**: Этот метод конфигурирует аутентификацию, указывая, что UserDetailsService используется для загрузки пользовательских данных и BCryptPasswordEncoder для проверки паролей.

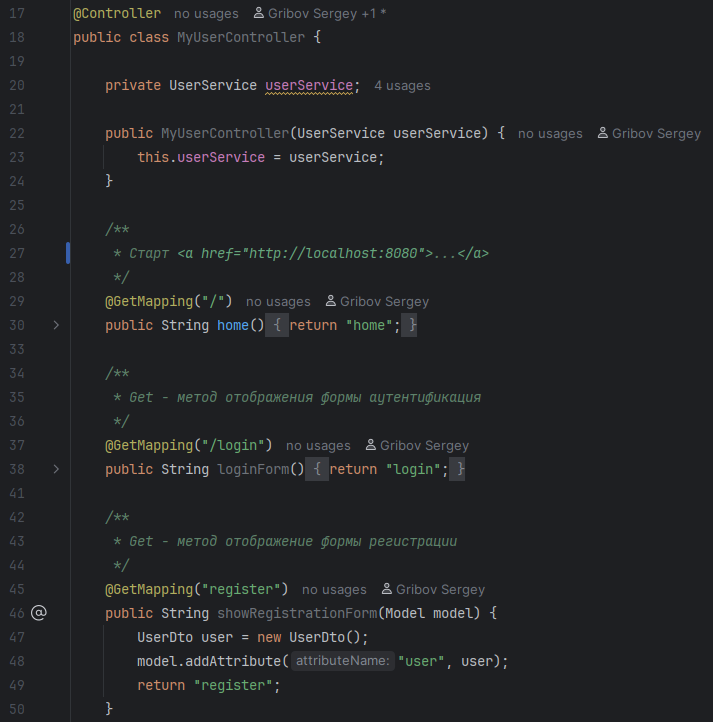
Этот код предоставляет базовую конфигурацию безопасности для нашего приложения Spring,

**Класс MyUserController.**

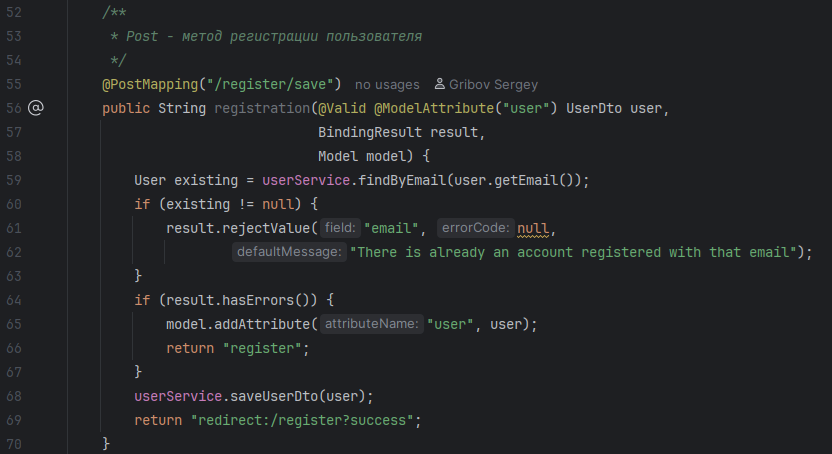
Также в ходе реализации безопасности был использован метод MyUserController, рассмотрим его подробнее с этой точки зрения.

Этот класс представляет собой контроллер Spring MVC для обработки запросов, связанных с пользователями.

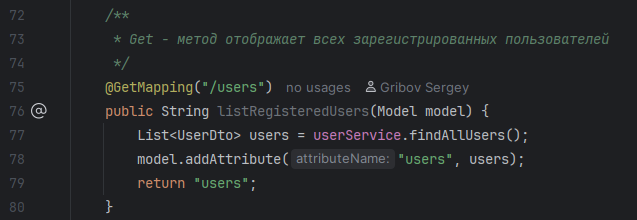
Этот контроллер обеспечивает основную функциональность для регистрации пользователей, отображения списка зарегистрированных пользователей и управления представлением для логина. Он использует объекты UserDto для передачи данных о пользователях между контроллером и представлением.



* **@Controller**: Аннотация, которая указывает, что этот класс является контроллером Spring MVC.
* **@Autowired UserService userService**: Инъекция зависимости UserService для доступа к функциональности управления пользователями.
* **@GetMapping** и **@PostMapping**: Аннотации, определяющие, что методы обрабатывают GET и POST запросы соответственно.
* **home()**: Метод, обрабатывающий запросы к корневому URL. Возвращает имя представления "home".
* **loginForm()**: Метод, обрабатывающий GET запросы к URL "/login" и возвращающий имя представления "login".
* **showRegistrationForm(Model model)**: Метод, обрабатывающий GET запросы к URL "/register" и возвращающий имя представления "register", при этом добавляется атрибут "user" в модель для использования в представлении.



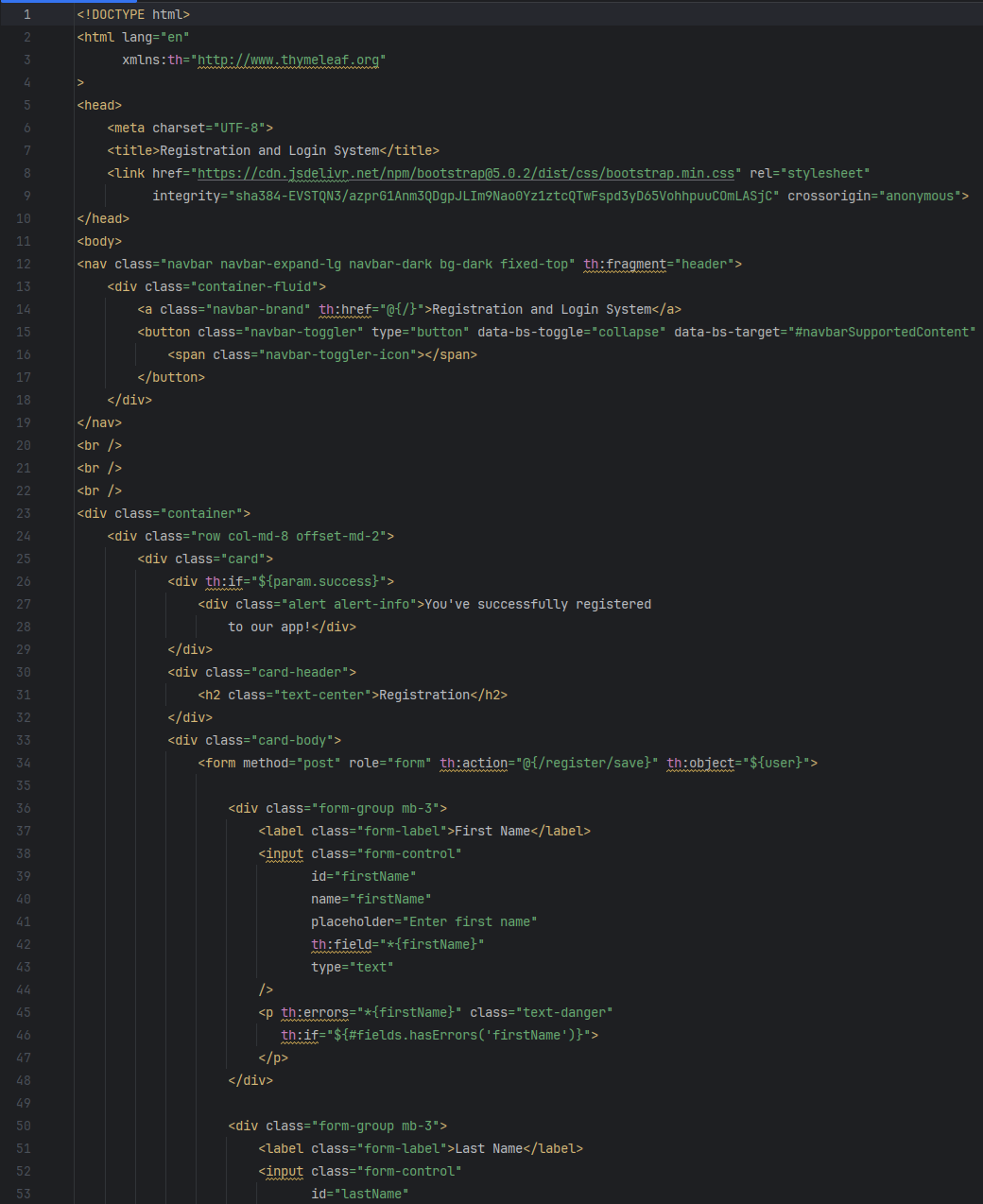
* **registration(@Valid @ModelAttribute("user") UserDto user, BindingResult result, Model model)**: Метод, обрабатывающий POST запросы к URL "/register/save" для регистрации новых пользователей. Аннотация @Valid используется для валидации объекта UserDto. Если есть ошибки валидации, метод возвращает имя представления "register" с атрибутами ошибок. В противном случае пользователь сохраняется через UserService, и пользователь перенаправляется на страницу "/register?success".



* **listRegisteredUsers(Model model)**: Метод, обрабатывающий GET запросы к URL "/users" для отображения списка зарегистрированных пользователей. Получает список пользователей через UserService и добавляет его в модель для использования в представлении "users".

Далее рассмотрим несколько HTML форм, исполняющие механизмы регистрации и аутентификации пользователей

**HTML форма register.html** - функции регистрации при открытии приложения.



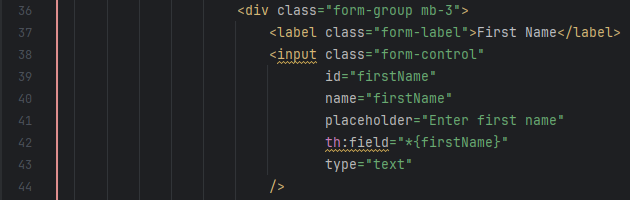


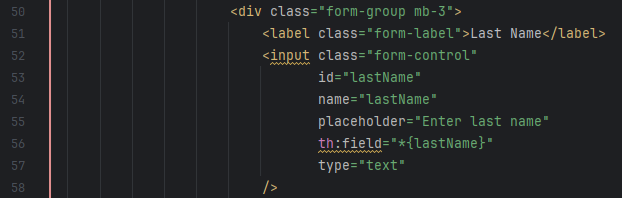
Вот краткое описание основных компонентов:

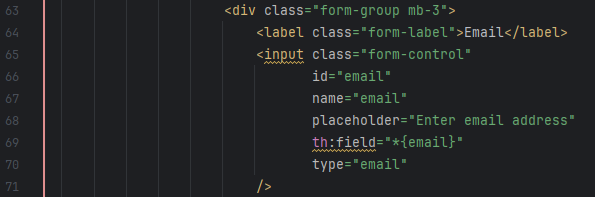
1. Подключает таблицу стилей Bootstrap из CDN.

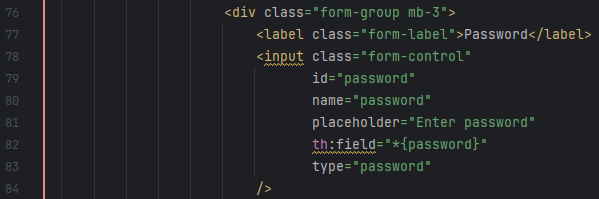


1. Отображают формы ввода регистрационных данных – имя, фамилия, email, пароль









1. Отображают сообщения об ошибках в случае неверного ввода

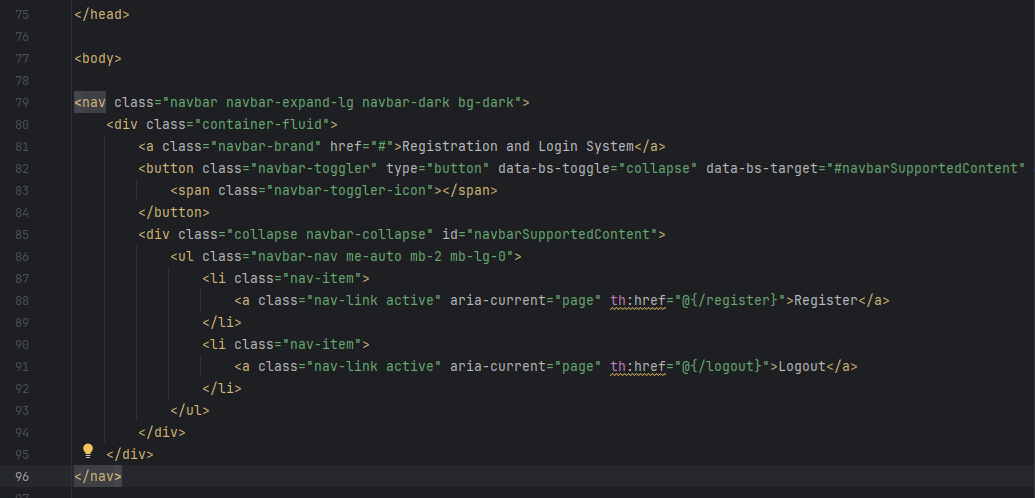








**HTML форма users.html** – форма предназначена для просмотра всех зарегистрированных пользователей, но в навигационном меню есть возможность регистрации нового пользователя и выхода (деаутентификация).



Разберем основные элементы:

1. **<title>Registration and Login System</title>**: Заголовок страницы.
2. Ссылка на стили Bootstrap для оформления страницы.
3. **<body>**: Основное содержимое страницы.
4. **<nav>**: Навигационное меню.
5. **<div class="container">**: Контейнер для размещения содержимого страницы.

**HTML форма registrationNewUser.html –** форма аутентификации



1. Создает поле ввода для логина с указанием класса стиля и обязательностью заполнения.



1. Аналогично создают поле ввода для пароля.



1. Создает скрытое поле для передачи токена CSRF (Cross-Site Request Forgery) для защиты от подделки межсайтовых запросов.



1. Создает кнопку для отправки формы с текстом "Зарегистрироваться" и стилями Bootstrap.



## Docker-контейнер

В случае запуска приложения в рабочей режим, требуется установить ПО Docker и создать контейнер с PostgreSQL.

1. Установить Docker
2. Ввести команду:

**docker run --name mypostgres -e POSTGRES\_DB=mynewdb -e POSTGRES\_USER=gribov -e POSTGRES\_PASSWORD=1234 -p 5432:5432 -d postgres**

Команда **docker run** используется для создания и запуска контейнера на основе Docker-образа. Давайте разберем каждый параметр данной команды:

* **--name mypostgres** - устанавливает имя контейнера **mypostgres**.
* -**e POSTGRES\_DB= mynewdb** - устанавливает переменную окружения **POSTGRES\_DB** со значением **mynewdb** для создания базы данных с именем **mynewdb**.
* **-e POSTGRES\_USER=gribov** - устанавливает login
* **-e POSTGRES\_PASSWORD=1234** - устанавливает пароль
* **-d** - запускает контейнер в фоновом режиме (detached mode).
* **-p 5432:5432** - пробрасывает порт 5432 из контейнера на хостовую машину. Первое число (5432) - порт на хостовой машине, второе число (5432) - порт контейнера PostgreSQL.
* **postgres**: Имя Docker-образа, на основе которого будет создан контейнер. В данном случае, это официальный образ PostgreSQL из Docker Hub.

# ****Заключение****

В рамках данного дипломного проекта было разработано веб-приложение – интернет-магазина с системой рекомендаций на основе предпочтений пользователей «Магазин аквариумных растений».

В процессе разработки были достигнуты следующие цели:

* Создание удобного и интуитивно понятного интерфейса для пользователей приложения.
* Реализован функционал: отображения информации о товарах, перемещения и удаления товаров из корзины покупок, создание заказов, администрирование заказов.
* Обеспечение безопасности данных пользователей с использованием механизмов аутентификации и авторизации Spring Security.
* Использование принципов SOLID и паттернов проектирования для обеспечения гибкости и расширяемости кода.
* Дальнейшее развитие приложения может включать в себя добавление нового функционала, такого как: предоставление справочной информации о каждом товаре, функционал отмены, изменения заказов, внедрение платёжных сервисов, и конечно же отладка, поиск ошибок.

# Список литературы

1. Учебный материал Geekbrains.
2. <https://www.baeldung.com/>
3. <https://www.thymeleaf.org/>