**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«**

**»**

**Курсовая работа**

ДИСЦИПЛИНА: «»

Тема: Фитнес-центр. Управление членством и расписанием групповых занятий.

Выполнил: Студент группы

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2024

Санкт-Петербург

2024 г.

**Содержание:**

Оглавление

[1. Постановка задачи 3](#_Toc185282066)

[Реализация 3](#_Toc185282067)

[2. Описание API 3](#_Toc185282068)

[3. Документация кода серверной части 6](#_Toc185282069)

[Раздел model 6](#_Toc185282070)

[Раздел controller 7](#_Toc185282071)

[Раздел repository 8](#_Toc185282072)

[Остальные файлы 9](#_Toc185282073)

[4. Документация кода клиентской части 10](#_Toc185282074)

[5. Приложение серверной части 12](#_Toc185282075)

[Раздел aspect 12](#_Toc185282076)

[Раздел config 13](#_Toc185282077)

[Раздел controller 13](#_Toc185282078)

[Раздел model 14](#_Toc185282079)

[Раздел repository 16](#_Toc185282080)

[Остальные файлы 17](#_Toc185282081)

[6. Приложение клиентской части: 19](#_Toc185282082)

# Постановка задачи

Для выполнения курсовой работы необходимо создать сайт фитнес зала c серверным приложением управления членством и расписанием групповых занятий.

* *Клиентское приложение:* HTML + CSS + JS
* *Серверное приложение:* Java
* *REST API :* Spring Boot.
* *База данных:* MySQL. Взаимодействие через Hibernate
* *Логирование:* Библиотека Logback

## Реализация

Для реализации поставленной задачи, было необходимо реализовать следующее наполнение проекта:

* Регистрация людей в зал.
* Создание заявок/заказов на покупку спортивных добавок.
* Управление статусами пропусков клуба (пропуск действителен / не действителен).
* Отслеживание даты окончания срока действия пропуска.
* Расчет стоимости пропуска (учёт ожидаемого времени членства в зале и количества посещений в год).
* Создание и размещение групповых занятий

Также необходимо реализовать следующие аспекты серверной части проекта:

* Необходимо обеспечить сериализацию/десериализацию состояний сервера
* Сервер должен логировать все события, которые происходят на его стороне.
* Данные приложения на сервере необходимо хранить в БД
* Связь между клиентом и сервером обеспечить с помощью REST API Spring-boot.
* Сборка серверной части с помощью Maven

# Описание API

**Метод авторизации** - Используется для аутентификации пользователей путем передачи их логина и пароля.

URL: /api/auth/login

Метод: POST

**Параметры запроса:**

* login (String) — логин пользователя.
* password (String) — пароль пользователя.

Пример запроса:

{

"login": "user123",

"password": "securePassword"

}

**Ответ от сервера:**

* 200 OK - Успешная аутентификация.

Тело ответа:

{

"id": 1,

"name": "Иван Иванов",

"login": "user123",

"createdAt": "2023-10-01T12:34:56"

}

* 401 Unauthorized - Неверный логин или пароль.

Тело ответа:

{

"message": "Неверный логин или пароль."

}

**Метод создания абонемента** - Создает новый абонемент для пользователя.

URL: /api/subscriptions

Метод: POST

**Параметры запроса:**

* private String firstName; - имя клиента
* private String lastName; - фамилия клиента
* private String login; - логин клиента
* private String phone; - телефон клиента
* private Integer months; - количество месяцев абонемента
* private Integer trainerSessions; - занятия с тренером в комлпекте
* private Boolean unlimited; - безлимитное посещение
* private Boolean fullDay; - время посещения
* private Boolean freezeAvailable; - заморозка абонемента
* private Boolean poolAccess; - зал + бассейн
* private LocalDate startDate; - дата начала абонемента
* private String status; - статус абонемента
* private LocalDate endDate; -конечная дата абонемента

Пример запроса:

{

"firstName": "Иван",

"lastName": "Иванов",

"login": "ivanov",

"phone": "+1234567890",

"months": 6,

"trainerSessions": 10,

"unlimited": true,

"fullDay": false,

"freezeAvailable": true,

"poolAccess": true,

"startDate": "2024-01-01",

"status": "active",

"endDate": "2024-07-01"}

**Ответ от сервера:**

* 201 Created - Абонемент успешно создан.

Тело ответа:

{

"id": 1,

"firstName": "Иван",

"lastName": "Иванов",

"login": "ivanov",

"phone": "+1234567890",

"months": 6,

"trainerSessions": 10,

"unlimited": true,

"fullDay": false,

"freezeAvailable": true,

"poolAccess": true,

"startDate": "2024-01-01",

"status": "active",

"endDate": "2024-07-01"

}

* 400 Bad Request - Ошибка в данных запроса.

Тело ответа:

{

"message": "Некорректные данные абонемента."

}

3. **Метод создания занятия группы** - Создает новое групповое занятие

URL: /api/classes

Метод: POST

Параметры запроса:

* className (String) — название занятия.
* classDate (String, формат YYYY-MM-DD) — дата проведения занятия.

Пример запроса:

{

"className": "бокс",

"classDate": "2024-01-15",

}

Ответ от сервера:

* 201 Created - Занятие успешно создано.

Тело ответа:

{

"id": 55,

"className": "бокс",

"classDate": "2024-01-15",

"createdAt": "2023-10-01T14:20:30"

}

* 400 Bad Request- Ошибка в данных запроса.

Тело ответа:

{

"message": "Некорректные данные занятия."

}

4. **Метод регистрации пользователя** - Регистрирует нового пользователя в системе.

URL: /api/users

Метод: POST

Параметры запроса:

* name (String) — имя пользователя.
* login (String) — уникальный логин.
* password (String) — пароль.

Пример запроса:

{

"name": "Мария Петрова",

"login": "maria.p",

"password": "strongPassword123"

}

Ответ от сервера:

* 201 Created - Пользователь успешно зарегистрирован.

Тело ответа:

{

"id": 10,

"name": "Мария Петрова",

"login": "maria.p",

"createdAt": "2023-10-01T15:00:00"

}

* 400 Bad Request - Ошибка в данных запроса или логин уже существует.

Тело ответа:

{

"message": "Логин уже используется или некорректные данные."

}

# Документация кода серверной части

## Раздел model

Файл **User**.**java** Класс **User** - сущность для хранения информации о пользователях системы

Аннотации: @Entity, @Table(name = "users") - указывают что это сущность для базы данных

Свойства:

* id (Long) - уникальный идентификатор пользователя
* name (String) - имя пользователя
* login (String) - логин пользователя
* password (String) - пароль пользователя
* createdAt (LocalDateTime) - дата и время создания пользователя

Конструктор по умолчанию: **User()**

Методы:

* Геттеры и сеттеры для всех полей:
* getId()/setId(Long id)
* getName()/setName(String name)
* getLogin()/setLogin(String login)
* getPassword()/setPassword(String password)
* getCreatedAt()/setCreatedAt(LocalDateTime createdAt)

Файл **Subscription**.java Класс **Subscription** - сущность для хранения информации об абонементах Аннотации: @Entity, @Table(name = "subscriptions")

Свойства:

* id (Long) - уникальный идентификатор абонемента
* firstName (String) - имя владельца
* lastName (String) - фамилия владельца
* login (String) - логин владельца
* phone (String) - телефон владельца
* months (Integer) - длительность абонемента в месяцах
* trainerSessions (Integer) - количество тренировок с тренером
* unlimited (Boolean) - признак безлимитного абонемента
* fullDay (Boolean) - признак полного дня
* freezeAvailable (Boolean) - возможность заморозки
* poolAccess (Boolean) - доступ в бассейн
* startDate (LocalDate) - дата начала действия
* status (String) - статус абонемента
* endDate (LocalDate) - дата окончания действия

Методы:

* Геттеры и сеттеры для всех полей

Файл **Order**.java Класс **Order** - сущность для хранения информации о заказах

Аннотации: @Entity, @Table(name = "orders")

Свойства:

* id (Long) - уникальный идентификатор заказа
* orderDate (LocalDate) - дата заказа
* items (String) - список позиций заказа
* totalPrice (BigDecimal) - общая стоимость заказа

Методы:

* Геттеры и сеттеры для всех полей:
* getId()/setId(Long id)
* getOrderDate()/setOrderDate(LocalDate orderDate)
* getItems()/setItems(String items)
* getTotalPrice()/setTotalPrice(BigDecimal totalPrice)

Файл **GroupClass**.java Класс **GroupClass** - сущность для хранения информации о групповых занятиях

Аннотации: @Entity, @Table(name = "group\_classes")

Свойства:

* id (Long) - уникальный идентификатор занятия
* classDate (LocalDate) - дата проведения занятия
* classType (String) - тип/название занятия

Методы:

* Геттеры и сеттеры для всех полей:
* getId()/setId(Long id)
* getClassDate()/setClassDate(LocalDate classDate)
* getClassType()/setClassType(String classType)

## Раздел controller

Файл **AuthController**.java, Класс: AuthController - Управление аутентификацией пользователей

Методы:

* register(User user):
* Регистрация нового пользователя
* Проверка уникальности логина
* Установка даты создания аккаунта
* login(User user):
* Авторизация пользователя
* Проверка логина и пароля

Файл **AuthController**.java, класс **AuthController** - Контроллер для аутентификации и регистрации пользователей

Зависимости:

* @Autowired private UserRepository userRepository

Основные методы:

* **register**(@RequestBody User user) - регистрация нового пользователя, проверяет существование логина и сохраняет с текущей датой
* **login**(@RequestBody User user) - аутентификация пользователя, проверяет логин и пароль

Файл **GroupClassController**.java, класс **GroupClassController** - Контроллер для управления групповыми занятиями

Зависимости:

* @Autowired private GroupClassRepository groupClassRepository

Основные методы:

* **createClass**(@RequestBody GroupClass groupClass) - создание нового группового занятия
* **getUpcomingClasses**() - получение списка предстоящих занятий после текущей даты

Файл **OrderController**.java, класс **OrderController** - Контроллер для управления заказами

Зависимости:

* @Autowired private OrderRepository orderRepository

Основные методы:

* **createOrder**(@RequestBody Order order) - создание нового заказа с установкой текущей даты

Файл **SubscriptionController**.java, класс **SubscriptionController** - Контроллер для управления абонементами

Зависимости:

* @Autowired private **SubscriptionRepository** **subscriptionRepository**

Основные методы:

* **createSubscription**(@RequestBody Subscription subscription) - создание подписки с установкой дат и статуса
* **getUserSubscriptions**(@PathVariable String login) - получение списка подписок пользователя
* **freezeSubscription**(@PathVariable Long id) - заморозка/разморозка подписки с продлением срока

## Раздел repository

Файл **UserRepository**.**java** - Репозиторий для работы с сущностью User в базе данных

Интерфейс: **UserRepository**

Наследует: **JpaRepository**<User, Long>

Методы:

* findByLoginAndPassword(String login, String password) - поиск пользователя по логину и паролю
* findByLogin(String login) - поиск пользователя по логину

Унаследованные методы от JpaRepository:

* save() - сохранение пользователя
* findById() - поиск по ID
* findAll() - получение всех пользователей
* delete() - удаление пользователя и другие стандартные методы JpaRepository

Файл **OrderRepository**.**java** - Репозиторий для работы с заказами в базе данных

Интерфейс: **OrderRepository**

Аннотация: @**Repository**

Наследует: **JpaRepository**<Order, Long>

Унаследованные методы от **JpaRepository**:

* save() - сохранение заказа
* findById() - поиск заказа по ID
* findAll() - получение всех заказов
* delete() - удаление заказа и другие стандартные методы JpaRepository

Файл **SubscriptionRepository**.**java** - Репозиторий для работы с подписками в базе данных

Интерфейс: **SubscriptionRepository**

Аннотация: @**Repository**

Наследует: **JpaRepository**<Subscription, Long>

Методы:

* **findByLogin**(**String** **login**) - получение списка подписок по логину пользователя

Унаследованные методы от JpaRepository:

* save() - сохранение подписки
* findById() - поиск подписки по ID
* findAll() - получение всех подписок
* delete() - удаление подписки и другие стандартные методы JpaRepository

Файл **GroupClassRepository**.**java** - Репозиторий для работы с групповыми занятиями в базе данных

Интерфейс: **GroupClassRepository**

Аннотация: @**Repository**

Наследует: **JpaRepository**<**GroupClass**, **Long**>

Методы:

* **findByClassDateAfterOrderByClassDate**(LocalDate date) - получение списка групповых занятий после указанной даты, отсортированных по дате

Унаследованные методы от **JpaRepository**:

* save() - сохранение группового занятия
* findById() - поиск занятия по ID
* findAll() - получение всех занятий
* delete() - удаление занятия и другие стандартные методы JpaRepository

## Остальные файлы

Файл **Application**.**java**, Класс **Application** – главный класс приложения Spring Boot, точка входа в приложение, запускает Spring Boot приложение

Аннотации:

* @**SpringBootApplication** - указывает, что это главный класс Spring Boot приложения
* @**EnableAspectJAutoProxy** - включает поддержку AOP

Методы:

* **main**(**String**[] **args**) - точка входа в приложение

Параметры: **String**[] **args** - аргументы командной строки

Файл **pom**.**xml** - Файл конфигурации Maven, который определяет структуру проекта, его зависимости и настройки сборки.

Основные зависимости:

* spring-boot-devtools - инструменты разработки Spring Boot для автоматической перезагрузки приложения при изменениях
* spring-session-jdbc - поддержка хранения сессий в базе данных
* mysql-connector-java - драйвер для работы с MySQL базой данных
* logback-classic - реализация системы логирования

Файл **logback**-**spring**.**xml** - Конфигурационный файл для настройки системы логирования Logback в Spring Boot приложении.

Параметры логирования:

* level - уровень логирования (INFO, DEBUG, ERROR и т.д.)
* pattern - формат вывода сообщений лога
* file - путь к файлу для сохранения логов (если используется файловый аппендер)
* rollingPolicy - политика ротации лог-файлов
* maxFileSize - максимальный размер файла лога
* maxHistory - количество хранимых архивных файлов

Файл **MultipleServersConfig.java,** класс **MultipleServersConfig -** Конфигурационный класс для настройки нескольких серверов **Tomcat**

Методы:

* **servletContainer**() - создает и настраивает контейнер сервлетов Tomcat
* **createConnector**() - создает дополнительный коннектор на порту 8081

Файл **LoggingAspect**.java, Класс **LoggingAspect** - для логирования вызовов методов контроллеров.

Свойства:

* **private** **static** **final** **Logger** **logger** - логгер для записи информации

Методы:

* **logMethod**(ProceedingJoinPoint joinPoint) - метод для логирования вызовов
* Параметры: **ProceedingJoinPoint** **joinPoint** - точка внедрения аспекта
* Возвращает: **Object** - результат выполнения метода

# Документация кода клиентской части

Файл: **index**.**html**

Назначение: Основной HTML-файл структуры веб-страницы

Содержит базовую разметку главной страницы

Файл: **custom**-**scripts**.**js** Назначение файла: JavaScript-файл, содержащий пользовательские скрипты для анимации и навигации на веб-странице.

Основные методы и функции:

* Обработчик кликов по навигации:
* $('.navbar-collapse ul li a').on('click', function () {
* Параметры:
* - this.hash: целевой якорь
* - scrollTop: позиция прокрутки
* - offset().top: верхняя позиция элемента
* Действие: Плавная прокрутка к секции при клике на пункт меню
* Возвращает: false (предотвращение стандартного поведения)

Метод animateNumbers:

* $.fn.animateNumbers
* Параметры:
* - stop: конечное число анимации
* - commas: флаг добавления запятых (опционально)
* - duration: длительность анимации (по умолчанию 1000мс)
* - ease: тип анимации (по умолчанию "swing")

Вспомогательные функции внутри animateNumbers:

* - step: функция обновления значения во время анимации
* - complete: функция завершения анимации

Файл: **auth**.**html**

Назначение: HTML-файл структуры веб-страницы авторизации

Содержит базовую разметку страницы авторизации и регистрации

Файл **auth.js** - Обработка аутентификации пользователей (регистрация и вход)

Основные обработчики событий:

**loginFormHandler** - Обработка входа пользователя

* Событие: **submit** на форме с id 'loginForm'

Параметры формы:

* **login**: строка (получается из элемента с id 'registerUsername')
* **password**: строка (получается из элемента с id 'loginPassword')

Функциональность:

* Отправляет **POST** запрос на http://localhost:8080/api/auth/login
* В случае успеха (получен id) перенаправляет на profile.html
* В случае ошибки показывает alert с сообщением об ошибке

**registerFormHandler** - Обработка регистрации нового пользователя

* Событие: **submit** на форме с id 'registerForm'

**Параметры формы:**

* **login**: строка (получается из элемента с id 'registerUsername')
* **password**: строка (получается из элемента с id 'registerPassword')

**Функциональность:**

* Отправляет POST запрос на http://localhost:8080/api/auth/register
* В случае успеха (получен id) очищает форму и показывает сообщение об успешной регистрации
* В случае ошибки показывает alert с сообщением об ошибке

**Общие характеристики запросов:**

* Headers: Content-Type: application/json
* Формат данных: JSON
* Обработка ответа:
* Успех: проверка наличия id в ответе
* Ошибка: вывод сообщения об ошибке через alert

Файл **profile.html** – разметка страницы для пользовательского личного кабинета

Файл **profile.js** - Управление профилем пользователя, обработка заказов, отображение групповых занятий и управление абонементом

Основные функциональные блоки:

**Управление групповыми занятиями:**

**classes.forEach()**

- Метод для итерации по списку классов

- Параметры: **classItem** - объект с информацией о классе

- Функционал: отображение информации о классах, включая форматирование даты

**Управление абонементом:**

- Компоненты:

- **freezeButton** - кнопка для заморозки/разморозки абонемента

- Свойства **subscription**.**status** для определения текущего статуса ('active' или иной)

- Методы:

- Обработчик состояния кнопки, меняющий текст в зависимости от статуса абонемента

**Обработка заказов:**

- Методы:

- **POST** запрос на '/api/orders'

- **updateCart**() - обновление корзины

- Параметры:

- **orderData** - данные заказа в формате JSON

- Свойства:

- **cart** - массив товаров в корзине

Утилиты:

- **formatDate**() - форматирование даты для отображения

- Обработка ответов сервера:

- **response**.json() - преобразование ответа в JSON

- Обработка ошибок через **catch**

Файл **admin.html** - Страница для создания абонемента в административной панели и публикации групповых занятий

Файл **admin**.**js** - Обработка логики создания и управления абонементами на стороне клиента, а также создание групповых занятий

Глобальные объекты:

* **classData** - объект для хранения данных о классах

Основные обработчики событий:

* **DOMContentLoaded** - колбэк-функция - Инициализация скрипта при загрузке DOM
  + Параметры: **function**()

Event Listeners:

* **input**[type="radio"].addEventListener('change') - Отслеживание изменений в radio-кнопках для обновления цены
* **monthsInput**.**addEventListener**('input') - Обработка изменений в поле выбора количества месяцев

Методы:

* **updatePrice**() - пересчитывает стоимость при изменении опций
* **getMonthWord**(value) - возвращает правильное склонение слова "месяц"
* **updatePrice**() - обновляет отображаемую цену

Вспомогательные методы:

* **updatePrice**() - Пересчет и обновление отображаемой цены абонемента
* **getMonthWord**(value) - Возвращает корректное склонение слова "месяц"
  + Параметры: **value** - количество месяцев

# Приложение серверной части

## Раздел aspect

**LoggingAspect.java**

package com.example.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

@Around("execution(\* com.example.controller.\*.\*(..))")

public Object logMethod(ProceedingJoinPoint joinPoint) throws Throwable {

String methodName = joinPoint.getSignature().getName();

logger.info("Вызов метода: " + methodName);

Object result = joinPoint.proceed();

logger.info("Метод " + methodName + " завершен");

return result;}}

## Раздел config

**MultipleServersConfig.java**

package com.example.config;

import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;

import org.springframework.boot.web.servlet.server.ServletWebServerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.boot.web.embedded.tomcat.ConnectorStartFailedException;

import org.apache.catalina.connector.Connector;

@Configuration

public class MultipleServersConfig {

@Bean

public ServletWebServerFactory servletContainer() {

TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory();

tomcat.addAdditionalTomcatConnectors(createConnector());

return tomcat;}

private Connector createConnector() {

Connector connector = new Connector(TomcatServletWebServerFactory.DEFAULT\_PROTOCOL);

connector.setPort(8081);

return connector;}}

## Раздел controller

**AuthController.java**

package com.example.controller;

import com.example.model.User;

import com.example.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDateTime;

@RestController

@CrossOrigin

@RequestMapping("/api/auth")

public class AuthController {

@Autowired

private UserRepository userRepository;

@PostMapping("/register")

public ResponseEntity<?> register(@RequestBody User user) {

if (userRepository.findByLogin(user.getLogin()) != null) {

return ResponseEntity.badRequest().body("Login already exists");}

user.setCreatedAt(LocalDateTime.now());

return ResponseEntity.ok(userRepository.save(user));}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody User user) {

User existingUser = userRepository.findByLoginAndPassword(user.getLogin(), user.getPassword());

if (existingUser != null) {

return ResponseEntity.ok(existingUser);}

return ResponseEntity.badRequest().body("Invalid credentials");}}

**GroupClassController.java**

package com.example.controller;

import com.example.model.GroupClass;

import com.example.repository.GroupClassRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

import java.util.List;

@RestController

@RequestMapping("/api/classes")

public class GroupClassController {

@Autowired

private GroupClassRepository groupClassRepository;

@PostMapping

public GroupClass createClass(@RequestBody GroupClass groupClass) {

return groupClassRepository.save(groupClass);}

@GetMapping

public List<GroupClass> getUpcomingClasses() {

return groupClassRepository.findByClassDateAfterOrderByClassDate(LocalDate.now());}}

**OrderController.java**

package com.example.controller;

import com.example.model.Order;

import com.example.repository.OrderRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

@RestController

@RequestMapping("/api/orders")

public class OrderController {

@Autowired

private OrderRepository orderRepository;

@PostMapping

public Order createOrder(@RequestBody Order order) {

order.setOrderDate(LocalDate.now());

return orderRepository.save(order);}}

**SubscribtionController.java**  
package com.example.controller;

import com.example.model.Subscription;

import com.example.repository.SubscriptionRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

import java.util.List;

@RestController

@RequestMapping("/api/subscriptions")

public class SubscriptionController {

@Autowired

private SubscriptionRepository subscriptionRepository;

@PostMapping

public Subscription createSubscription(@RequestBody Subscription subscription) {

subscription.setStartDate(LocalDate.now());

subscription.setEndDate(subscription.getStartDate().plusMonths(subscription.getMonths()));

subscription.setStatus("active");

return subscriptionRepository.save(subscription);}

@GetMapping("/user/{login}")

public List<Subscription> getUserSubscriptions(@PathVariable String login) {

return subscriptionRepository.findByLogin(login);}

@PostMapping("/{id}/freeze")

public Subscription freezeSubscription(@PathVariable Long id) {

Subscription subscription = subscriptionRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Subscription not found"));

if ("active".equals(subscription.getStatus())) {

subscription.setStatus("frozen");

subscription.setEndDate(subscription.getEndDate().plusDays(30));} else if ("frozen".equals(subscription.getStatus())) {

subscription.setStatus("active");}

return subscriptionRepository.save(subscription);}}

## Раздел model

**GroupClass.java**

package com.example.model;

import javax.persistence.\*;

import java.time.LocalDate;

@Entity

@Table(name = "group\_classes")

public class GroupClass {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private LocalDate classDate;

private String classType;

public Long getId() {

return id;}

public void setId(Long id) {

this.id = id;}

public LocalDate getClassDate() {

return classDate;}

public void setClassDate(LocalDate classDate) {

this.classDate = classDate;}

public String getClassType() {

return classType;}

public void setClassType(String classType) {

this.classType = classType;}}

**Order.java**

package com.example.model;

import javax.persistence.\*;

import java.math.BigDecimal;

import java.time.LocalDate;

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private LocalDate orderDate;

private String items;

private BigDecimal totalPrice;

public Long getId() {

return id;}

public void setId(Long id) {

this.id = id;}

public LocalDate getOrderDate() {

return orderDate;}

public void setOrderDate(LocalDate orderDate) {

this.orderDate = orderDate;}

public String getItems() {

return items;}

public void setItems(String items) {

this.items = items;}

public BigDecimal getTotalPrice() {

return totalPrice;}

public void setTotalPrice(BigDecimal totalPrice) {

this.totalPrice = totalPrice;}}

**Subscription.java**

package com.example.model;

import javax.persistence.\*;

import java.time.LocalDate;

@Entity

@Table(name = "subscriptions")

public class Subscription {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String login;

private String phone;

private Integer months;

private Integer trainerSessions;

private Boolean unlimited;

private Boolean fullDay;

private Boolean freezeAvailable;

private Boolean poolAccess;

private LocalDate startDate;

private String status;

private LocalDate endDate;

public Long getId() {

return id;}

public void setId(Long id) {

this.id = id;}

public String getFirstName() {

return firstName;}

public void setFirstName(String firstName) {

this.firstName = firstName;}

public String getLastName() {

return lastName;}

public void setLastName(String lastName) {

this.lastName = lastName;}

public String getLogin() {

return login;}

public void setLogin(String login) {

this.login = login;}

public String getPhone() {

return phone;}

public void setPhone(String phone) {

this.phone = phone;}

public Integer getMonths() {

return months;}

public void setMonths(Integer months) {

this.months = months;}

public Integer getTrainerSessions() {

return trainerSessions;}

public void setTrainerSessions(Integer trainerSessions) {

this.trainerSessions = trainerSessions;}

public Boolean getUnlimited() {

return unlimited;}

public void setUnlimited(Boolean unlimited) {

this.unlimited = unlimited;}

public Boolean getFullDay() {

return fullDay;}

public void setFullDay(Boolean fullDay) {

this.fullDay = fullDay;}

public Boolean getFreezeAvailable() {

return freezeAvailable;}

public void setFreezeAvailable(Boolean freezeAvailable) {

this.freezeAvailable = freezeAvailable;}

public Boolean getPoolAccess() {

return poolAccess;}

public void setPoolAccess(Boolean poolAccess) {

this.poolAccess = poolAccess;}

public LocalDate getStartDate() {

return startDate;}

public void setStartDate(LocalDate startDate) {

this.startDate = startDate;}

public String getStatus() {

return status;}

public void setStatus(String status) {

this.status = status;}

public LocalDate getEndDate() {

return endDate;}

public void setEndDate(LocalDate endDate) {

this.endDate = endDate;}}

**User.java**

package com.example.model;

import javax.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name")

private String name;

@Column(name = "login")

private String login;

@Column(name = "password")

private String password;

@Column(name = "created\_at")

private LocalDateTime createdAt;

public User() {}

public Long getId() {

return id;}

public void setId(Long id) {

this.id = id;}

public String getName() {

return name;}

public void setName(String name) {

this.name = name;}

public String getLogin() {

return login;}

public void setLogin(String login) {

this.login = login;}

public String getPassword() {

return password;}

public void setPassword(String password) {

this.password = password;}

public LocalDateTime getCreatedAt() {

return createdAt;}

public void setCreatedAt(LocalDateTime createdAt) {

this.createdAt = createdAt;}}

## Раздел repository

**GroupClassRepository.java**

package com.example.repository;

import com.example.model.GroupClass;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.time.LocalDate;

import java.util.List;

@Repository

public interface GroupClassRepository extends JpaRepository<GroupClass, Long> {

List<GroupClass> findByClassDateAfterOrderByClassDate(LocalDate date);}

**OrderRepository.java**

package com.example.repository;

import com.example.model.Order;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {}

**SubscriptionRepository.java**

package com.example.repository;

import com.example.model.Subscription;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface SubscriptionRepository extends JpaRepository<Subscription, Long> {

List<Subscription> findByLogin(String login);}

**UserRepository.java**

package com.example.repository;

import com.example.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findByLoginAndPassword(String login, String password);

User findByLogin(String login);}

## Остальные файлы

**Application.java**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.EnableAspectJAutoProxy;

@SpringBootApplication

@EnableAspectJAutoProxy

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);}}

**Application.properties**

server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/cursejava

spring.datasource.username=root

spring.datasource.password=aaaa

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

logging.config=classpath:logback-spring.xml

**Application-admin.properties**

server.port=8081

spring.application.name=admin-panel

**Logback-spring.xml**

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<property name="LOG\_PATH" value="./logs"/>

<appender name="GENERAL" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${LOG\_PATH}/general.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${LOG\_PATH}/archived/general.%d{yyyy-MM-dd}.log</fileNamePattern>

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="SECURITY" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${LOG\_PATH}/security.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${LOG\_PATH}/archived/security.%d{yyyy-MM-dd}.log</fileNamePattern>

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level - %msg%n</pattern>

</encoder>

</appender>

<logger name="com.example.security" level="INFO">

<appender-ref ref="SECURITY"/>

</logger>

<root level="INFO">

<appender-ref ref="GENERAL"/>

</root>

</configuration>

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>gym-management</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.0</version>

</parent>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.28</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-jdbc</artifactId>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

# Приложение клиентской части:

Папка **css** – таблицы стилей, необходимые для корректной работы стилей и анимаций в различных разделах проекта

Папка **fonts** – шрифты, необходимые для проекта

Папка **images** – изображения, используемые для сайта

**Js:**

* **bootstrap**.**min**.**js** - минифицированная версия Bootstrap, фреймворка для создания пользовательских интерфейсов
* **jquery** и связанные библиотеки (**jquery**.**inview**.**min**.**js**, **jquery**.**isotope**.**min**.**js**, **jquery**.**prettyphoto**.**js**) - набор jQuery-плагинов для различных функций интерфейса.

Вспомогательные библиотеки для улучшения пользовательского опыта:

* **mousescroll**.js
* **smoothscroll**.js
* **respond**.**min**.js

HTML:

**Index**.html – разметка главной страницы сайта

**Profile**.html – разметка пользовательского личного кабинета

**Admin**.html – разметка админ панели

**Auth**.html – разметка страницы авторизации и регистрации

**JS**:

**Custom**-**scripts**.js – скрипты для index.html

jQuery(function ($) {

'use strict';

$(window).scroll(function (event) {

Scroll();});

$('.navbar-collapse ul li a').on('click', function () {

$('html, body').animate({

scrollTop: $(this.hash).offset().top - 5

}, 1000);

return false;});

function Scroll() {

var contentTop = [];

var contentBottom = [];

var winTop = $(window).scrollTop();

var rangeTop = 200;

var rangeBottom = 500;

$('.navbar-collapse').find('.scroll a').each(function () {

contentTop.push($($(this).attr('href')).offset().top);

contentBottom.push($($(this).attr('href')).offset().top + $($(this).attr('href')).height());})

$.each(contentTop, function (i) {

if (winTop > contentTop[i] - rangeTop) {

$('.navbar-collapse li.scroll')

.removeClass('active')

.eq(i).addClass('active');}})};

$('#tohash').on('click', function () {

$('html, body').animate({

scrollTop: $(this.hash).offset().top - 5

}, 1000);

return false;});

new WOW().init();

smoothScroll.init();

$(window).load(function () {

'use strict';

var $portfolio\_selectors = $('.portfolio-filter >li>a');

var $portfolio = $('.portfolio-items');

$portfolio.isotope({

itemSelector: '.portfolio-item',

layoutMode: 'fitRows'});

$portfolio\_selectors.on('click', function () {

$portfolio\_selectors.removeClass('active');

$(this).addClass('active');

var selector = $(this).attr('data-filter');

$portfolio.isotope({

filter: selector});

return false;});});

$(document).ready(function () {

$.fn.animateNumbers = function (stop, commas, duration, ease) {

return this.each(function () {

var $this = $(this);

var start = parseInt($this.text().replace(/,/g, ""));

commas = (commas === undefined) ? true : commas;

$({

value: start}).animate({

value: stop}, {

duration: duration == undefined ? 1000 : duration,

easing: ease == undefined ? "swing" : ease,

step: function () {

$this.text(Math.floor(this.value));

if (commas) {

$this.text($this.text().replace(/(\d)(?=(\d\d\d)+(?!\d))/g, "$1,"));}},

complete: function () {

if (parseInt($this.text()) !== stop) {

$this.text(stop);

if (commas) {

$this.text($this.text().replace(/(\d)(?=(\d\d\d)+(?!\d))/g, "$1,"));}}}});});};

$('.business-stats').bind('inview', function (event, visible, visiblePartX, visiblePartY) {

var $this = $(this);

if (visible) {

$this.animateNumbers($this.data('digit'), false, $this.data('duration'));

$this.unbind('inview');}});});

$("a[rel^='prettyPhoto']").prettyPhoto({

social\_tools: false});});

**Profile**.js – скрипты для просмотра групповых занятий, оформления заказа

let cart = [];

const prices = {

'Протеин': 2000,

'BCAA': 1500,

'Креатин': 1200,

'Витамины': 800,

'Омега-3': 1000,

'L-карнитин': 1800,

'Изотоник': 900,

'Гейнер': 2500};

document.addEventListener('DOMContentLoaded', function () {

loadSubscriptionInfo();

loadGroupClasses();

updateCart();});

function loadSubscriptionInfo() {

const login = localStorage.getItem('userLogin');

fetch(`/api/subscriptions/user/${login}`)

.then(response => response.json())

.then(data => {

if (data && data.length > 0) {

const subscription = data[0];

document.getElementById('status').textContent = subscription.status;

document.getElementById('end-date').textContent = subscription.endDate;

const freezeButton = document.getElementById('freeze-button');

freezeButton.textContent = subscription.status === 'active' ?

'Заморозить абонемент' : 'Разморозить абонемент';}})

.catch(error => console.error('Error:', error));}

function loadGroupClasses() {

fetch('/api/classes')

.then(response => response.json())

.then(classes => {

const container = document.getElementById('classes-list');

container.innerHTML = '';

classes.forEach(classItem => {

const card = document.createElement('div');

card.className = 'class-card';

card.innerHTML = `

<p>${formatDate(classItem.classDate)}</p>

<p>${classItem.classType}</p>

`;

container.appendChild(card);});})

.catch(error => console.error('Error:', error));}

function formatDate(dateString) {

const date = new Date(dateString);

return date.toLocaleDateString('ru-RU');}

function addToCart(item, price) {

const existingItem = cart.find(cartItem => cartItem.name === item);

if (existingItem) {

existingItem.quantity++;} else {

cart.push({ name: item, price: price, quantity: 1 });}

updateCart();}

function updateCart() {

const cartItems = document.getElementById('cart-items');

const cartTotal = document.getElementById('cart-total');

cartItems.innerHTML = '';

let total = 0;

cart.forEach(item => {

const itemDiv = document.createElement('div');

itemDiv.className = 'cart-item';

itemDiv.innerHTML = `

<span>${item.name}</span>

<div class="quantity-controls">

<button onclick="updateQuantity('${item.name}', -1)">-</button>

<span>${item.quantity}</span>

<button onclick="updateQuantity('${item.name}', 1)">+</button>

</div>

<span>${item.price \* item.quantity} ₽</span>

`;

cartItems.appendChild(itemDiv);

total += item.price \* item.quantity;});

cartTotal.textContent = total;}

function updateQuantity(itemName, change) {

const item = cart.find(i => i.name === itemName);

if (item) {

item.quantity += change;

if (item.quantity <= 0) {

cart = cart.filter(i => i.name !== itemName);}

updateCart();}}

function placeOrder() {

if (cart.length === 0) {

alert('Корзина пуста');

return;}

const orderData = {

items: cart.map(item => `${item.name} "${item.quantity}"`).join(', '),

totalPrice: cart.reduce((sum, item) => sum + (item.price \* item.quantity), 0)};

fetch('/api/orders', {

method: 'POST',

headers: {

'Content-Type': 'application/json',},

body: JSON.stringify(orderData)})

.then(response => response.json())

.then(data => {

alert('Заказ успешно оформлен');

cart = [];

updateCart();})

.catch(error => alert('Ошибка при оформлении заказа: ' + error));}

function freezeSubscription() {

const subscriptionId = localStorage.getItem('subscriptionId');

fetch(`/api/subscriptions/${subscriptionId}/freeze`, {

method: 'POST'})

.then(response => response.json())

.then(data => {

loadSubscriptionInfo();})

.catch(error => alert('Ошибка при изменении статуса абонемента: ' + error));}

**Auth**.js – скрипты для анимации страницы авторизации при клике, а также верификации значений регистрации и авторизации

const container = document.getElementById('container');

const registerForm = document.getElementById('registerForm');

const loginForm = document.getElementById('loginForm');

document.getElementById('register').addEventListener('click', () => {

container.classList.add("active");});

document.getElementById('login').addEventListener('click', () => {

container.classList.remove("active");});

document.getElementById('registerForm').addEventListener('submit', function (e) {

e.preventDefault();

const userData = {

name: document.getElementById('registerName').value,

login: document.getElementById('registerUsername').value,

password: document.getElementById('registerPassword').value};

fetch('http://localhost:8080/api/auth/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json',},

body: JSON.stringify(userData)})

.then(response => response.json())

.then(data => {

if (data.id) {

alert('Registration successful');

document.getElementById('registerForm').reset();} else {

alert('Registration failed: ' + data);}})

.catch(error => alert('Error: ' + error));});

document.getElementById('loginForm').addEventListener('submit', function (e) {

e.preventDefault();

const userData = {

login: document.getElementById('loginUsername').value,

password: document.getElementById('loginPassword').value};

fetch('http://localhost:8080/api/auth/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json',},

body: JSON.stringify(userData)})

.then(response => response.json())

.then(data => {

if (data.id) {

window.location.href = 'profile.html';} else {

alert('Login failed: ' + data);}})

.catch(error => alert('Error: ' + error));});

**Admin**.js – скрипты для создания абонементов и публикации групповых занятий:

document.addEventListener('DOMContentLoaded', function () {

initSubscriptionForm();

initClassForm();});

function initSubscriptionForm() {

const form = document.getElementById('subscription-form');

const monthsInput = document.getElementById('months');

const sessionsInput = document.getElementById('trainer-sessions');

function updatePrice() {

let total = 0;

const months = parseInt(monthsInput.value);

switch (months) {

case 1: total = 2000; break;

case 2: total = 4000; break;

case 3: total = 4500; break;

case 4: total = 5500; break;

case 5: total = 6800; break;

case 6: total = 7000; break;

case 7: total = 9000; break;

case 8: total = 9200; break;

case 9: total = 1000; break;

case 10: total = 11000; break;

case 11: total = 11800; break;

case 12: total = 12000; break;}

total += parseInt(sessionsInput.value) \* 400;

if (document.querySelector('input[name="visits"]:checked')?.value === 'unlimited') {

total += 1500;}

if (document.querySelector('input[name="time"]:checked')?.value === 'full') {

total += 750;}

if (document.querySelector('input[name="freeze"]:checked')?.value === 'yes') {

total += 1500;}

if (document.querySelector('input[name="pool"]:checked')?.value === 'yes') {

total += 1000;}

document.getElementById('total-price').textContent = total;}

monthsInput.addEventListener('input', function () {

document.getElementById('months-value').textContent = `${this.value} ${getMonthWord(this.value)}`;

updatePrice();});

sessionsInput.addEventListener('input', function () {

document.getElementById('sessions-value').textContent =

`${this.value} ${getSessionWord(this.value)}`;

updatePrice();});

document.querySelectorAll('input[type="radio"]').forEach(radio => {

radio.addEventListener('change', updatePrice);});

form.addEventListener('submit', function (e) {

e.preventDefault();

const subscriptionData = {

firstName: document.getElementById('first-name').value,

lastName: document.getElementById('last-name').value,

login: document.getElementById('login').value,

phone: document.getElementById('phone').value,

months: parseInt(monthsInput.value),

trainerSessions: parseInt(sessionsInput.value),

unlimited: document.querySelector('input[name="visits"]:checked').value === 'unlimited',

fullDay: document.querySelector('input[name="time"]:checked').value === 'full',

freezeAvailable: document.querySelector('input[name="freeze"]:checked').value === 'yes',

poolAccess: document.querySelector('input[name="pool"]:checked').value === 'yes',

startDate: document.getElementById('start-date').value};

fetch('/api/subscriptions', {

method: 'POST',

headers: {

'Content-Type': 'application/json',},

body: JSON.stringify(subscriptionData)})

.then(response => response.json())

.then(data => {

alert('Абонемент успешно создан');

form.reset();})

.catch(error => alert('Ошибка при создании абонемента: ' + error));});}

function initClassForm() {

const form = document.getElementById('class-form');

form.addEventListener('submit', function (e) {

e.preventDefault();

const classData = {

classDate: document.getElementById('class-date').value,

classType: document.querySelector('input[name="class-type"]:checked').value};

fetch('/api/classes', {

method: 'POST',

headers: {

'Content-Type': 'application/json',},

body: JSON.stringify(classData)})

.then(response => response.json())

.then(data => {

alert('Занятие успешно создано');

form.reset();})

.catch(error => alert('Ошибка при создании занятия: ' + error));});}

function getMonthWord(number) {

const cases = [2, 0, 1, 1, 1, 2];

const titles = ['месяц', 'месяца', 'месяцев'];

return titles[(number % 100 > 4 && number % 100 < 20) ? 2 :

cases[(number % 10 < 5) ? number % 10 : 5]];}

function getSessionWord(number) {

const cases = [2, 0, 1, 1, 1, 2];

const titles = ['занятие', 'занятия', 'занятий'];

return titles[(number % 100 > 4 && number % 100 < 20) ? 2 :

cases[(number % 10 < 5) ? number % 10 : 5]];}