МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики и кибернетики
Кафедра технической кибернетики

# ОТЧЕТ

## Проектирование приложения «FarmLab»

Выполнил студент
группы 6303-010302D
Корнев Никита
Гафутулина Эльвира

САМАРА 2025

**Схема взаимодействия компонентов**

**1. Клиент**

Клиентская часть системы представляет собой интерфейс, через который пользователи (пациенты, врачи) взаимодействуют с приложением. Клиенты могут использовать веб-браузеры для доступа к функционалу системы.

**2. Сервер приложений**

Сервер приложений обрабатывает бизнес-логику и управляет запросами от клиентов. Он отвечает за:

- Обработку запросов на регистрацию пациентов.
- Назначение услуг и запись на прием.
- Управление историей болезни, включая доступ врачей и пациентов к данным.

**3. База данных**

База данных хранит всю информацию о пациентах, врачах, услугах и результатах анализов. Она включает следующие таблицы:

- **Таблица "Клиенты"**: Хранит информацию о пациентах.
- **Таблица "Врач"**: Содержит данные о врачах, их специализациях.
- **Таблица "История болезни"**: Сохраняет информацию о диагнозах и лечении клиентов.
- **Таблица "Список записей"**: Хранит записи на прием к врачам, разграничивая доступ для клиентов и врачей.
- **Таблица "Результаты анализов"**: Содержит результаты лабораторных исследований пациентов.

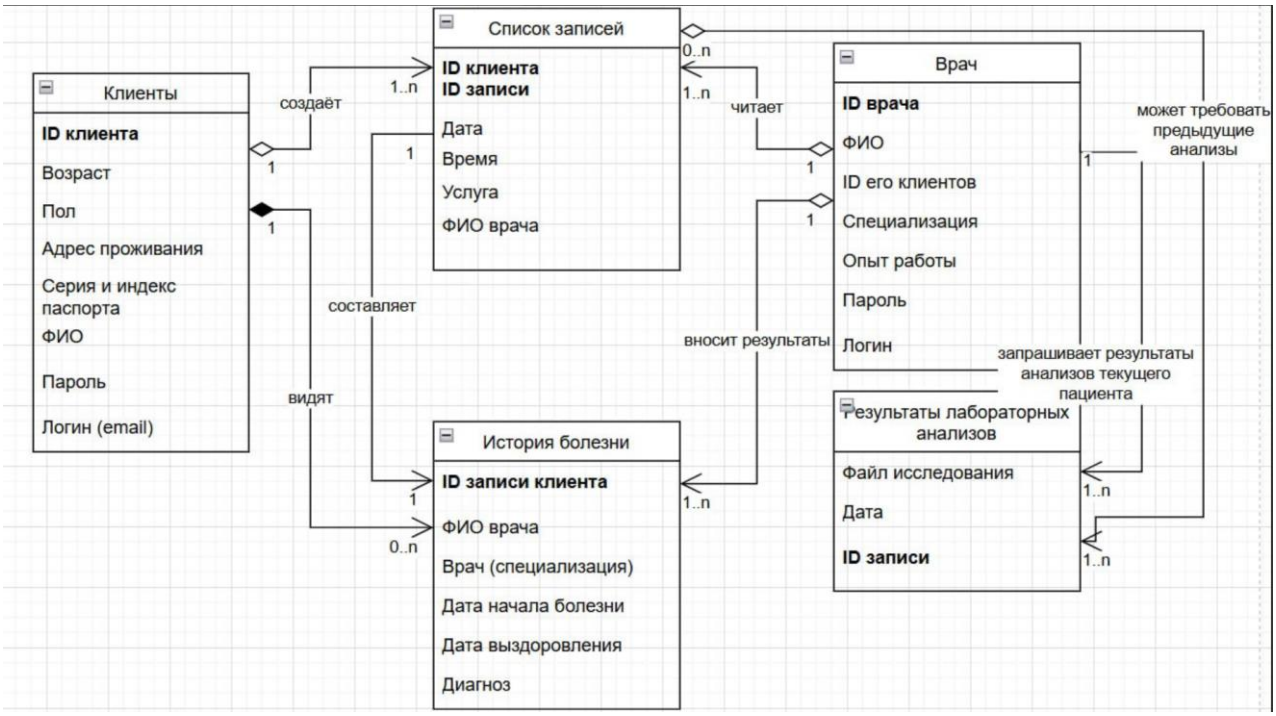

**Схема связей:**

- **Клиенты** (1) — (1) **История болезни**
- **Клиенты** (1) — (N) **Результаты анализов**
- **Клиенты** (1) — (N) **Список записей**
- **Врач** (1) — (N) **История болезни**
- **Врач** (1) — (N) **Список записей**
- **Врач** (1) — (N) **История болезни**
- **Список записей** (1) — (1) **История болезни**

**Взаимодействие компонентов**

1. **Запрос от клиента**: Клиент отправляет запрос на сервер (например, для записи на прием).
2. **Обработка запроса сервером**: Сервер принимает запрос, обрабатывает его с использованием бизнес-логики и обращается к базе данных для извлечения необходимой информации.
3. **Ответ базы данных**: База данных возвращает запрашиваемую информацию серверу.
4. **Ответ сервера клиенту:** Сервер формирует ответ и отправляет его обратно клиенту (как пример, запись сохранена).

**Структура API**

Основные методы HTTP
**GET**: Получение информации о ресурсах.
**POST**: Создание нового ресурса.

| | | |
|---|---|---|
| GET / POST / PUT / DELETE | / | Публичный доступ ко всем запросам на главной странице |
| POST | /auth/login/**,/api/auth/login/** | Авторизация пользователя (логин) — публичный эндпоинт |
| POST | /auth/register/client,/api/auth/register/client | Регистрация клиента — публичный эндпоинт |
| POST | /auth/register/doctor,/api/auth/register/doctor | Регистрация доктора — доступно только администратору |
| POST | /auth/register/admin,/api/auth/register/admin | Регистрация администратора — публичный эндпоинт (временно) |
| GET | /client/dashboard | Доступ к личному кабинету клиента — доступно для клиента, врача и пациента |
| POST | /auth/logout | Выход из аккаунта — требует аутентификации |
| GET | /api/v1/clients/{id} | Получение информации о клиенте по ID — доступно для клиента(по своему id), врача, администратора |
| GET | /api/v1/disease-history/{id} | Получение истории болезней клиента по ID — доступно для клиента, врача, администратора |
| GET | /api/v1/analysis-results/{id} | Получение результатов анализов клиента по ID — доступно для клиента, врача, администратора |
| GET | /api/v1/appointment-records/{recordId} | Получение записи о приёме по ID — доступно для клиента, врача, администратора |

| | | |
|---|---|---|
| GET | /api/v1/doctors/{id} | Получение информации о враче по ID — доступно для клиента, врача, администратора |
| GET / POST / PUT / DELETE | /web/appointments/** | Веб-интерфейс записей на приём — доступно для клиента, врача, администратора |
| GET / POST / PUT / DELETE | /web/analysis-results/api/**,/web/analysis-results/** | Веб-интерфейс результатов анализов — доступно для клиента(может просматривать только свои анализы), доктора, администратора |
| POST / PUT / DELETE | /api/v1/doctors/** | Создание / обновление / удаление врачей — доступно только администратору |
| GET | /api/v1/** | Чтение данных из всех таблиц — доступно для доктора и администратора |
| POST / PUT / DELETE | /api/v1/** | Изменение данных в таблицах — доступно для доктора и администратора |
| GET / POST / PUT / DELETE | /css/**,/js/**,/images/**,/favicon.ico | Статические ресурсы — публичный доступ |

**Выбор сервера приложений и СУБД.**

Сервер –Windows , СУБД – PostgreSQL.

Back – Java+Spring boot+ Thymeleaf, Front –JS+html+css.

**Архитектура приложения:**

1. Клиентская часть (HTML + JS):
    - Отображение форм регистрации, входа, профиля и страницы с информацией о записях, истории болезни и результатах анализа пациента.
    - Реализует взаимодействие с сервером через fetch-запросы с JWT-аутентификацией.
2. Серверная часть (Java + Spring+ Thymeleaf):
    - Обрабатывает все HTTP-запросы.

- Реализует REST API.
- Обеспечивает JWT-аутентификацию и управление пользователями, событиями и реакциями.

3. База данных (PostgreSQL):
    - Хранит информацию о пользователях (пациенты, врачи, администраторы), записи к врачам, медицинские карты пациентов, анализы пациентов, хранение с последующим удалением токена, если был совершён выход из системы пользователем.
4. Docker-контейнеризация:
    - postgres (PostgreSQL): Используется официальный образ postgres:latest, в котором настроена база данных с именем mydatabase, пользователем postgres и паролем 1234. Контейнер предоставляет стандартный порт 5432 для подключения, а также сохраняет данные в Docker volume для их сохранения между перезапусками контейнера.
    - app (Spring Boot приложение): Приложение упаковано в отдельный Docker-образ. В контейнер копируется собранный JAR-файл приложения, который запускается с помощью команды java -jar app.jar. Сервис использует имя хоста postgres для подключения к базе данных и пробрасывает порт 8080 для доступа к REST API и веб-интерфейсу через браузер.
    - Сетевая связность: Оба контейнера работают в одной пользовательской сети Docker, что обеспечивает сетевую доступность между ними по имени сервиса (postgres).

**1. Общий принцип работы приложения:**

1) Зарегистрировать админа – это откроет доступ к регистрации доктора
2) Зарегистрировать доктора и клиента (для клиента входной токен не нужен)
3) Пока есть альтернатива в виде эндпоинта POST в таблицах “clients” и “doctors”. Из-за неполной ясности с верстkой, пока принято решение оставить эти избыточные методы.
4) При помощи эндпоинта POST в таблице disease_history добавить историю болезни с привязкой на существующие идентификаторы доктора и клиента. Без указания существующих id запись в таблице создана не будет.
5) После создания записи в disease_history можно заметить, что значение атрибута-объекта appointment_records = null, это нормально. Это реализовано для того, чтобы уже после создания истории болезни установить нестрогую связь с таблицей appointment_records, по логике приложения вполне допустима регистрация пользователя без обязательной записи на прием. Пока не создана запись в appointment_records с привязкой на существующую запись в disease_history при помощи идентификатора, она не будет добавлена в запись disease_history как

объект. Это как раз определяется аннотацией *OneToOne* в entity классе AppointmentRecord, где запись связывает поле disease_history_id из appointmentRecord с полем record_id в DiseaseHistory.

6) При удалении DiseaseHistory удаляются и связанные записи в AppointmentRecord, при удалении записи в AppointmentRecord из DiseaseHistory удаляется только поле – ссылка на запись в AppointmentRecord, сохраняя целостность структуры и допуская нулевое значение. Это достигается за счет аннотации *OneToMany* в DiseaseHistory на запись в AppointmentRecord. Аннотация *ManyToOne* в AppointmentRecord допускает наличие нескольких записей в этом классе, привязанных к конкретным объектам Client и Doctor – то есть клиент и доктор могут иметь несколько записей в AppointmentRecord.

7) Класс AnalysisResult связан с классом Client при помощи аннотации *ManyToOne*, опять же обеспечивая допустимость привязки нескольких записей из AnalysisResult к одной записи из Client.

Таблица эндпоинтов приложения

| Role | CRUD | Название таблицы | | | | |
|---|---|---|---|---|---|---|
| | | doctors | disease_history | appointment_records | analysis_results | clients |
| CLIENT | GET{id} | + | + | + | + | + |
| | GET | - | - | - | - | - |
| | POST | - | - | + | - | - |
| | PUT | - | - | - | - | - |
| | DELETE | - | - | - | - | - |
| | | | | | | |
| DOCTOR | GET{id} | + | + | + | + | + |
| | GET | + | + | + | + | + |
| | POST | - | + | + | + | + |
| | PUT | - | + | + | + | + |
| | DELETE | - | + | + | + | + |
| | | | | | | |
| ADMIN | GET{id} | + | + | + | + | + |
| | GET | + | + | + | + | + |
| | POST | + | + | + | + | + |
| | PUT | + | + | + | + | + |
| | DELETE | + | + | + | + | + |

Скрины работающего приложения

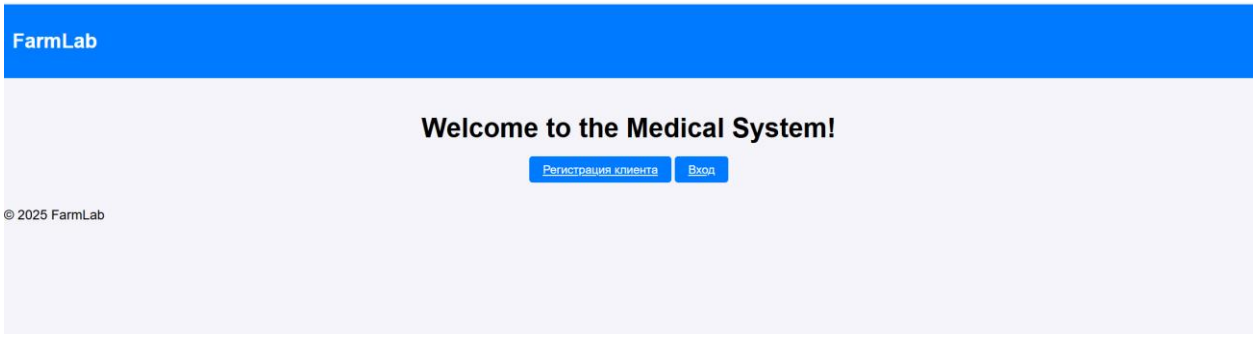


Рисунок 1 – основная страница

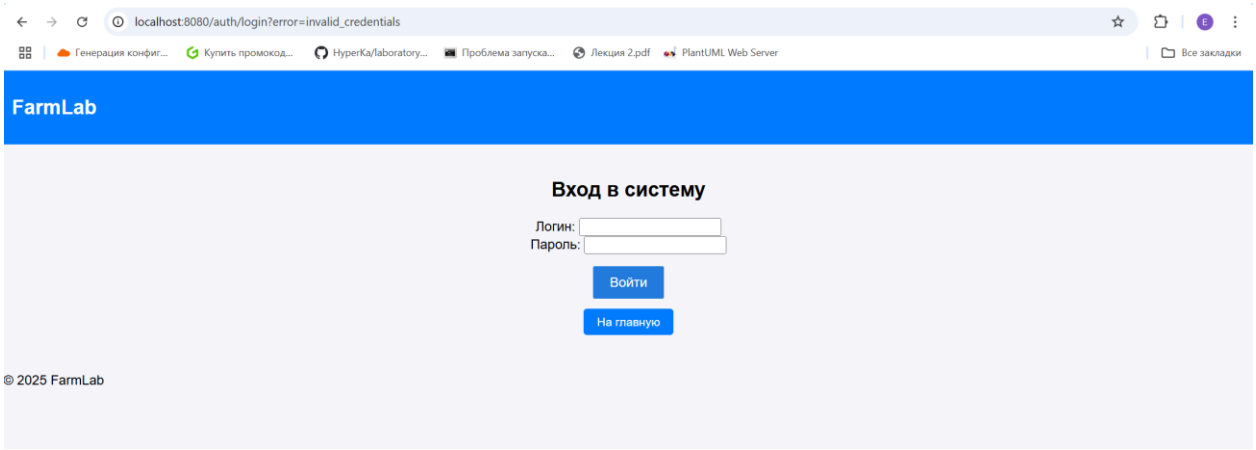


Рисунок 2 – вход в систему

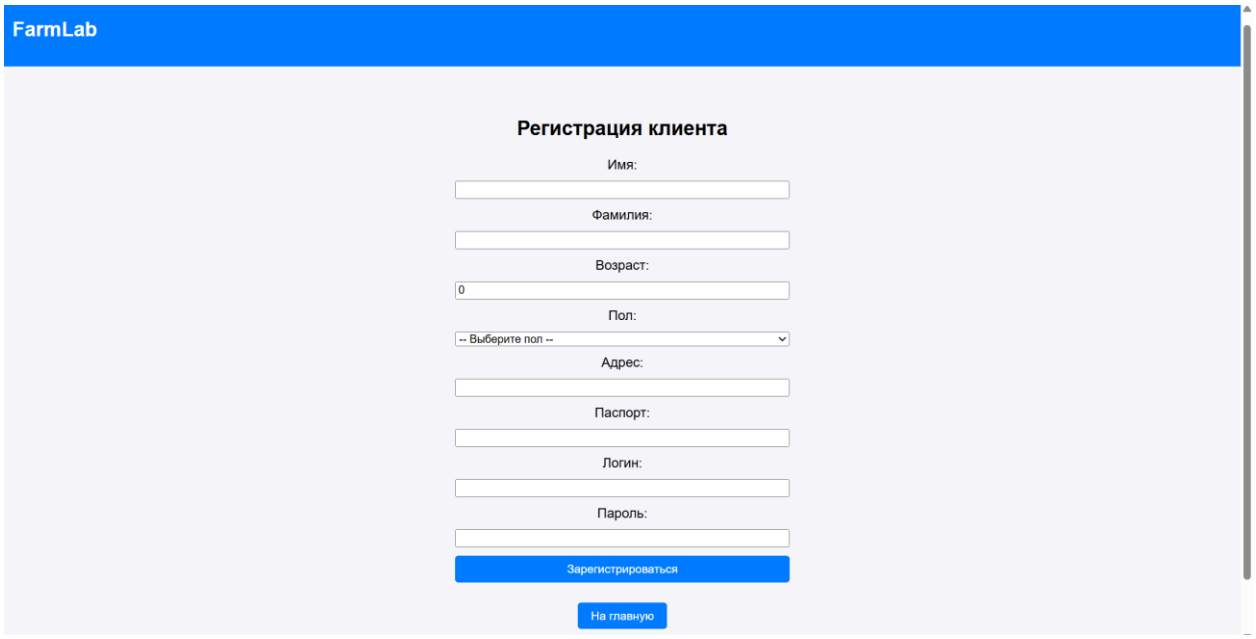


Рисунок 3 – регистрация клиента

Личный кабинет клиента с возможностью добавить новую запись

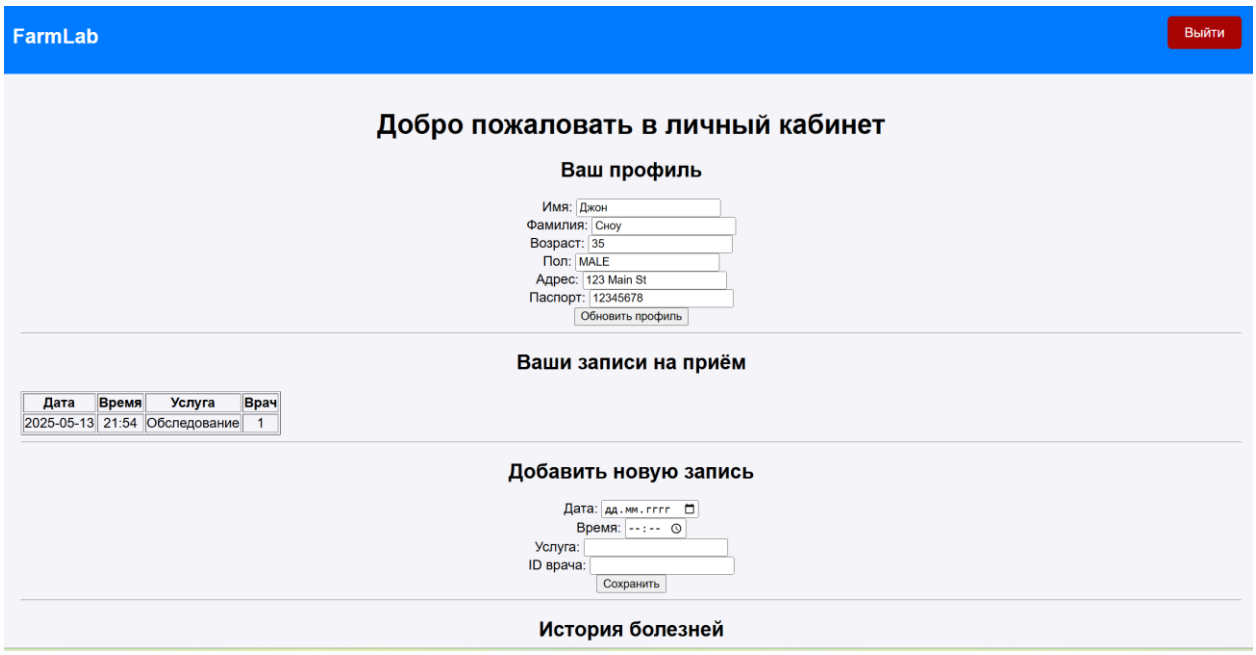


Рисунок 4 – личный кабинет клиента

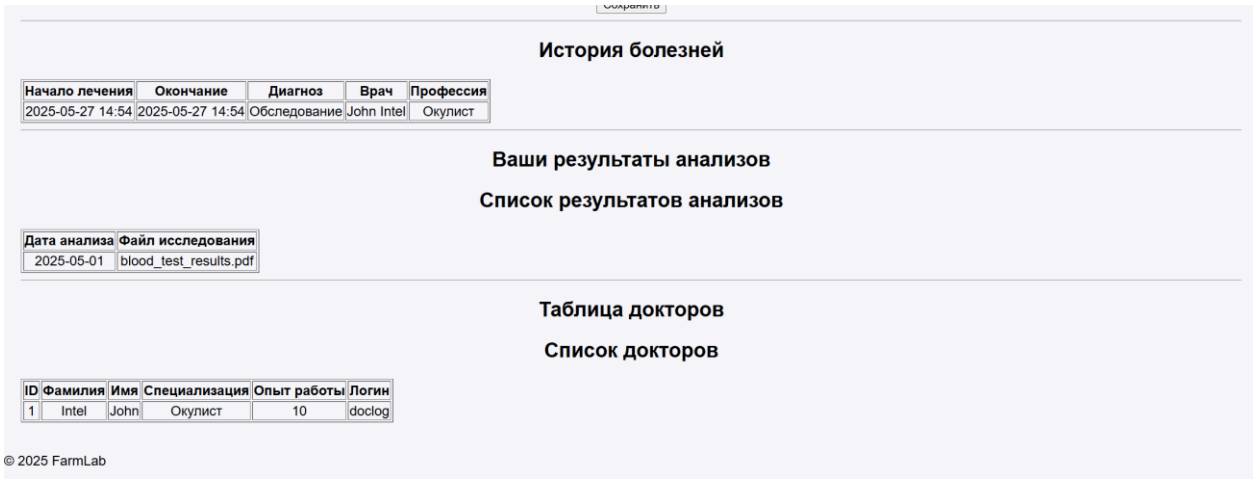


Рисунок 5 – личный кабинет клиента

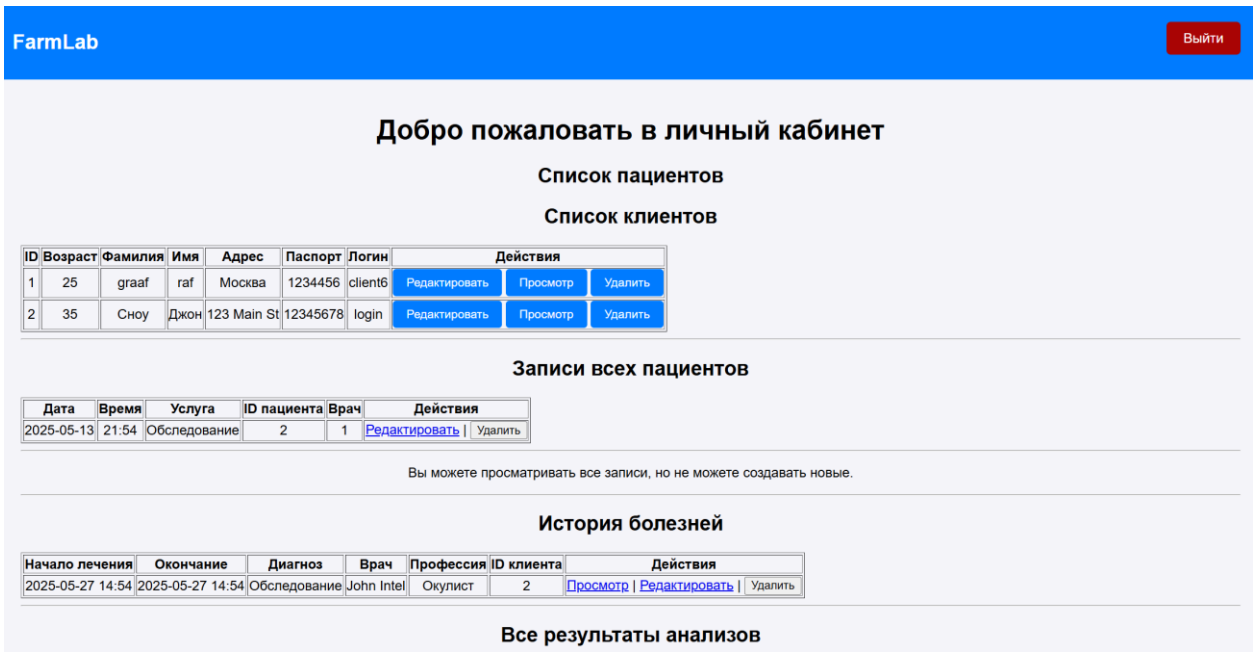


Рисунок 6 – личный кабинет администратора

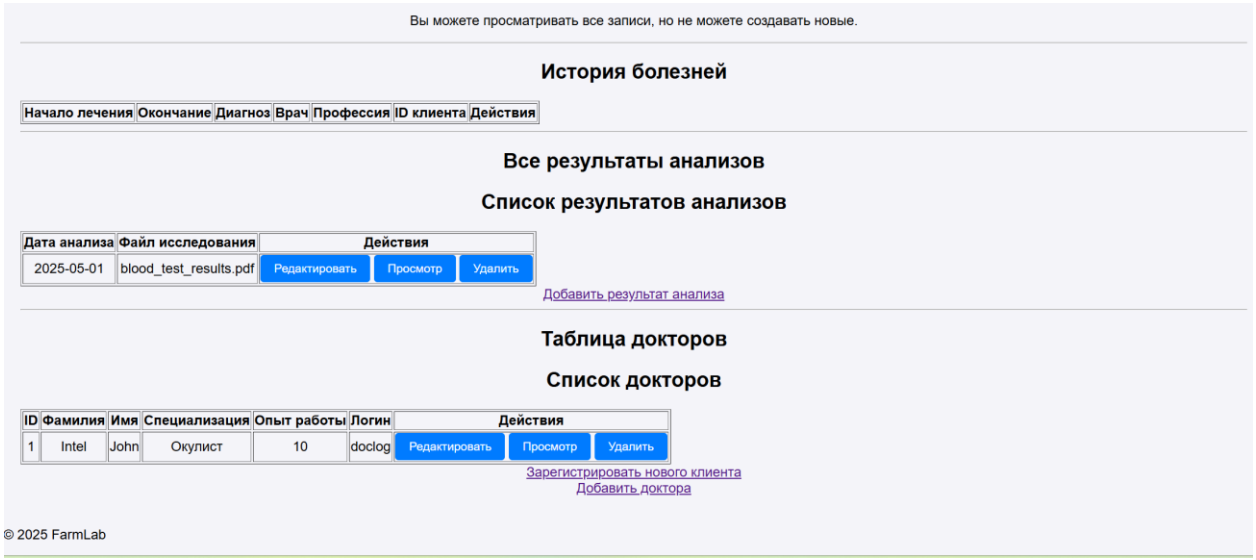


Рисунок 7 – личный кабинет администратора

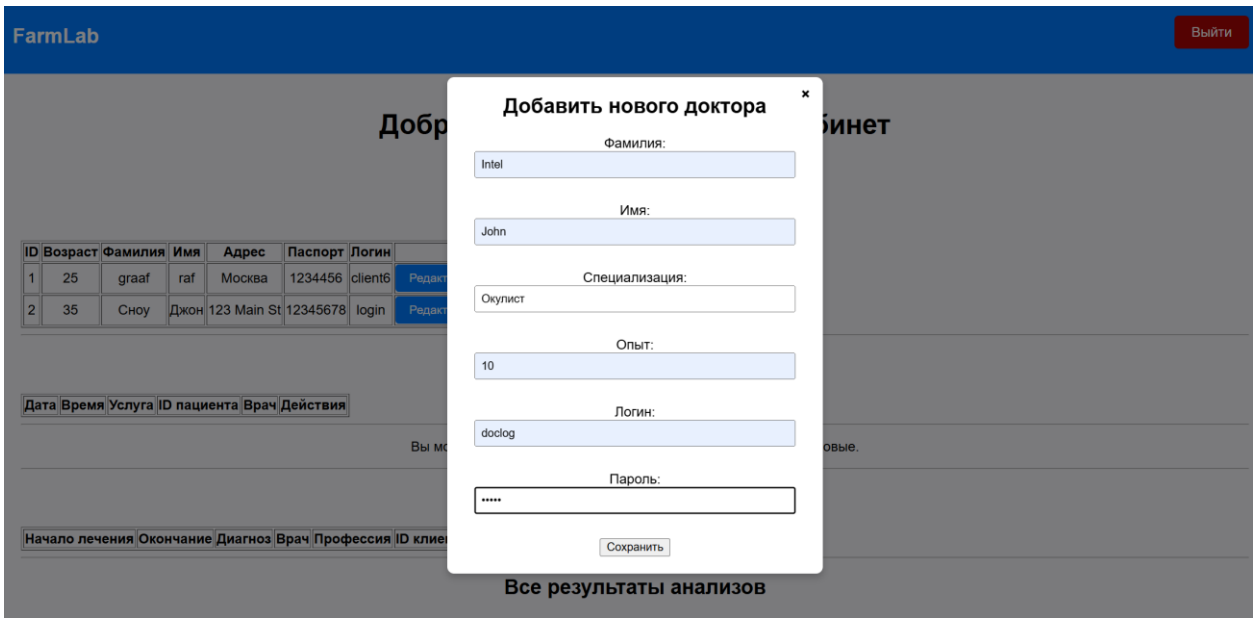


Рисунок 8 – регистрация доктора

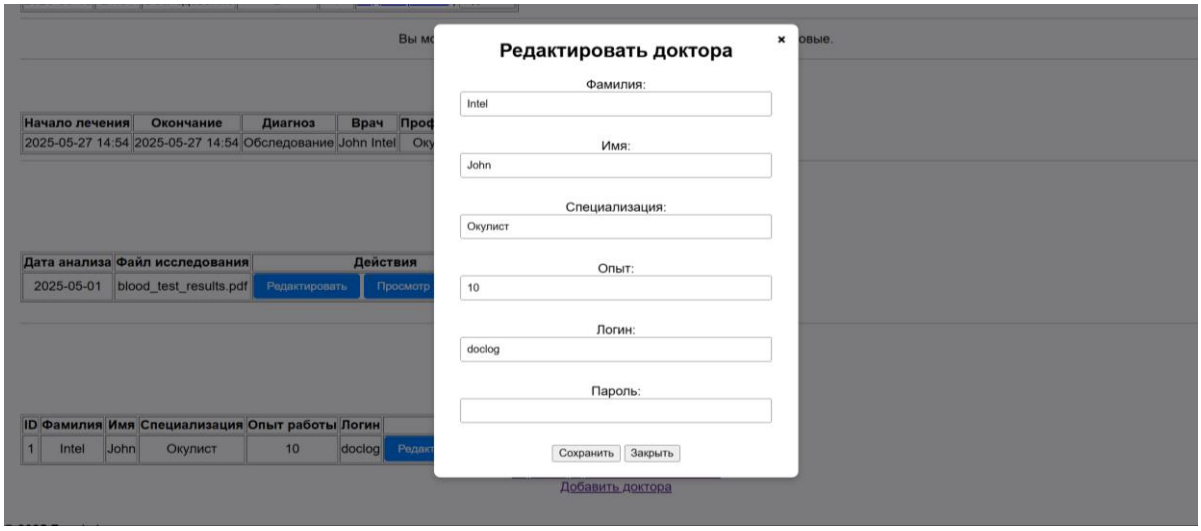


Рисунок 9 – редактировать профиль доктора

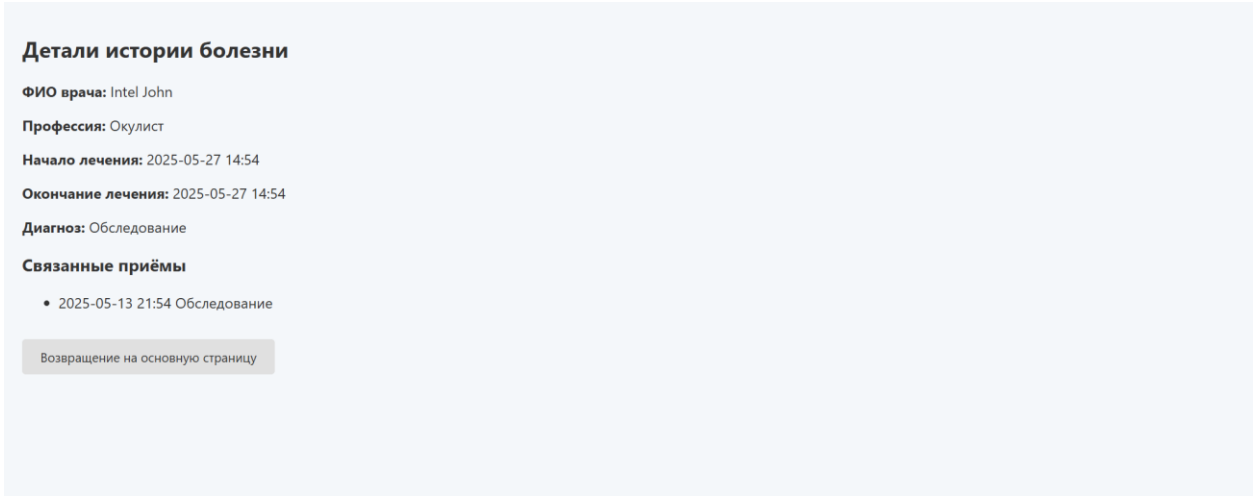


Рисунок 10 – просмотр истории болезни

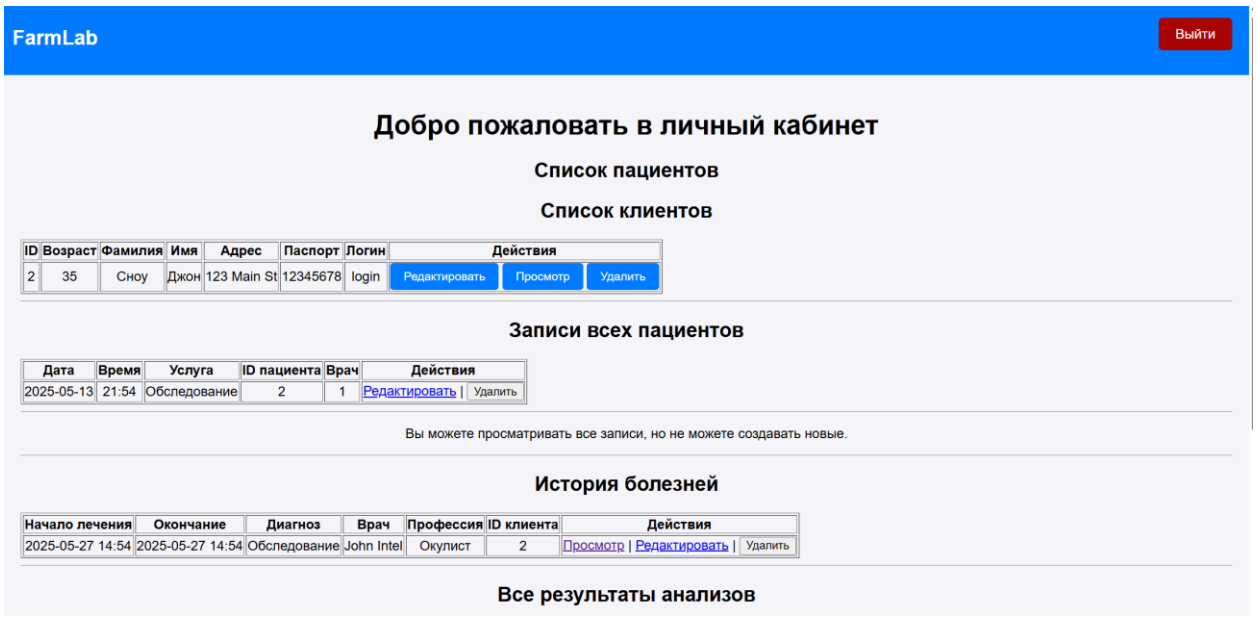


Рисунок 11 – личный кабинет доктора

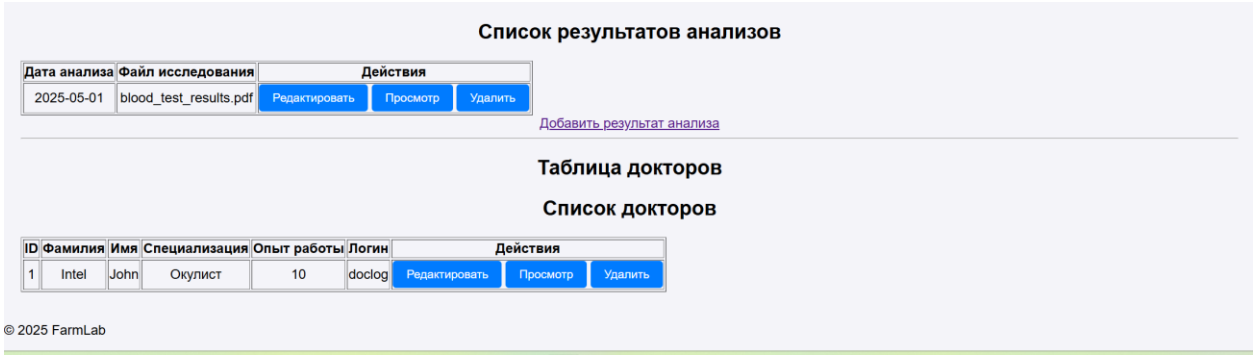


Рисунок 12 – личный кабинет доктора

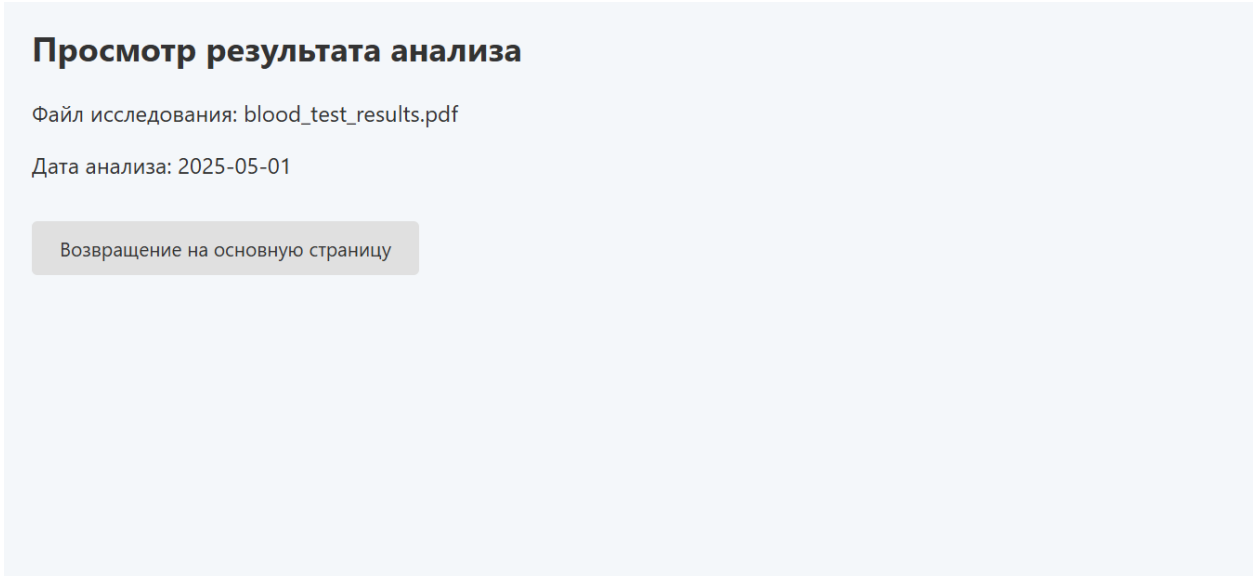


Рисунок 13 – просмотр результатов анализа

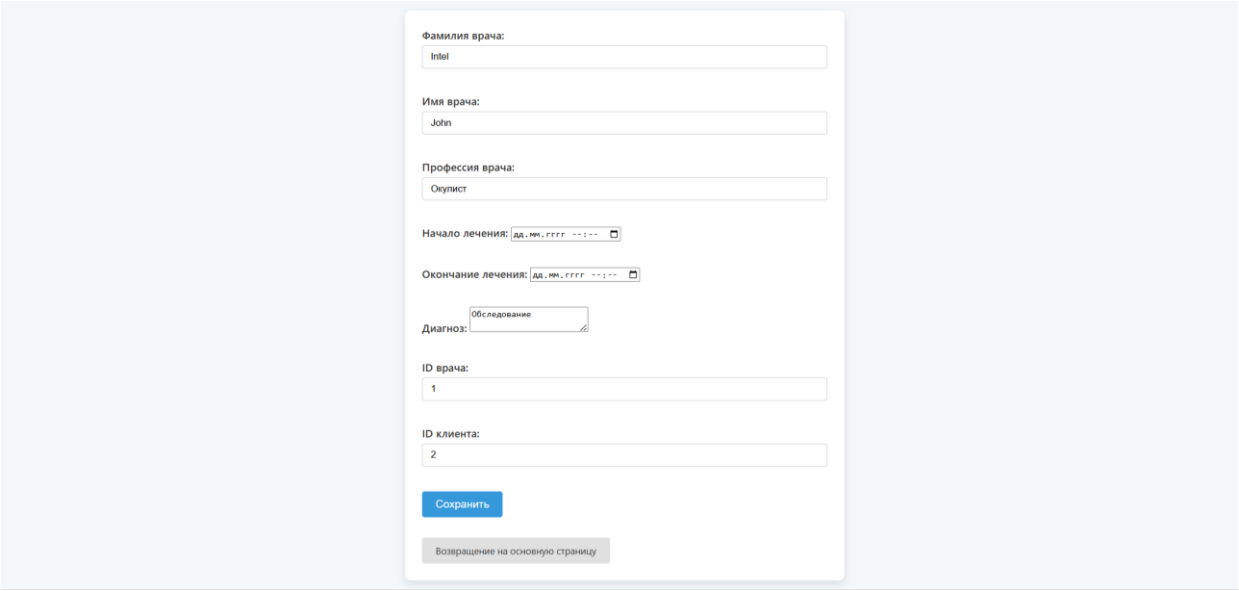


Рисунок 14 – редактирование истории болезни

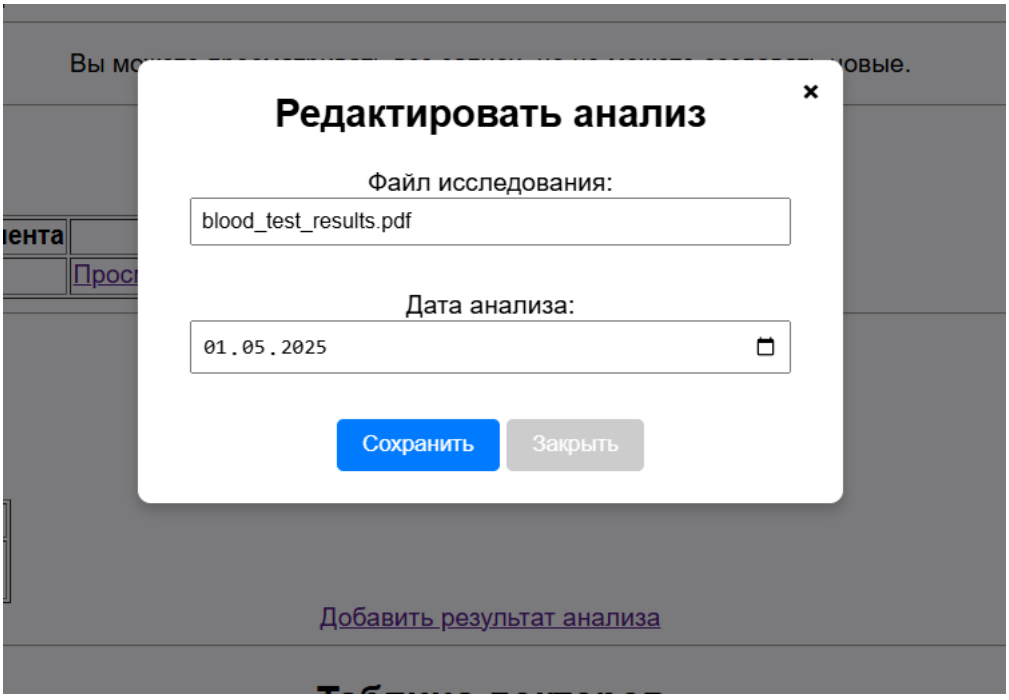


Рисунок 15 – редактировать результаты анализов

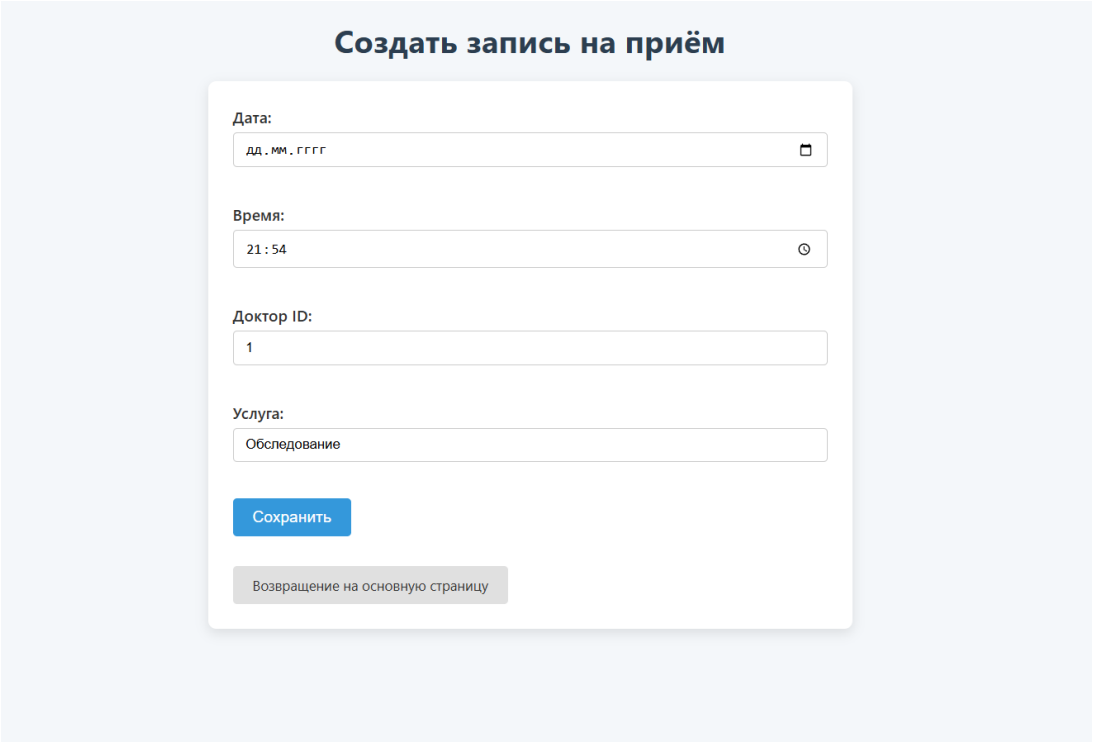


Рисунок 16 – создать запись на приём

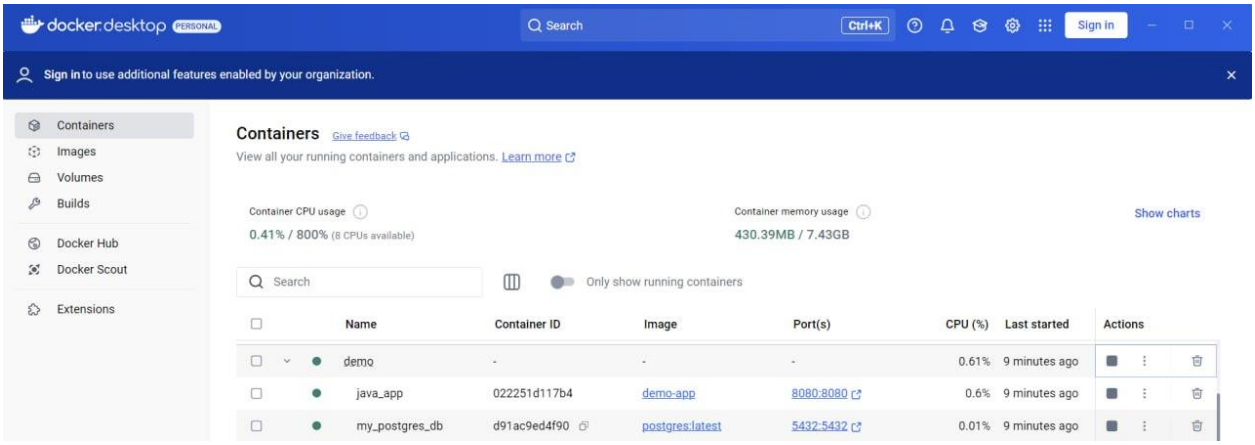


Рисунок 17 – докер

# Приложение А

## SecurityConfig

```
package com.example.demo.config;

import com.example.demo.security.JwtAuthenticationFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConf
iguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.List;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf(csrf -> csrf.disable())
                .csrf(csrf -> csrf.disable())
                .cors(cors -> cors.configurationSource(corsConfigurationSource()))

                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/auth/login/**").permitAll()
                        .requestMatchers("/auth/register/client").permitAll()
                        .requestMatchers("/auth/register/admin").permitAll()
                        .requestMatchers("/auth/register/doctor").hasRole("ADMIN")
                        .requestMatchers(HttpMethod.POST, "/auth/logout").authenticated()
                        .requestMatchers("/api/v1/clients/me").authenticated()
                        .requestMatchers(HttpMethod.GET,
"/api/v1/clients/{id}").hasAnyRole("CLIENT", "DOCTOR", "ADMIN")
```

```
                .requestMatchers(HttpMethod.GET, "/api/v1/analysis-
results/{id}").hasAnyRole("CLIENT", "DOCTOR", "ADMIN")
                .requestMatchers(HttpMethod.GET, "/api/v1/appointment-
records/{recordId}").hasAnyRole("CLIENT", "DOCTOR", "ADMIN")
                .requestMatchers(HttpMethod.GET, "/api/v1/disease-
history/{id}").hasAnyRole("CLIENT", "DOCTOR", "ADMIN")
                .requestMatchers(HttpMethod.GET,
"/api/v1/doctors/{id}").hasAnyRole("CLIENT", "DOCTOR", "ADMIN")
                .requestMatchers(HttpMethod.POST, "/api/v1/doctors/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.PUT, "/api/v1/doctors/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.DELETE,
"/api/v1/doctors/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.GET, "/api/v1/**").hasAnyRole("DOCTOR",
"ADMIN")
                .requestMatchers(HttpMethod.POST, "/api/v1/**").hasAnyRole("DOCTOR",
"ADMIN")
                .requestMatchers(HttpMethod.PUT, "/api/v1/**").hasAnyRole("DOCTOR",
"ADMIN")
                .requestMatchers(HttpMethod.DELETE, "/api/v1/**").hasAnyRole("DOCTOR",
"ADMIN")
                .requestMatchers(HttpMethod.GET, "/api/v1/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.POST, "/api/v1/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.PUT, "/api/v1/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.DELETE, "/api/v1/**").hasRole("ADMIN")
                .anyRequest().authenticated()
        )

        .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
        .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

    return http.build();
  }

  @Bean
  public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
    return authenticationConfiguration.getAuthenticationManager();
  }

  @Bean
  public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(List.of("http://localhost:5500")); // или фронтовый адрес
    configuration.setAllowCredentials(true);
    configuration.addAllowedHeader("*");
    configuration.addAllowedMethod("*");

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
```

```
    }
}
```

## ClientController

```
package com.example.demo.controllers;

import com.example.demo.dto.ChangePasswordRequest;
import com.example.demo.dto.ClientDTO;
import com.example.demo.entity.Client;
import com.example.demo.service.ClientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.security.core.Authentication;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/v1/clients")
public class ClientController {

    @Autowired
    private ClientService clientService;


    // Получение данных текущего авторизованного клиента
    @GetMapping("/me")
    public ResponseEntity<ClientDTO> getCurrentClient(Authentication authentication) {
        String username = authentication.getName();
        return clientService.findByLogin(username)
                .map(client -> ResponseEntity.ok(clientService.convertToDTO(client)))
                .orElse(ResponseEntity.status(HttpStatus.NOT_FOUND).build());
    }

    // CREATE
    @PostMapping
    public ResponseEntity<ClientDTO> createClient(@RequestBody ClientDTO clientDTO) {
        return
ResponseEntity.status(HttpStatus.CREATED).body(clientService.createClientFromDTO(clientD
TO));
    }

    // READ (все записи)
    @GetMapping
    public ResponseEntity<List<ClientDTO>> getAllClients() {
        return ResponseEntity.ok(clientService.getAllClientsAsDTO());
```

```
    }


    @GetMapping("/{id}")
    public ResponseEntity<ClientDTO> getClientById(@PathVariable Long id, Authentication
auth) {
        UserDetails user = (UserDetails) auth.getPrincipal();

        if (user.getAuthorities().contains(new SimpleGrantedAuthority("ROLE_CLIENT"))) {
            Optional<Client> current = clientService.findByLogin(user.getUsername());
            if (current.isEmpty() || !Long.valueOf(current.get().getId()).equals(id)) {
                return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
            }
        }

        return clientService.getClientByIdAsDTO(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    // UPDATE
    @PutMapping("/{id}")
    public ResponseEntity<ClientDTO> updateClient(@PathVariable Long id, @RequestBody
ClientDTO updatedClientDTO) {
        return ResponseEntity.ok(clientService.updateClientFromDTO(id, updatedClientDTO));
    }

    // DELETE
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteClient(@PathVariable Long id) {
        clientService.deleteClient(id);
        return ResponseEntity.noContent().build();
    }

}
```

## AuthController

```
package com.example.demo.controllers;

import jakarta.servlet.http.HttpServletResponse;
import org.springframework.http.ResponseCookie;
import org.springframework.ui.Model;
import com.example.demo.dto.*;
import com.example.demo.entity.Admin;
import com.example.demo.entity.Client;
import com.example.demo.entity.Doctor;
import com.example.demo.entity.Role;
import com.example.demo.repository.AdminRepository;
import com.example.demo.repository.ClientRepository;
import com.example.demo.repository.DoctorRepository;
import com.example.demo.security.JwtTokenService;
import com.example.demo.service.BlacklistService;
```

```
import com.example.demo.service.UserDetailsServiceImpl;
import io.jsonwebtoken.ExpiredJwtException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;
import com.example.demo.dto.ChangePasswordRequest;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;

import java.time.LocalDateTime;


@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired private AuthenticationManager authenticationManager;
    @Autowired private JwtTokenService jwtTokenService;
    @Autowired private UserDetailsServiceImpl userDetailsService;
    @Autowired private ClientRepository clientRepository;
    @Autowired private DoctorRepository doctorRepository;
    @Autowired private AdminRepository adminRepository;
    @Autowired private PasswordEncoder passwordEncoder;
    @Autowired private BlacklistService blacklistService;




    // 🔐 Авторизация (общая)
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest,
HttpServletResponse response) {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        loginRequest.getUsername(),
                        loginRequest.getPassword()
                )
        );

        UserDetails userDetails =
userDetailsService.loadUserByUsername(loginRequest.getUsername());
        String token = jwtTokenService.generateToken(userDetails);

        // Теперь создаём куку с токеном
        ResponseCookie cookie = ResponseCookie.from("jwt", token)
                .httpOnly(true)
```

```
                .path("/")
                .maxAge(24 * 60 * 60) // 1 день
                .sameSite("Strict")
                .build();
        response.setHeader("Set-Cookie", cookie.toString());

        return ResponseEntity.ok(new JwtResponse(token));
    }




    @PostMapping("/register/client")
    public ResponseEntity<?> registerClient(@RequestBody ClientDTO dto) {
        if (clientRepository.findByLogin(dto.getLogin()).isPresent()) {
            return ResponseEntity.badRequest().body("Client already exists");
        }

        Client client = new Client();
        client.setLogin(dto.getLogin());
        client.setPassword(passwordEncoder.encode(dto.getPassword()));
        client.setAge(dto.getAge());
        client.setGender(dto.getGender());
        client.setLastName(dto.getLastName());
        client.setFirstName(dto.getFirstName());
        client.setAddress(dto.getAddress());
        client.setPassport(dto.getPassport());
        client.setRole(Role.CLIENT);

        clientRepository.save(client);

        return ResponseEntity.ok(new
JwtResponse(jwtTokenService.generateTokenFromLogin(client.getLogin(),
"ROLE_CLIENT")));
    }


    @PostMapping("/register/doctor")
    public ResponseEntity<?> registerDoctor(@RequestBody DoctorDTO dto) {
        if (doctorRepository.findByLogin(dto.getLogin()).isPresent()) {
            return ResponseEntity.badRequest().body("Doctor already exists");
        }

        Doctor doctor = new Doctor();
        doctor.setLogin(dto.getLogin());
        doctor.setPassword(passwordEncoder.encode(dto.getPassword()));
        doctor.setLastName(dto.getLastName());
        doctor.setFirstName(dto.getFirstName());
        doctor.setSpecialization(dto.getSpecialization());
        doctor.setExperience(dto.getExperience());
        doctor.setRole(Role.DOCTOR);
```

```
        doctorRepository.save(doctor);

        return ResponseEntity.ok(new
JwtResponse(jwtTokenService.generateTokenFromLogin(doctor.getLogin(),
"ROLE_DOCTOR")));
    }


    @PostMapping("/register/admin")
    public ResponseEntity<?> registerAdmin(@RequestBody Admin dto) {
        if (adminRepository.findByLogin(dto.getLogin()).isPresent()) {
            return ResponseEntity.badRequest().body("Admin already exists");
        }

        Admin admin = new Admin();
        admin.setLogin(dto.getLogin());
        admin.setPassword(passwordEncoder.encode(dto.getPassword()));

        adminRepository.save(admin);

        return ResponseEntity.ok(new
JwtResponse(jwtTokenService.generateTokenFromLogin(admin.getLogin(),
"ROLE_ADMIN")));
    }

    @PostMapping("/change-password")
    public ResponseEntity<?> changePassword(@RequestBody ChangePasswordRequest request,
Authentication authentication) {
        String username = authentication.getName();
        String role = authentication.getAuthorities().stream()
                .findFirst()
                .map(grantedAuthority -> grantedAuthority.getAuthority()) // например:
ROLE_CLIENT
                .orElseThrow(() -> new RuntimeException("Роль не найдена"));

        try {
            userDetailsService.changePassword(username, role, request.getOldPassword(),
request.getNewPassword());
            return ResponseEntity.ok("Пароль успешно изменен");
        } catch (IllegalArgumentException e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
        }
    }
    /*
    @PostMapping("/logout")
    public ResponseEntity<?> logout(@RequestHeader("Authorization") String authHeader) {
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String token = authHeader.substring(7);
            try {
                // Извлекаем дату истечения токена
                LocalDateTime expiryDate = jwtTokenService.extractExpiration(token);
```

```
            blacklistService.addToBlacklist(token, expiryDate);
            return ResponseEntity.ok("Logged out successfully");
        } catch (ExpiredJwtException e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Token already
expired");
        }
    }
    return ResponseEntity.badRequest().body("Invalid token");
  }

   */

  @PostMapping("/logout")
  public ResponseEntity<?> logout(
      @CookieValue(value = "jwt", required = false) String token,
      HttpServletResponse response
  ) {
    if (token != null) {
      try {
        // Извлекаем дату истечения токена
        LocalDateTime expiryDate = jwtTokenService.extractExpiration(token);
        blacklistService.addToBlacklist(token, expiryDate);

        // Удаляем cookie
        ResponseCookie deleteCookie = ResponseCookie.from("jwt", "")
            .httpOnly(true)
            .path("/")
            .maxAge(0) // Удаляем куку
            .sameSite("Strict")
            .build();
        response.setHeader("Set-Cookie", deleteCookie.toString());

        return ResponseEntity.ok("Logged out successfully");
      } catch (ExpiredJwtException e) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Token already
expired");
      }
    }
    return ResponseEntity.badRequest().body("No token found in cookies");
  }

}
```

## ClientDTO

```
package com.example.demo.entity;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import java.util.*;

@Entity
@Table(name = "clients")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Client implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "age", nullable = false)
    private int age;

    @Column(name = "gender", nullable = false)
    private String gender;

    @Column(name = "last_name", nullable = false, length = 50)
    private String lastName;

    @Column(name = "first_name", nullable = false, length = 50)
    private String firstName;

    @Column(name = "address")
    private String address;

    @Column(name = "passport", nullable = false)
    private String passport;

    @Column(name = "login", nullable = false, unique = true)
    private String login;

    @Column(name = "password", nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(name = "role", nullable = false)
    private Role role;

/*
    // Связь 1:1 с таблицей "История болезни"
    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "disease_history_id", referencedColumnName = "record_id")
    private DiseaseHistory diseaseHistory;
*/

    // Связь 1:N с таблицей "Результаты анализов"
    @OneToMany(mappedBy = "client", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<AnalysisResult> analysisResults = new ArrayList<>();
```

```
    // Связь 1:N с таблицей "Список записей"
    @OneToMany(mappedBy = "client", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<AppointmentRecord> appointmentRecords = new ArrayList<>();


    public Client() {}

    public Client(int age, String gender, String lastName, String firstName, String address, String
passport) {
        this.age = age;
        this.gender = gender;
        this.lastName = lastName;
        this.firstName = firstName;
        this.address = address;
        this.passport = passport;
    }

    public Client(int age, String gender, String lastName, String firstName, String passport) {
        this.age = age;
        this.gender = gender;
        this.lastName = lastName;
        this.firstName = firstName;
        this.passport = passport;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getLastName() {
        return lastName;
```

```
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getFirstName() {
   return firstName;
}

public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public String getAddress() {
   return address;
}

public void setAddress(String address) {
   this.address = address;
}

public String getPassport() {
   return passport;
}

public void setPassport(String passport) {
   this.passport = passport;
}

public String getLogin() {
   return login;
}

public void setLogin(String login) {
   this.login = login;
}

public Role getRole() {
   return role;
}

public void setRole(Role role) {
   this.role = role;
}

@Override
public String toString() {
   return "Client{" +
         "id=" + id +
         ", age=" + age +
         ", gender='" + gender + '\'' +
```

```
                ", lastName='" + lastName + '\'' +
                ", firstName='" + firstName + '\'' +
                ", address='" + address + '\'' +
                ", passport='" + passport + '\'' +
                '}';
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singletonList(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getUsername() {
        return login;
    }

    @Override
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true; // Учетная запись всегда активна
    }

    @Override
    public boolean isAccountNonLocked() {
        return true; // Учетная запись не заблокирована
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true; // Срок действия пароля не истек
    }

    @Override
    public boolean isEnabled() {
        return true; // Учетная запись включена
    }

}
```

## Client

```
package com.example.demo.entity;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
```

```
import jakarta.persistence.*;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;


import java.util.*;

@Entity
@Table(name = "clients")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Client implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "age", nullable = false)
    private int age;

    @Column(name = "gender", nullable = false)
    private String gender;

    @Column(name = "last_name", nullable = false, length = 50)
    private String lastName;

    @Column(name = "first_name", nullable = false, length = 50)
    private String firstName;

    @Column(name = "address")
    private String address;

    @Column(name = "passport", nullable = false)
    private String passport;

    @Column(name = "login", nullable = false, unique = true)
    private String login;

    @Column(name = "password", nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(name = "role", nullable = false)
    private Role role;

/*
    // Связь 1:1 с таблицей "История болезни"
    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "disease_history_id", referencedColumnName = "record_id")
    private DiseaseHistory diseaseHistory;
```

```
*/

    // Связь 1:N с таблицей "Результаты анализов"
    @OneToMany(mappedBy = "client", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<AnalysisResult> analysisResults = new ArrayList<>();

    // Связь 1:N с таблицей "Список записей"
    @OneToMany(mappedBy = "client", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<AppointmentRecord> appointmentRecords = new ArrayList<>();


    public Client() {}

    public Client(int age, String gender, String lastName, String firstName, String address, String
passport) {
        this.age = age;
        this.gender = gender;
        this.lastName = lastName;
        this.firstName = firstName;
        this.address = address;
        this.passport = passport;
    }

    public Client(int age, String gender, String lastName, String firstName, String passport) {
        this.age = age;
        this.gender = gender;
        this.lastName = lastName;
        this.firstName = firstName;
        this.passport = passport;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
```

```
    this.gender = gender;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getPassport() {
    return passport;
}

public void setPassport(String passport) {
    this.passport = passport;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public Role getRole() {
    return role;
}

public void setRole(Role role) {
    this.role = role;
}

@Override
```

```
    public String toString() {
        return "Client{" +
                "id=" + id +
                ", age=" + age +
                ", gender='" + gender + '\'' +
                ", lastName='" + lastName + '\'' +
                ", firstName='" + firstName + '\'' +
                ", address='" + address + '\'' +
                ", passport='" + passport + '\'' +
                '}';
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singletonList(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getUsername() {
        return login;
    }

    @Override
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true; // Учетная запись всегда активна
    }

    @Override
    public boolean isAccountNonLocked() {
        return true; // Учетная запись не заблокирована
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true; // Срок действия пароля не истек
    }

    @Override
    public boolean isEnabled() {
        return true; // Учетная запись включена
    }

}
```

## ClientRepository

```
package com.example.demo.repository;

import com.example.demo.entity.Client;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface ClientRepository extends JpaRepository<Client, Long> {
    Optional<Client> findByLogin(String login);
}
```

## JwtTokenService

```
package com.example.demo.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;

import java.time.LocalDateTime;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Service
public class JwtTokenService {

    @Value("${jwt.secret}")
    private String secretKey; // Секретный ключ берется из application.properties

    private final long expirationTime = 86400000; // Время жизни токена (например, 24 часа)

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("roles", userDetails.getAuthorities().stream()
                .map(grantedAuthority -> grantedAuthority.getAuthority())
                .toList());
        return createToken(claims, userDetails.getUsername());
    }

    private String createToken(Map<String, Object> claims, String subject) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
```

```
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expirationTime))
            .signWith(SignatureAlgorithm.HS512, secretKey) // Убираем .getBytes()
            .compact();
    }

    public Boolean validateToken(String token) {
        try {
            // Парсим токен и проверяем подпись
            Jwts.parser()
                    .setSigningKey(secretKey)
                    .build()
                    .parseClaimsJws(token);

            // Проверяем, истек ли срок действия токена
            return !isTokenExpired(token);
        } catch (Exception e) {
            // Если возникла ошибка (например, неверная подпись или истекший токен),
возвращаем false
            return false;
        }
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    private <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        try {
            return Jwts.parser()
                    .setSigningKey(secretKey)// Убираем .getBytes() и .build()
                    .build()
                    .parseClaimsJws(token)
                    .getBody();
        } catch (Exception e) {
            throw new RuntimeException("Invalid or expired token", e);
        }
    }

    private Boolean isTokenExpired(String token) {
        return extractClaim(token, Claims::getExpiration).before(new Date());
    }

    public String generateTokenFromLogin(String login, String role) {
        // Создаем временного пользователя с нужной ролью
        User userDetails = new User(
                login,
```

```
                "", // Пароль не нужен для генерации токена
                Collections.singletonList(new SimpleGrantedAuthority(role))
        );
        return generateToken(userDetails);
    }

    // это нужно для фиксации времени, когда токен удаляется из блэклиста автоматически
    public LocalDateTime extractExpiration(String token) {
        Date expirationDate = extractClaim(token, Claims::getExpiration);
        return expirationDate.toInstant()
                        .atZone(java.time.ZoneId.systemDefault())
                        .toLocalDateTime();
    }
}
```

## ClientService

```
package com.example.demo.service;


import com.example.demo.dto.ClientDTO;
import com.example.demo.entity.Client;
import com.example.demo.entity.Role;
import com.example.demo.repository.ClientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
@Service
public class ClientService {

    @Autowired
    private ClientRepository clientRepository;

    @Autowired
    private PasswordEncoder passwordEncoder; // Добавляем PasswordEncoder

    public List<Client> getAllClients() {
        return clientRepository.findAll();
    }

    public Optional<Client> getClientById(Long id) {
        return clientRepository.findById(id);
    }

    public void deleteClient(Long id) {
        clientRepository.deleteById(id);
    }
```

```
    // получение списка столбцов
    public List<String> getClientColumns() {
        return Arrays.stream(Client.class.getDeclaredFields())
                .map(Field::getName)
                .collect(Collectors.toList());
    }

    // создание клиента "в одну строку"

    public Client createClient(int age, String gender, String lastName, String firstName, String
address, String passport) {
        Client client = new Client();
        client.setAge(age);
        client.setGender(gender);
        client.setLastName(lastName);
        client.setFirstName(firstName);
        client.setAddress(address);
        client.setPassport(passport);
        return clientRepository.save(client);
    }

    public Client createClient(Client client) {
        return clientRepository.save(client);
    }

    public Client updateClient(Long id, Client updatedClient) {
        Client existingClient = clientRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Client not found with ID: " + id));

        // Обновляем поля существующего клиента
        existingClient.setAge(updatedClient.getAge());
        existingClient.setGender(updatedClient.getGender());
        existingClient.setLastName(updatedClient.getLastName());
        existingClient.setFirstName(updatedClient.getFirstName());
        existingClient.setAddress(updatedClient.getAddress());
        existingClient.setPassport(updatedClient.getPassport());

        return clientRepository.save(existingClient);
    }

    // Преобразование Entity -> DTO
    public ClientDTO convertToDTO(Client client) {
        return new ClientDTO(client);
    }

    // Преобразование DTO -> Entity
    public Client convertToEntity(ClientDTO dto) {
        Client client = new Client();
        //client.setId(Math.toIntExact(dto.getId()));
        if (dto.getId() != null) {
            client.setId(Math.toIntExact(dto.getId())); // Устанавливаем id только если оно не null
```

```
    }
    client.setAge(dto.getAge());
    client.setGender(dto.getGender());
    client.setLastName(dto.getLastName());
    client.setFirstName(dto.getFirstName());
    client.setAddress(dto.getAddress());
    client.setPassport(dto.getPassport());

    client.setLogin(dto.getLogin());
    client.setPassword(passwordEncoder.encode(dto.getPassword())); // Хэшируем пароль
    client.setRole(dto.getRole()); // Преобразуем строку в Enum
    return client;
}

// Получение всех клиентов в виде DTO
public List<ClientDTO> getAllClientsAsDTO() {
    return clientRepository.findAll().stream()
            .map(this::convertToDTO)
            .collect(Collectors.toList());
}

// Получение клиента по ID в виде DTO
public Optional<ClientDTO> getClientByIdAsDTO(Long id) {
    return clientRepository.findById(id).map(this::convertToDTO);
}

// Создание клиента из DTO
public ClientDTO createClientFromDTO(ClientDTO dto) {
    Client client = convertToEntity(dto);
    client.setRole(Role.CLIENT); // Автоматическая установка роли
    Client savedClient = clientRepository.save(client);
    return convertToDTO(savedClient);
}

// Обновление клиента из DTO
public ClientDTO updateClientFromDTO(Long id, ClientDTO updatedDto) {
    Client existingClient = clientRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Client not found with ID: " + id));

    // Обновляем поля существующего клиента
    existingClient.setAge(updatedDto.getAge());
    existingClient.setGender(updatedDto.getGender());
    existingClient.setLastName(updatedDto.getLastName());
    existingClient.setFirstName(updatedDto.getFirstName());
    existingClient.setAddress(updatedDto.getAddress());
    existingClient.setPassport(updatedDto.getPassport());

    Client savedClient = clientRepository.save(existingClient);
    return convertToDTO(savedClient);
}

public Optional<Client> findByLogin(String login) {
```

```
        return clientRepository.findByLogin(login);
    }

}
```

## DemoApplication

```
package com.example.demo;

import com.example.demo.entity.Client;
import com.example.demo.entity.Doctor;
import com.example.demo.service.ClientService;
import com.example.demo.service.DoctorService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

import java.util.List;
import java.util.Optional;

@SpringBootApplication
@EnableScheduling // Включение поддержки планирования задач
public class DemoApplication implements CommandLineRunner {

    @Autowired
    private ClientService clientService;
    @Autowired
    private DoctorService doctorService;

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {

        System.out.println("Столбцы таблицы clients: " + clientService.getClientColumns());

        List<Client> allClients = clientService.getAllClients();
        System.out.println("Список всех клиентов: " + allClients);

    }
}
```

## Dashboard

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.w3.org/1999/xhtml">
<th:block th:replace="~{layout/base :: layout(~{::content})}">
    <th:block th:fragment="content">
```

```
<h1>Добро пожаловать в личный кабинет</h1>
<div class="logout-container">
  <button id="logoutBtn">Выйти</button>
  <script src="/js/logout.js"></script>
</div>


<div th:if="${успешно}" class="alert success">
  <p th:text="${успешно}">Профиль обновлен</p>
</div>

<div th:if="${ошибка}" class="alert error">
  <p th:text="${ошибка}">Ошибка при обновлении профиля</p>
</div>

<h2 th:unless="${isAdminOrDoctor}">Ваш профиль</h2>
<form th:unless="${isAdminOrDoctor}" th:action="@{/client/dashboard}" method="post"
th:object="${client}" >
  <label>Имя:
    <input type="text" th:field="*{firstName}" required/>
  </label><br/>

  <label>Фамилия:
    <input type="text" th:field="*{lastName}" required/>
  </label><br/>

  <label>Возраст:
    <input type="number" th:field="*{age}" required/>
  </label><br/>

  <label>Пол:
    <input type="text" th:field="*{gender}" required/>
  </label><br/>

  <label>Адрес:
    <input type="text" th:field="*{address}" required/>
  </label><br/>

  <label>Паспорт:
    <input type="text" th:field="*{passport}" required/>
  </label><br/>

  <button type="submit">Обновить профиль</button>
</form>

<!-- Список пациентов — только для DOCTOR / ADMIN -->
<!--
<div th:if="${isAdminOrDoctor}">
  <h2>Список пациентов</h2>
  <table border="1">
    <thead>
      <tr>
```

```
                <th>ID</th>
                <th>Логин</th>
                <th>Имя</th>
                <th>Фамилия</th>
                <th>Возраст</th>
                <th>Действия</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="c : ${allClients}">
                <td th:text="${c.id}">ID</td>
                <td th:text="${c.login}">login</td>
                <td th:text="${c.firstName}">Имя</td>
                <td th:text="${c.lastName}">Фамилия</td>
                <td th:text="${c.age}">30</td>
                <td>
                    <a th:href="@{'/web/clients/' + ${c.id}}">Просмотр</a> |
                    <a th:href="@{'/web/clients/' + ${c.id} + '/edit'}">Редактировать</a> |
                    <form th:action="@{'/web/clients/' + ${c.id} + '/delete'}" method="post"
style="display:inline;">
                        <button type="submit">Удалить</button>
                    </form>
                </td>
            </tr>
        </tbody>
    </table>
</div>
-->


<!-- Список пациентов -->
<div th:if="${isAdminOrDoctor}">
    <h2 th:if="${role == 'ROLE_DOCTOR'}">Ваши пациенты</h2>
    <h2 th:if="${role == 'ROLE_ADMIN'}">Все клиенты</h2>
    <h2>Список пациентов</h2>
    <th:block th:replace="~{clients/list :: content}"></th:block>
</div>

<!-- Модальные окна -->
<div th:if="${isAdminOrDoctor}">
    <!-- <th:block th:insert="~{clients/edit :: content}"></th:block> -->
    <th:block th:insert="~{clients/view :: content(${client})}"></th:block>

</div>

<!-- Модальное окно редактирования клиента -->
<div id="editClientModal" class="modal">
    <div class="modal-content">
        <span class="close-btn" onclick="closeModal('editClientModal')">&times;</span>
        <h2>Редактировать данные клиента</h2>
        <form id="editClientForm" method="post">
            <input type="hidden" name="id" id="client_id"/>
```

```
            <label>Имя:
              <input type="text" id="client_firstName" name="firstName" required/>
            </label>
            <label>Фамилия:
              <input type="text" id="client_lastName" name="lastName" required/>
            </label>
            <label>Логин:
              <input type="text" id="client_login" name="login" required/>
            </label>
            <label>Пароль:
              <input type="password" id="client_password" name="password"/>
            </label>
            <label>Возраст:
              <input type="number" id="client_age" name="age" required/>
            </label>
            <label>Пол:
              <input type="text" id="client_gender" name="gender" required/>
            </label>
            <label>Адрес:
              <input type="text" id="client_address" name="address" required/>
            </label>
            <label>Паспорт:
              <input type="text" id="client_passport" name="passport" required/>
            </label><br/>
            <button type="submit" class="button">Сохранить изменения</button>
            <button type="button" class="button button-cancel"
onclick="closeModal('editClientModal')">Отмена</button>
          </form>
        </div>
      </div>

      <hr/>

      <!-- Записи на приём -->
      <h2 th:unless="${isAdminOrDoctor}">Ваши записи на приём</h2>
      <h2 th:if="${isAdminOrDoctor}">Записи всех пациентов</h2>

      <table border="1">
        <thead>
          <tr>
            <th>Дата</th>
            <th>Время</th>
            <th>Услуга</th>
            <th th:if="${isAdminOrDoctor}">ID пациента</th>
            <th>Врач</th>
            <th th:if="${isAdminOrDoctor}">Действия</th>
          </tr>
        </thead>
        <tbody>
          <tr th:each="record : ${records}">
            <td th:text="${record.appointmentDate}">2025-05-01</td>
            <td th:text="${record.appointmentTime}">10:00</td>
```

```
            <td th:text="${record.serviceName}">Терапия</td>

            <!-- Только доктор и админ видят ID пациента -->
            <td th:if="${isAdminOrDoctor}" th:text="${record.clientId}">1</td>

            <!-- Для клиента показываем врача
            <td th:unless="${isAdminOrDoctor}" th:text="${record.doctor.firstName + ' ' +
record.doctor.lastName}">
              Иванов И.И.
            </td>  -->
            <td th:text="${record.doctorId}">ID врача</td>


            <!-- Доступные действия -->
            <td th:if="${isAdminOrDoctor}">
              <a th:href="@{'/web/appointments/' + ${record.recordId} +
'/edit'}">Редактировать</a> |
              <form th:action="@{'/web/appointments/' + ${record.recordId} + '/delete'}"
method="post" style="display:inline;">
                <button type="submit">Удалить</button>
              </form>
            </td>
          </tr>
        </tbody>
      </table>

      <hr/>

      <!-- Кнопка добавления записи -->
      <h2 th:unless="${isAdminOrDoctor}">Добавить новую запись</h2>
      <form th:unless="${isAdminOrDoctor}" action="/web/appointments" method="post">
        <label>Дата:
          <input type="date" name="appointmentDate" required />
        </label><br/>
        <label>Время:
          <input type="time" name="appointmentTime" required />
        </label><br/>
        <label>Услуга:
          <input type="text" name="serviceName" required />
        </label><br/>
        <label>ID врача:
          <input type="number" name="doctorId" required />
        </label><br/>
        <button type="submit">Сохранить</button>
      </form>

      <!-- Сообщение для доктора / админа -->
      <div th:if="${isAdminOrDoctor}">
        <p>Вы можете просматривать все записи, но не можете создавать новые.</p>
      </div>

      <hr/>
```

```
<!-- История болезней -->
<h2>История болезней</h2>
<table border="1">
  <thead>
    <tr>
      <th>Начало лечения</th>
      <th>Окончание</th>
      <th>Диагноз</th>
      <th>Врач</th>
      <th>Профессия</th>
      <th th:if="${isAdminOrDoctor}">ID клиента</th>
      <th th:if="${isAdminOrDoctor}">Действия</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="history : ${diseaseHistories}">
      <td th:text="${#temporals.format(history.startDate, 'yyyy-MM-dd
HH:mm')}">2025-05-01 12:00</td>
      <td th:text="${#temporals.format(history.endDate, 'yyyy-MM-dd
HH:mm')}">2025-05-10 13:00</td>
      <td th:text="${history.disease}">Грипп</td>
      <td th:text="${history.firstNameDoctor + ' ' + history.lastNameDoctor}">Иванов
И.И.</td>
      <td th:text="${history.profession}">Терапевт</td>

      <!-- Только доктор и админ видят ID клиента -->
      <td th:if="${isAdminOrDoctor}" th:text="${history.clientId}">1</td>

      <!-- Действия — только для доктора / админа -->
      <td th:if="${isAdminOrDoctor}">
        <a th:href="@{'/web/disease-history/' + ${history.recordId}}">Просмотр</a> |
        <a th:href="@{'/web/disease-history/' + ${history.recordId} +
'/edit'}">Редактировать</a> |
        <form th:action="@{'/web/disease-history/' + ${history.recordId} + '/delete'}"
method="post" style="display:inline;">
          <button type="submit">Удалить</button>
        </form>
      </td>
    </tr>
  </tbody>
</table>

<hr/>

<!-- Результаты анализов -->
<h2 th:unless="${isAdminOrDoctor}">Ваши результаты анализов</h2>
<h2 th:if="${isAdminOrDoctor}">Все результаты анализов</h2>
<th:block th:replace="~{analysis-results/list :: content}"></th:block>


<!-- <a th:if="${isAdminOrDoctor}" href="@{'web/analysis-results/new'}"
```

```
onclick="openAddAnalysisModal()">Добавить результат анализа</a> -->
        <a href="#" sec:authorize="hasAnyRole('ADMIN', 'DOCTOR')"
onclick="openAddAnalysisModal()">Добавить результат анализа</a>


        <hr/>
        <!-- Модальное окно добавления анализа -->
        <div id="addAnalysisModal" class="modal">
            <div class="modal-content">
                <span class="close-btn" onclick="closeModal('addAnalysisModal')">&times;</span>
                <h2>Добавить результат анализа</h2>
                <form id="addAnalysisForm" method="post" action="/web/analysis-results">
                    <label th:if="${isAdminOrDoctor}">
                        Клиент:
                        <select id="analysisClientId" name="clientId" required>
                            <option value="">Выберите клиента</option>
                            <option th:each="client : ${allClients}"
                                    th:value="${client.id}"
                                    th:text="${client.firstName + ' ' + client.lastName + ' (' + client.login +
')'}">
                            </option>
                        </select>
                    </label>
                    <label>Файл исследования:
                        <input type="text" id="researchFile" name="researchFile" required/>
                    </label><br/>
                    <label>Дата анализа:
                        <input type="date" id="analysisDate" name="analysisDate" required/>
                    </label><br/>
                    <button type="submit" class="button">Сохранить</button>
                    <button type="button" class="button button-cancel"
onclick="closeModal('addAnalysisModal')">Закрыть</button>
                </form>
            </div>
        </div>
        <!-- Модальное окно редактирования анализа -->
        <div id="editAnalysisModal" class="modal">
            <div class="modal-content">
                <span class="close-btn" onclick="closeModal('editAnalysisModal')">&times;</span>
                <h2>Редактировать анализ</h2>
                <form id="editAnalysisForm" method="post">
                    <input type="hidden" name="recordId" id="editRecordId"/>

                    <label>Файл исследования:
                        <input type="text" name="researchFile" id="editResearchFile" required/>
                    </label><br/>

                    <label>Дата анализа:
                        <input type="date" name="analysisDate" id="editAnalysisDate" required/>
                    </label><br/>

                    <button type="submit" class="button">Сохранить</button>
```

```
                <button type="button" class="button button-cancel"
onclick="closeModal('editAnalysisModal')">Закрыть</button>
            </form>
        </div>
    </div>

    <!-- Таблица докторов -->
    <h2>Таблица докторов</h2>
    <th:block th:replace="~{doctor/list :: content}"></th:block>

    <!-- Модальное окно добавления доктора -->
    <div id="addDoctorModal" class="modal" th:if="${isAdminOrDoctor}">
        <div class="modal-content">
            <span class="close-btn" onclick="closeModal('addDoctorModal')">&times;</span>
            <h2>Добавить нового доктора</h2>
            <form id="addDoctorForm" method="post" action="/web/doctors">
                <label>Фамилия: <input type="text" name="lastName" required/></label><br>
                <label>Имя: <input type="text" name="firstName" required/></label><br>
                <label>Специализация: <input type="text" name="specialization"
required/></label><br>
                <label>Опыт: <input type="text" name="experience" required/></label><br>
                <label>Логин: <input type="text" name="login" required/></label><br>
                <label>Пароль: <input type="password" name="password" required/></label><br>
                <button type="submit">Сохранить</button>
            </form>
        </div>
    </div>

    <!-- Модальное окно редактирования доктора -->
    <div id="editDoctorModal" class="modal">
        <div class="modal-content">
            <span class="close-btn" onclick="closeModal('editDoctorModal')">&times;</span>
            <h2>Редактировать доктора</h2>
            <form id="editDoctorForm" method="post">
                <input type="hidden" name="id" id="editDoctorId" />
                <label>Фамилия: <input type="text" name="lastName" id="editLastName"
/></label><br>
                <label>Имя: <input type="text" name="firstName" id="editFirstName"
/></label><br>
                <label>Специализация: <input type="text" name="specialization"
id="editSpecialization" /></label><br>
                <label>Опыт: <input type="text" name="experience" id="editExperience"
/></label><br>
                <label>Логин: <input type="text" name="login" id="editLogin" /></label><br>
                <label>Пароль: <input type="password" name="password" id="editPassword"
/></label><br>
                <button type="submit">Сохранить</button>
                <button type="button"
onclick="closeModal('editDoctorModal')">Закрыть</button>
            </form>
        </div>
    </div>
```

```
        <!--
        <div sec:authorize="hasRole('ADMIN')">
            <a th:href="@{/auth/register/doctor}">Зарегистрировать доктора</a>
        </div>
        -->
        <div sec:authorize="hasRole('ADMIN')">
            <a th:href="@{/auth/register/client}">Зарегистрировать нового клиента</a>
        </div>


        <a href="#" sec:authorize="hasRole('ADMIN')"
onclick="openAddDoctorModal()">Добавить доктора</a>

    </th:block>
</th:block>
</html>
```