Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Отчет по курсу
Дисциплина: «Технологии сетевого программирования»

Выполнили:
Абубекяров Н.Р.
Хромов А.В.

Группа:
6301-010302D

Самара 2025

1 Общая архитектура приложения

1.1 Описание приложения

Данное веб-приложение представляет собой интернет-магазин с каталогом товаров, возможностью авторизации пользователей, оформления заказов, управления корзиной, профилем пользователя и просмотра информации о товарах.

1.2 Состав приложения

Приложение состоит из следующих функциональных частей:

1. Серверная часть приложения.
2. Клиентская часть приложения.
3. База данных.

1.3 Описание функциональных частей

1.3.1 Серверная часть

Серверная часть написана на языке программирования Java с использованием фреймворка Spring Boot. Проект использует Spring Security (см. Приложение 1) для авторизации и аутентификации, Spring Web для построения REST API (см. Приложение 2) и Spring Data JPA для работы с базой данных (см. Приложение 3). Безопасность реализована на основе JWT с поддержкой обновления токенов. Сборка и управление зависимостями выполняются с помощью Maven.

1.3.2 Клиентская часть

Клиентская часть реализована на языке JavaScript с использованием фреймворка React 19. Библиотека ReactRouter DOM используется для маршрутизации в проекте (см. Приложение 4), ReactIcons предоставляет популярные иконки, которые были использованы на сайте, Axios для взаимодействия с REST API (см. Приложение 5), а Zustand для управления состоянием приложения, примеры метода login и addInCard см. Приложение 6. Также были использованы HTML и модульные стили CSS.

1.3.3 База данных

В качестве СУБД используется PostgreSQL. А для работы с ней применяются ORM модели (см. Приложение 7).

1.4 Система авторизации и безопасности

Система авторизации реализована через JWT (JSON Web Token). Используется accessи refreshтокены. Accessиспользуется для краткосрочного доступа (время его действия принято устанавливать не больше 2 часов), а refreshдля обновления access токена (время действия от 7 до 30 дней) (см. Приложение 8).

1.5 Функционирование приложения

Приложение собрано и запускается с помощью файла docker-compose.yml, находящегося в корне проекта и который объединяет 3 контейнера (бэкенд, фронтенд и базу данных (см. Приложение 9).

2 Работа приложения

2.1 Регистрация и авторизация

На рисунке 1 представлена страница регистрации, где пользователь вводит свои данные (имя, фамилию, электронную почту, пароль, номер телефона и адрес). Зарегистрировать пользователя с уже имеющейся почтой в БД не выйдет. После успешной регистрации, пользователь может войти в магазин на странице входа, показанной на рисунке 2.

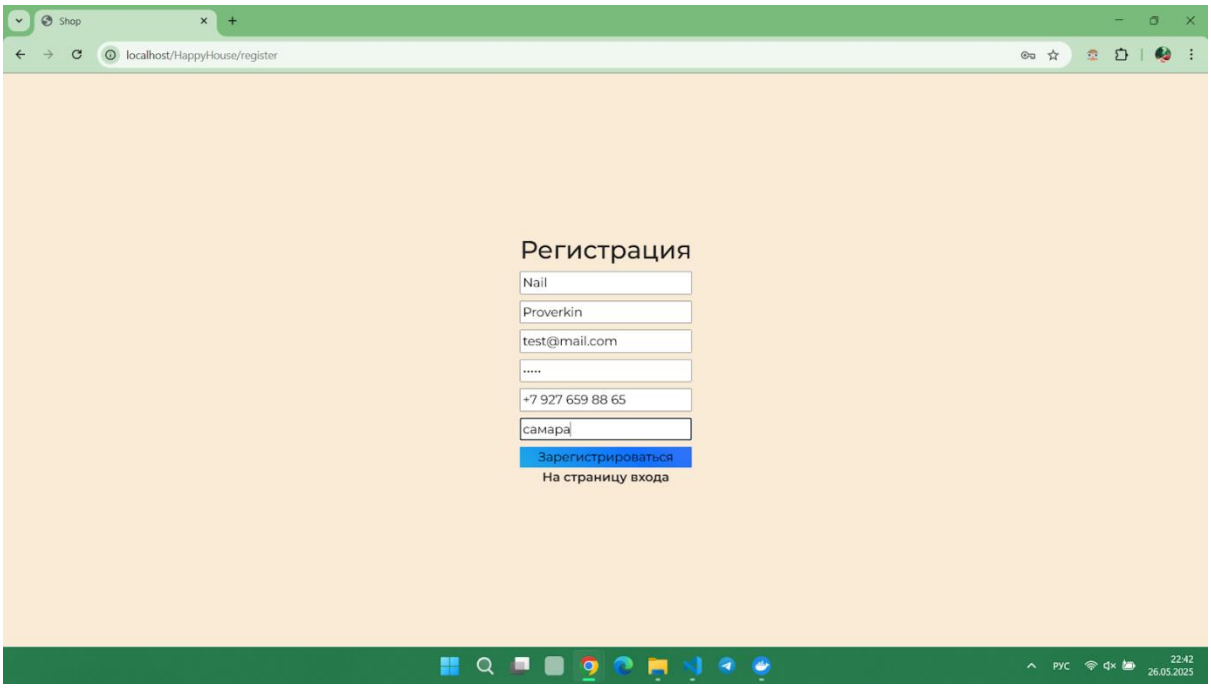

Рисунок 1 –Страница регистрации

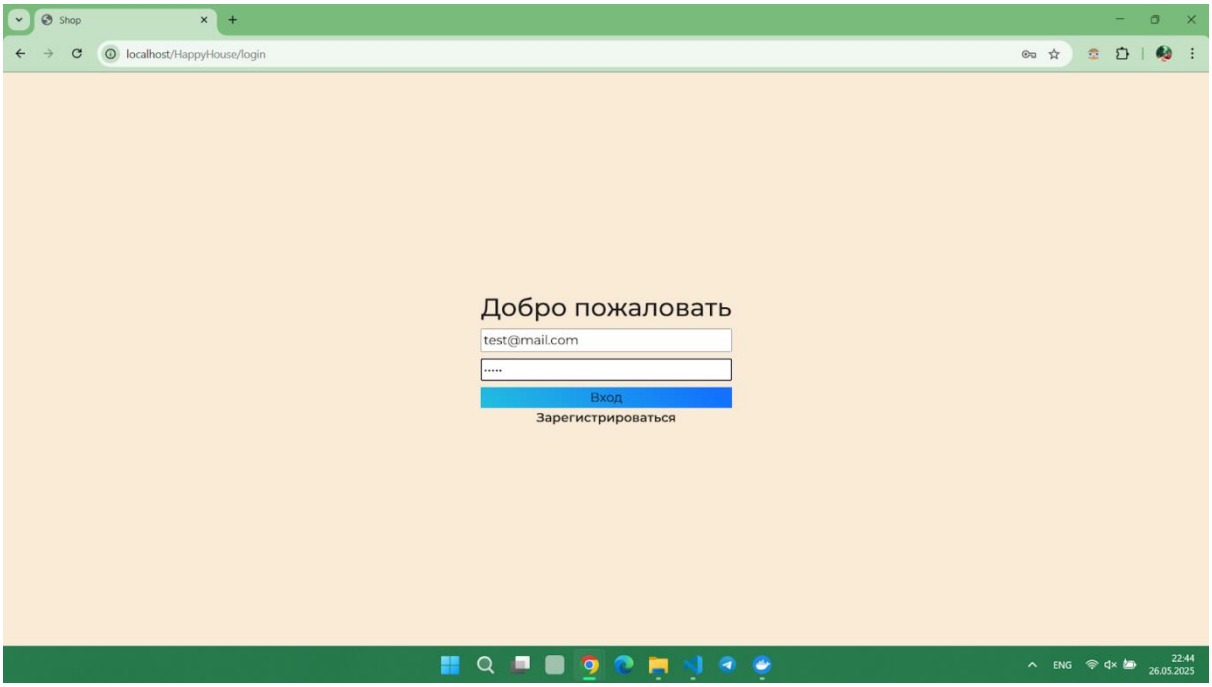

Рисунок 2– Страница входа

Далее открывается главная страница магазина, изображенная на рисунках 3, 4, на которой можно просматривать товары. Сверху находится панель навигации с главными страницами.

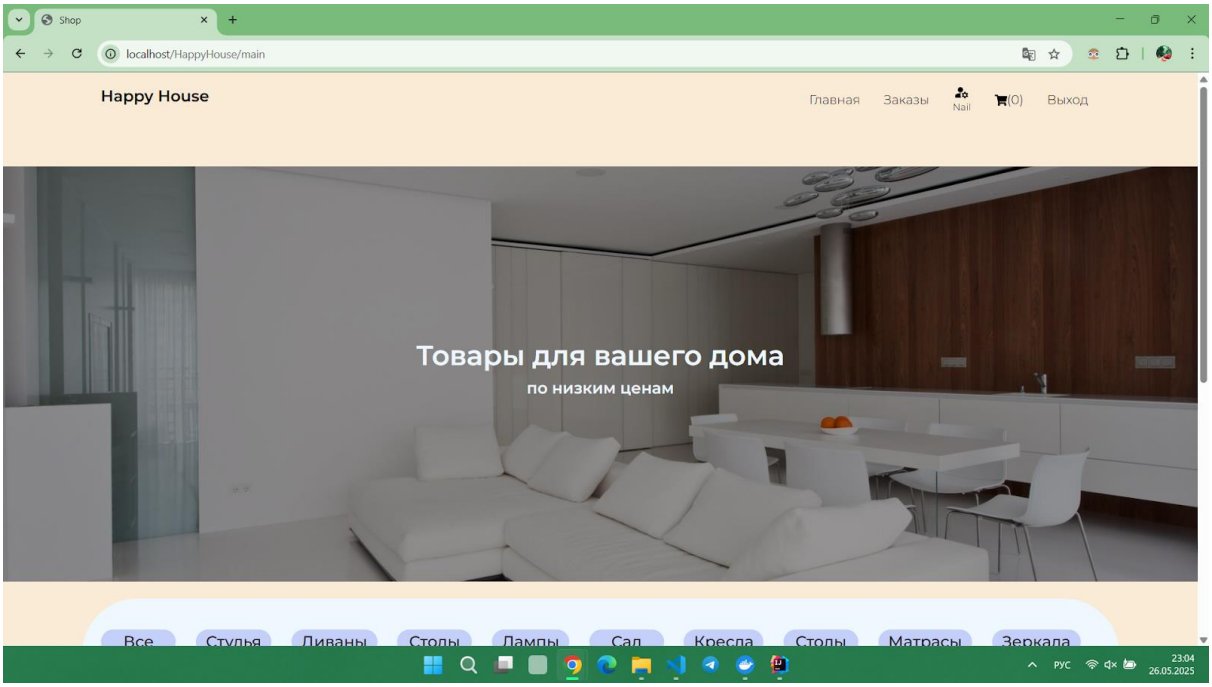

Рисунок 3 – Главная страница

2.1 Изменение корзины

Чтобы добавить товар в корзину пользователь должен нажать на синюю кнопку добавления внутри карточки товара, показанную на рисунке 4. После этого можно увидеть добавленную мебель на странице корзины (рисунок 5). Также в панели навигации обновится число товаров корзины.

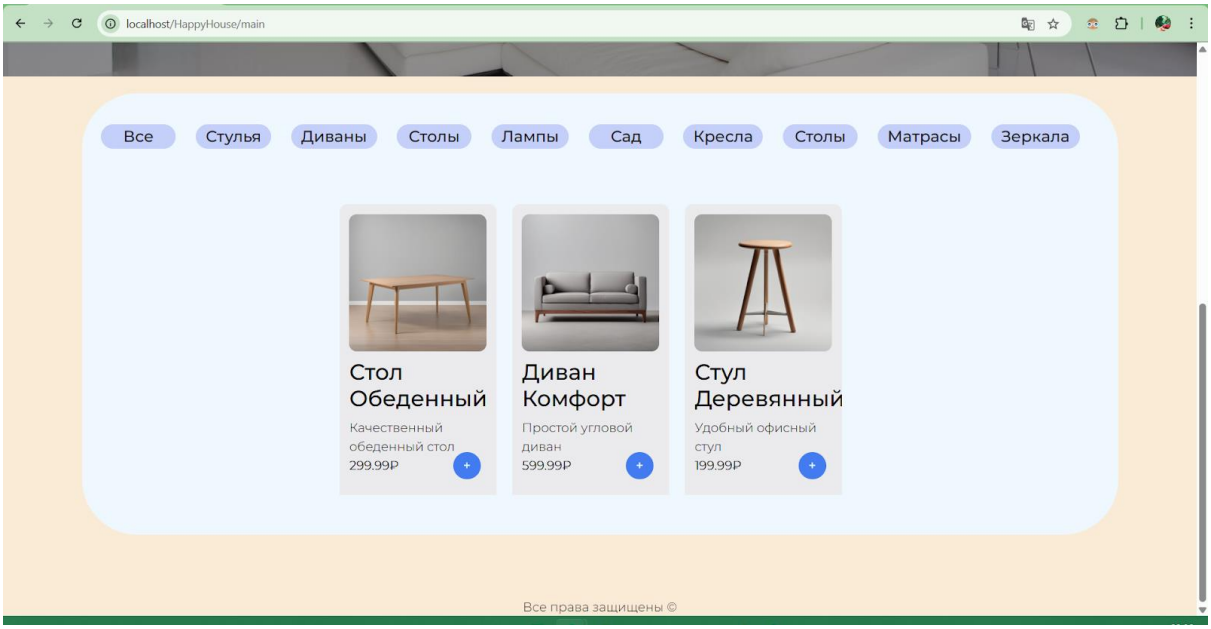

Рисунок 4 – Главная страница, товары

На странице корзины есть возможность изменить количество единиц товара или удалить товар из корзины вовсе.

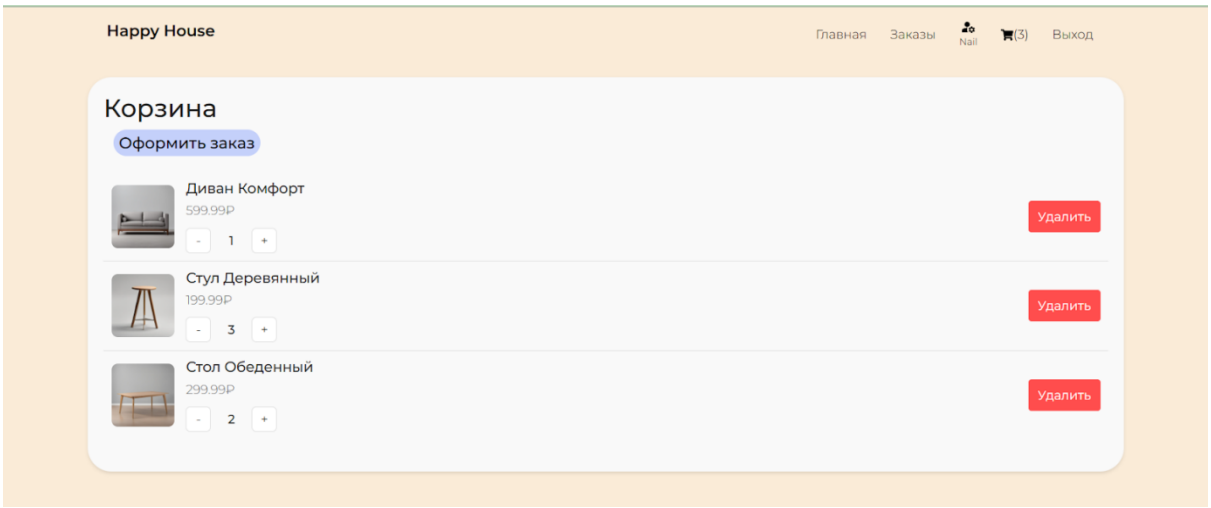

Рисунок 5–Страница корзины

2.2 Изменение данных пользователя

На странице профиля есть возможность поменять имя, фамилию, адрес и номер телефона, а также пароль (все кроме почты). Для изменения пароля необходимо нажать на соответствующую кнопку, написать актуальный пароль и новый нужно написать 2 раза, при несоответствии паролей, выйдет ошибка.

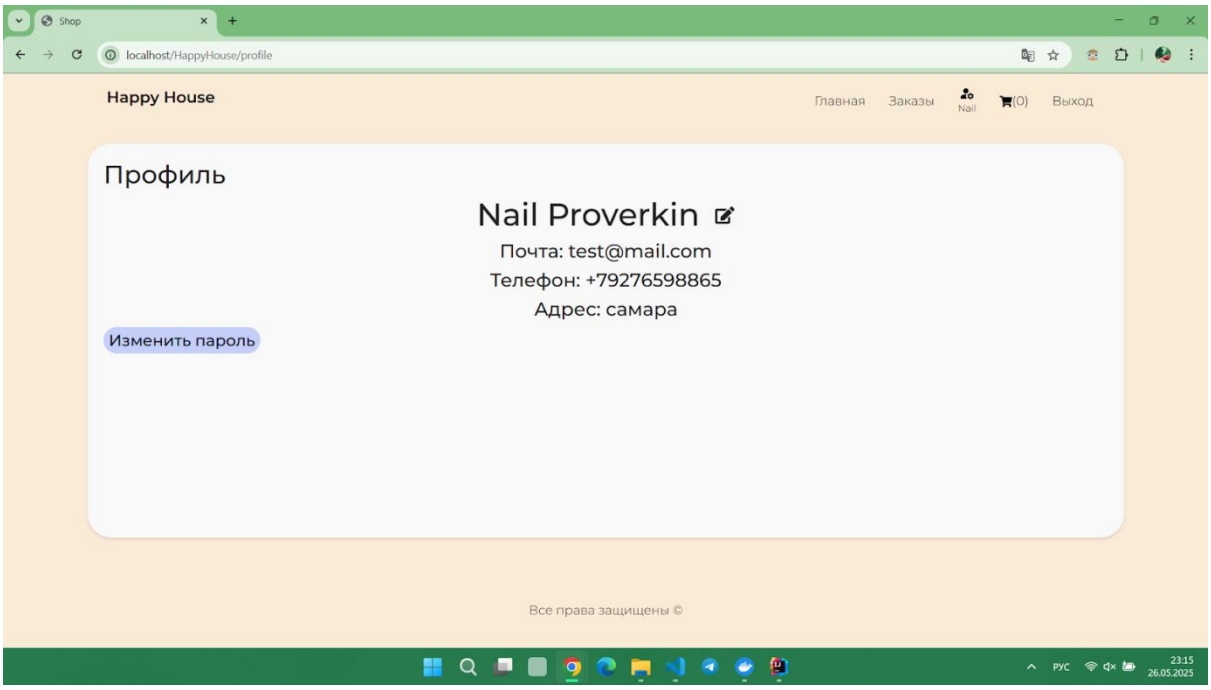

Рисунок 6– Профиль пользователя

Для изменения оставшихся данных надо нажать на иконку ручки, рядом с фамилией пользователя, как на рисунке 6.

После сохранения данных выходит сообщение об успешности изменений. Поле можно увидеть на рисунке 7.

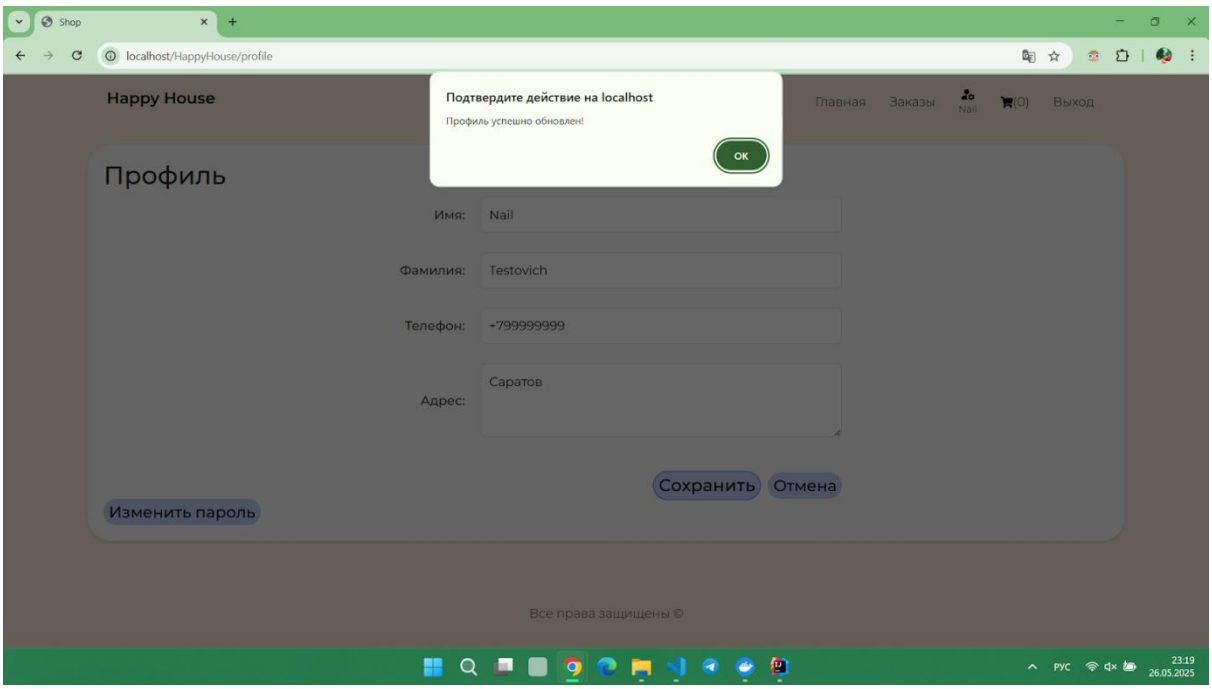

Рисунок 7 – Успешное изменение данных пользователя

2.3 Страница товара и страница заказов

При нажатии на товар на странице корзины или на главной, появляется страница товара, изображенная на рисунке 6, на которой можно более детально рассмотреть изображение или также добавить или удалить товар. При изменении статуса товара меняется стиль кнопки добавления, а также сам текст внутри нее.

Находясь на странице корзины пользователь имеет возможность оформить заказ. После чего он добавиться в историю заказов, а корзина очистится. Заказы можно проверить на соответствующей странице. На рисунке 9 можно увидеть данную страницу.

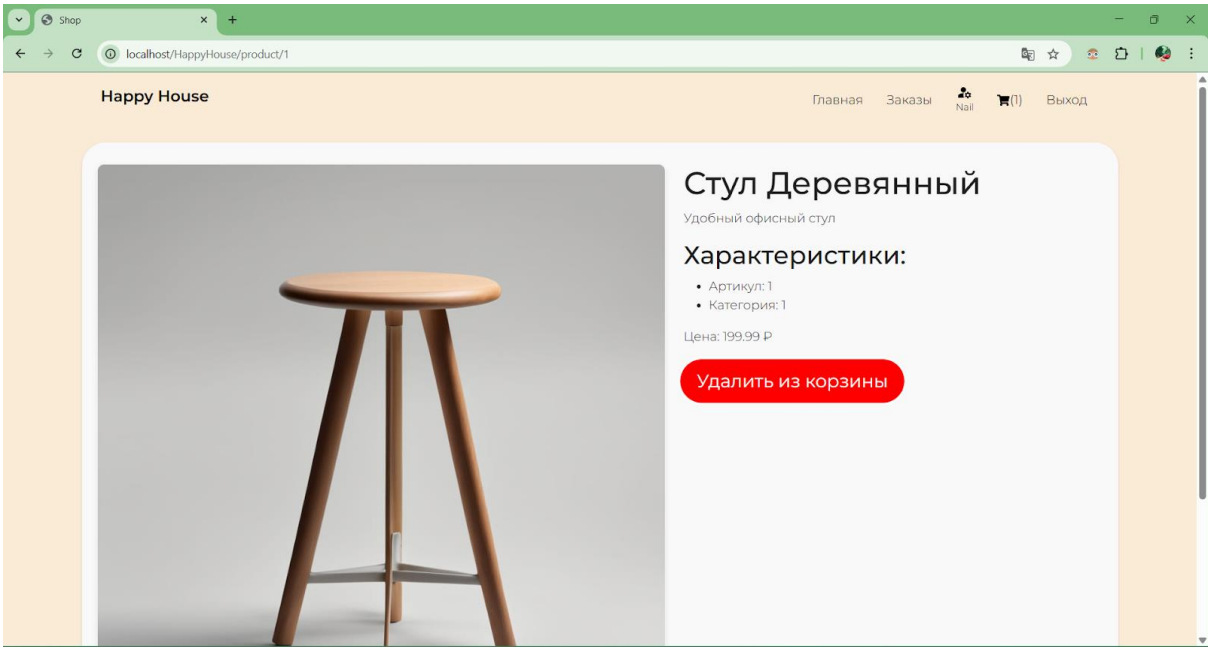

Рисунок 8 - Страница товара

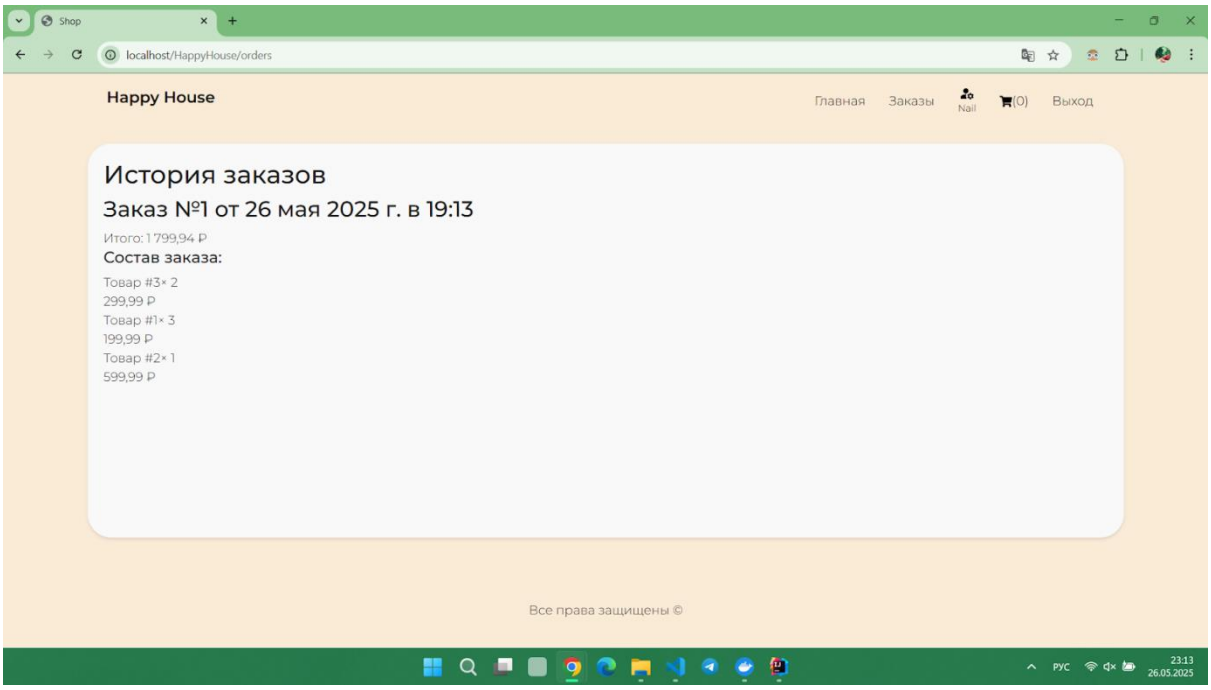

Рисунок 9 – Страница заказов

2.4 Выход из профиля

При нажатии на "Выход", стираются все данные пользователя (срабатывает функция, стирающая данные из localstorage: userId, access/refresh токены) и перед ним открывается страница входа. Если человек захочет переместиться на другую страницу приложения через адресную строку, его обратно перебросит на "вход".

ЗАКЛЮЧЕНИЕ

В ходе работы были освоены ключевые технологии: Spring Security, JWT, ReactRouter, Zustand,ORM.Стала более понятна работа с RESTAPI, контейнеризацией и работа с такими инструментами как Postman, DBeaver.

Были выполнены все основные требования к курсу:

1. Проектирование приложения.
2. Разработка базы данных.
3. Разработка API.
4. Авторизация.
5. Разработка клиентской части
6. Финализация приложения и упаковка в Docker

Что можно улучшить:

1. Добавить тесты (JUnit, Jest).
2. Реализовать ролевую модель (админ/пользователь).

Приложение 1

Файл SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

   private final CustomUserDetailsServiceuserDetailsService;
   private final JwtAuthenticationFilterjwtAuthenticationFilter;

   public SecurityConfig(CustomUserDetailsServiceuserDetailsService,
JwtAuthenticationFilterjwtAuthenticationFilter) {
this.userDetailsService = userDetailsService;
this.jwtAuthenticationFilter = jwtAuthenticationFilter;
   }

   @Bean
   public PasswordEncoderpasswordEncoder(){
      return new BCryptPasswordEncoder();
   }

   @Bean
   public SecurityFilterChainfilterChain(HttpSecurity http) throws Exception
{
      return http
.cors(cors ->cors.configurationSource(corsConfigurationSource())) // До-
бавляем CORS
.csrf(csrf ->csrf.disable())
.sessionManagement(session -
>session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
```

```
        .authorizeHttpRequests(auth -> auth
        .requestMatchers(
                                "/happyhouse/auth/register",
                                "/happyhouse/auth/login",
                                "/happyhouse/auth/refresh",
                                "/error"
        ).permitAll()
        .anyRequest().authenticated()
                    )
        .addFilterBefore(jwtAuthenticationFilter,      UsernamePasswordAuthentica-
tionFilter.class)
        .build();
           }

           @Bean
           public CorsConfigurationSourcecorsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of(
                "http://localhost:3000",
        "http://localhost",
                "http://localhost:80",
                "http://frontend:80"  // Добавляем доступ из контейнера фронтен-
да
        ));
        configuration.setAllowedMethods(List.of("*"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigura-
tionSource();
```

```
        source.registerCorsConfiguration("/**", configuration);
            return source;
        }

        @Bean
        public AuthenticationManagerauthenticationManager(HttpSecurity http)
throws Exception {
        AuthenticationManagerBuilder auth =
http.getSharedObject(AuthenticationManagerBuilder.class);
        auth.userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
            return auth.build();
        }
        }
```

Приложение 2

Файл AuthController.java

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

            private final UserServiceuserService;
            private final AuthenticationManagerauthenticationManager;
            private final PasswordEncoderpasswordEncoder;
            private final RefreshTokenServicerefreshTokenService;
            private final JwtUtiljwtUtil;

            @PostMapping("/register")
```

```
    public ResponseEntity<UserDto>register(@RequestBody UserRegistrationDtoregistrationDto) {
        User user = new User();
        user.setFirstName(registrationDto.getFirstName());
        user.setLastName(registrationDto.getLastName());
        user.setEmail(registrationDto.getEmail());
        // Шифруем пароль с помощью BCrypt
        user.setPasswordHash(passwordEncoder.encode(registrationDto.getPassword()));
        user.setPhoneNumber(registrationDto.getPhoneNumber());
        user.setAddress(registrationDto.getAddress());

        User registeredUser = userService.registerUser(user);
        return ResponseEntity.ok(DtoConverter.convertUserToDto(registeredUser));
    }

    @PostMapping("/login")
    public ResponseEntity<?>login(@RequestBody UserRegistrationDtologinRequest) {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
        loginRequest.getEmail(),
        loginRequest.getPassword()
            )
        );

    CustomUserDetailsuserDetails = (CustomUserDetails) authentication.getPrincipal();
        String accessToken = jwtUtil.generateAccessToken(userDetails);
```

```
RefreshTokenrefreshToken = refreshToken-
Service.createRefreshToken(userDetails.getUser());

    return ResponseEntity.ok(Map.of(
        "accessToken", accessToken,
        "refreshToken", refreshToken.getToken()/*,

    @PostMapping("/refresh")
    @Transactional
    public ResponseEntity<?>refreshToken(@RequestBody RefreshToken-
RequestDto request) {
    String refreshToken = request.getRefreshToken();

    return refreshTokenService.findByToken(refreshToken)
.map(token -> {
        // Проверяем срок действия
refreshTokenService.verifyExpiration(token);

// Получаем пользователя из токена
        User user = token.getUser();

        // Отзываем старый токен И ВСЕ ДРУГИЕ токены этого
пользователя
refreshTokenService.revokeAllUserTokens(user.getUserId());

        // Генерируем новые токены
String newAccessToken = jwtUtil.generateAccessToken(new Cus-
tomUserDetails(user));
RefreshTokennewRefreshToken = refreshToken-
Service.createRefreshToken(user);
```

```
                return ResponseEntity.ok(new TokenRefreshResponseDto(
        newAccessToken,
        newRefreshToken.getToken()
                ));
            })
        .orElseThrow(() -> new RuntimeException("Invalid refresh token"));
        }
        }
```

Приложение 3

Файл RefreshTokenRepository.java

```
public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Long> {
    //    Optional<RefreshToken>findByToken(String token);
        @Modifying
        @Query("DELETE FROM RefreshToken rt WHERE rt.user.userId= :userId")
        void deleteByUserId(@Param("userId") Long userId);

        @Modifying
        @Query("DELETE FROM RefreshToken rt WHERE rt.token= :token")
        void deleteByToken(@Param("token") String token);

        Optional<RefreshToken>findByToken(String token);

        @Query("SELECT rt FROM RefreshToken rt WHERE rt.user.userId= :userId")
        List<RefreshToken>findAllByUserId(@Param("userId") Long userId);
        }
```

Приложение 4

ФайлApp.js

```
<Router >
<div className="App">
<Routes>
<Route path="/HappyHouse" element={<Navigate to="/HappyHouse/login" replace />} />

        {/* Публичные маршруты */}
<Route path="/HappyHouse/login" element={
<AuthRedirect>
<Login />
</AuthRedirect>
        } />

<Route path="/HappyHouse/register" element={
<AuthRedirect>
<Registration />
</AuthRedirect>
        } />

        {/* Защищенные маршруты */}
<Route path="/HappyHouse" element={<MainLayout />}>
<Route path="main" element={
<ProtectedRoute>
<Main />
</ProtectedRoute>
          } />
```

```
<Route path="cart" element={
<ProtectedRoute>
<Cart />
</ProtectedRoute>
        } />

<Route path="product/:id" element={
<ProtectedRoute>
<Product_pg />
</ProtectedRoute>
        } />

<Route path="profile" element={
<ProtectedRoute>
<Profile />
</ProtectedRoute>
        } />

<Route path="orders" element={
<ProtectedRoute>
<Orders />
</ProtectedRoute>
        } />
</Route>

        {/* Обработка несуществующих маршрутов */}
<Route path="*" element={<NotFound />} />
</Routes>
</div>
</Router>
```

Приложение 5

Перехватчик ответов в axios.js

```
// Перехватчик ответов
instance.interceptors.response.use(
response => response,
  async error => {
    const originalRequest = error.config;

// Пропускаемобработкудля /auth/refresh
    if (
error.config.url?.includes('/auth/refresh') ||
error.response?.status !== 401
    ) {
      return Promise.reject(error);
    }

    if (!originalRequest._retry) {
originalRequest._retry = true;

      if (isRefreshing) {
        return new Promise((resolve, reject) => {
failedQueue.push({ resolve, reject });
        })
.then(token => {
originalRequest.headers.Authorization = `Bearer ${token}`;
           return instance(originalRequest);
         })
.catch(err =>Promise.reject(err));
      }
```

```
        isRefreshing = true;

            try {
              // 1. Обновляем токены
              const newAccessToken = await store.getState().refreshTokens();

              // 2. Обновляем глобальные настройки
        instance.defaults.headers.common['Authorization'] = `Bearer
${newAccessToken}`;

              // 3. Обновляем оригинальный запрос
        originalRequest.headers.Authorization = `Bearer ${newAccessToken}`;

              // 4. Повторяем запросы из очереди
        processQueue(null, newAccessToken);

        // 5. Возвращаем оригинальный запрос
              return instance(originalRequest);
        } catch (refreshError) {
              // 6. Очищаем данные при ошибке
        processQueue(refreshError, null);
        store.getState().clearUser();
        window.location.href = '/HappyHouse/login';
              return Promise.reject(refreshError);
            } finally {
        isRefreshing = false;
            }
          }

          return Promise.reject(error);
```

```
}
);
```

Приложение 6

Метод добавления товара в корзину

```
addInCart: (item) =>
set((state) => {
      if (!state.cart.some((el) =>el.productId === item.productId)) {
        return { cart: [...state.cart, item] };
}
      return state;
    }),
```

Метод аутентификации пользователя

```
login: async (email, password) => {
set({ loading: true, error: null });
    try {

      const { data } = await instance.post('/happyhouse/auth/login', {
        email,
        password
      });

localStorage.setItem('accessToken', data.accessToken);
localStorage.setItem('refreshToken', data.refreshToken);

      // 1. Получаем данные из токена
      const userDatarow = getCurrentUser();
localStorage.setItem('userId', userDatarow.id);
      // 2. Загружаем данные пользователя
      const userData = await get().fetchUserData(userDatarow.id);
```

```
        // 3. Загружаемкорзину
        const cartData = await get().fetchCart(userData.userId);

      set({
          user: {
            id: userData.userId,
      firstName: userData.firstName,
      lastName: userData.lastName,
      email:userData.email,
          },
          // cart: cartData,
          loading: false
        });

      } catch (error) {
      set({ error: error.message, loading: false });
        throw error;
      }
    },
```

Приложение 7

Настройки БД в application.yml

```
spring:
datasource:
   # используемENV-переменнуюSPRING_DATASOURCE_URL,
# если она не задана — откатимся к локальной разработке
url:
${SPRING_DATASOURCE_URL:jdbc:postgresql://localhost:5433/furniture_db}
    username: ${SPRING_DATASOURCE_USERNAME:furniture_user}
    password: ${SPRING_DATASOURCE_PASSWORD:furniture_pass}
```

```
    driver-class-name: org.postgresql.Driver

  jpa:
    hibernate:
  ddl-auto: none
    show-sql: true
    properties:
      hibernate:
  format_sql: true

  flyway:
    enabled: true
    locations: classpath:db/migration
```

Приложение 8

Настройки jwt в application.yml

```
jwt:
  # ENV-переменныеJWT_SECRET, ACCESSиREFRESH_EXPIRATION
  secret: ${JWT SECRET:mySuperSecretKeyThatIsAtLeast32BytesLong!}
  access-expiration-ms: ${JWT_ACCESS_EXPIRATION_MS:15000}
  refresh-expiration-ms:
${JWT_REFRESH_EXPIRATION_MS:2592000000}
```

Файл JwtUtil.java

```
@Data
@Component
public class JwtUtil {

    @Value("${jwt.secret}")
    private String secret;
```

```
    @Value("${jwt.access-expiration-ms}")
    private long accessExpirationMs;

    @Value("${jwt.refresh-expiration-ms}")
    private long refreshExpirationMs;

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

    public Date extractExpiration(String token) {
       return extractClaim(token, Claims::getExpiration);
    }

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

    private Claims extractAllClaims(String token) {
       return Jwts.parserBuilder()
    //
.setSigningKey(Keys.hmacShaKeyFor(SECRET_KEY.getBytes(StandardCharsets
.UTF_8)))
    .setSigningKey(Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UT
F_8)))
    .build()
    .parseClaimsJws(token)
    .getBody();
```

```
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public String generateAccessToken(UserDetailsuserDetails) {
        Map<String, Object> claims = new HashMap<>();
    CustomUserDetailscustomDetails = (CustomUserDetails) userDetails;

        // Добавляем дополнительные данныев claims
    claims.put("id", customDetails.getId());
    claims.put("firstName", customDetails.getUser().getFirstName());

        String subject = userDetails.getUsername();

        return buildToken(claims, subject, accessExpirationMs);
    }
    private String buildToken(Map<String, Object> claims, String subject,
long expiration) {
        return Jwts.builder()
    .setClaims(claims)
    .setSubject(subject)
    .setIssuedAt(new Date())
    .setExpiration(new Date(System.currentTimeMillis() + expiration))
    .signWith(Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8))
, SignatureAlgorithm.HS256)
    .compact();
    }
```

```
    public Boolean validateToken(String token, UserDetailsuserDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }
}
```

Файл RefreshTokenService.java

```
@Service
@RequiredArgsConstructor
public class RefreshTokenService {

    private final RefreshTokenRepositoryrefreshTokenRepository;
    private final JwtUtiljwtUtil;

    @Transactional
    public RefreshTokencreateRefreshToken(User user) {
RefreshTokenrefreshToken = new RefreshToken();
refreshToken.setUser(user);
        refreshToken.setExpiryDate(Instant.now().plusMillis(jwtUtil.getRefreshExpirationMs()));
refreshToken.setToken(UUID.randomUUID().toString());
        return refreshTokenRepository.save(refreshToken);
    }
    @Transactional
    public void revokeAllUserTokens(Long userId) {
refreshTokenRepository.deleteByUserId(userId);
    }

    @Transactional
    public void revokeToken(String token) {
```

```
refreshTokenRepository.deleteByToken(token);
    }

    public RefreshTokenverifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().isBefore(Instant.now())) {
refreshTokenRepository.delete(token);
            throw new RuntimeException("Refresh token expired");
        }
        return token;
    }
    public Optional<RefreshToken>findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }
}
```

Приложение 9

Файл docker-compose.yml

```
services:
db:
image: postgres:15-alpine
container_name: furniture_store_db
environment:
      - POSTGRES_USER=furniture_user
      - POSTGRES_PASSWORD=furniture_pass
      - POSTGRES_DB=furniture_db
ports:
      - "5433:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - app-network
```

```
  backend:
    build:
      context: ./backend
  dockerfile: Dockerfile
  container_name: furniture_store_backend
    environment:
      -
SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/furniture_db
      - SPRING_DATASOURCE_USERNAME=furniture_user
      - SPRING_DATASOURCE_PASSWORD=furniture_pass
      - JWT_SECRET=mySuperSecretKeyThatIsAtLeast32BytesLong!
      - JWT_ACCESS_EXPIRATION_MS=15000
  - JWT_REFRESH_EXPIRATION_MS=2592000000
  ports:
      - "8080:8080"
  depends_on:
      - db
    networks:
      - app-network

  frontend:
    build:
      context: ./frontend  #Путь к папке с Dockerfile фронтенда
  dockerfile: Dockerfile
  container_name: furniture_store_frontend
  ports:
      - "80:80"
  environment:
      - API_BASE_URL=http://localhost:8080
```

```
depends_on:
      - backend
    networks:
      - app-network

volumes:
db_data:

networks:
  app-network:
driver: bridge
```