# **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. Приложение реализует полную клиент-серверную архитектуру с использованием контейнеризации (Docker).

## **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**.  
Также используются:

1. **Spring Security** – для авторизации и аутентификации.
2. **Spring Web** – для построения REST API.
3. **Spring Data JPA** – для взаимодействия с базой данных.
4. **JWT** – для реализации безопасности и обновления токенов.
5. **Maven** – для сборки и управления зависимостями.
6. **Dockerfile** – для контейнеризации бэкенда.

Сервер реализует следующие REST-контроллеры:

1. **AuthController** – регистрация, вход, обновление токена.
2. **UserController** – изменение пароля, получение информации о пользователе.
3. **ProductController** – управление товарами.
4. **CartController** – управление корзиной пользователя.
5. **OrderController** – оформление и просмотр заказов.

Также используются классы DTO для передачи информации между клиентом и сервером (UserDto, ProductDto, OrderDto и др.).

Реализация **CartController**

package com.example.backend.controllers;

import com.example.backend.dto.CartDto;

import com.example.backend.dto.DtoConverter;

import com.example.backend.models.Cart;

import com.example.backend.services.CartService;

import lombok.RequiredArgsConstructor;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.security.access.prepost.PreAuthorize;

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

@RestController

@RequestMapping("/happyhouse/cart")

@RequiredArgsConstructor

public class CartController {

private final CartService cartService;

@GetMapping("/{userId}")

@ResponseStatus(HttpStatus.*OK*)

@PreAuthorize("authentication.principal.id == #userId")

public CartDto getCart(@PathVariable("userId") Long userId) {

Cart cart = cartService.getCartByUserId(userId);

return DtoConverter.*convertCartToDto*(cart);

}

@PostMapping("/{userId}")

@PreAuthorize("authentication.principal.id == #userId")

public ResponseEntity<CartDto> addProductToCart(@PathVariable("userId") Long userId,

@RequestParam("productId") Long productId,

@RequestParam("count") int count) {

Cart cart = cartService.addProductToCart(userId, productId, count);

return ResponseEntity.*ok*(DtoConverter.*convertCartToDto*(cart));

}

@PutMapping("/{cartItemId}")

public ResponseEntity<CartDto> updateCartItemCount(@PathVariable("cartItemId") Long cartItemId,

@RequestParam("count") int newCount) {

Cart cart = cartService.updateCartItemCount(cartItemId, newCount);

return ResponseEntity.*ok*(DtoConverter.*convertCartToDto*(cart));

}

@DeleteMapping("/{cartItemId}")

public ResponseEntity<CartDto> removeProductFromCart(@PathVariable("cartItemId") Long cartItemId) {

Cart cart = cartService.removeProductFromCart(cartItemId);

return ResponseEntity.*ok*(DtoConverter.*convertCartToDto*(cart));

}

}

### **1.3.2 Клиентская часть**

Клиентская часть реализована на языке JavaScript с использованием фреймворка **React 19, React Router DOM, React Icons**.  
Дополнительно используются:

* **CSS-модули** для стилизации компонентов.
* **Axios** – для взаимодействия с REST API.
* **Zustand** – для управления состоянием.

### **1.3.3 База данных**

В качестве СУБД используется **PostgreSQL**.  
Через Spring Data JPA создаются репозитории для моделей:

* Пользователь (User).
* Товары (Product).
* Корзина(Cart) и элементы корзины(CartItem).
* Заказы(Order) и их элементы(OrderItem).

Все сущности представлены в виде ORM-классов (Entity).

Настройка БД:

spring:

datasource:

*# используем ENV-переменную SPRING\_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

jwt:

*# ENV-переменные JWT\_SECRET, ACCESS и REFRESH\_EXPIRATION*

secret: ${JWT\_SECRET:mySuperSecretKeyThatIsAtLeast32BytesLong!}

access-expiration-ms: ${JWT\_ACCESS\_EXPIRATION\_MS:15000}

refresh-expiration-ms: ${JWT\_REFRESH\_EXPIRATION\_MS:2592000000}

### **1.3.4 Система авторизации и безопасности**

Система авторизации реализована через **JWT (JSON Web Token)**.  
Используется два токена:

* **Access Token** – для краткосрочного доступа.
* **Refresh Token** – для обновления access-токена без повторной авторизации.

Реализация Сервиса, отвечающего за рефреш токен

package com.example.backend.services;

import com.example.backend.models.RefreshToken;

import com.example.backend.models.User;

import com.example.backend.repositories.RefreshTokenRepository;

import com.example.backend.security.JwtUtil;

import jakarta.transaction.Transactional;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Service;

import java.time.Instant;

import java.util.Optional;

import java.util.UUID;

@Service

@RequiredArgsConstructor

public class RefreshTokenService {

private final RefreshTokenRepository refreshTokenRepository;

private final JwtUtil jwtUtil;

@Transactional

public RefreshToken createRefreshToken(User user) {

RefreshToken refreshToken = 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 RefreshToken verifyExpiration(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);

}

}

### **1.3.5 Управление товарами и заказами**

Пользователи могут:

* Просматривать товары и категории.
* Добавлять товары в корзину.
* Удалять из корзины.
* Оформлять заказы.
* Просматривать корзину и историю заказов

## **1.4 Функционирование приложения**

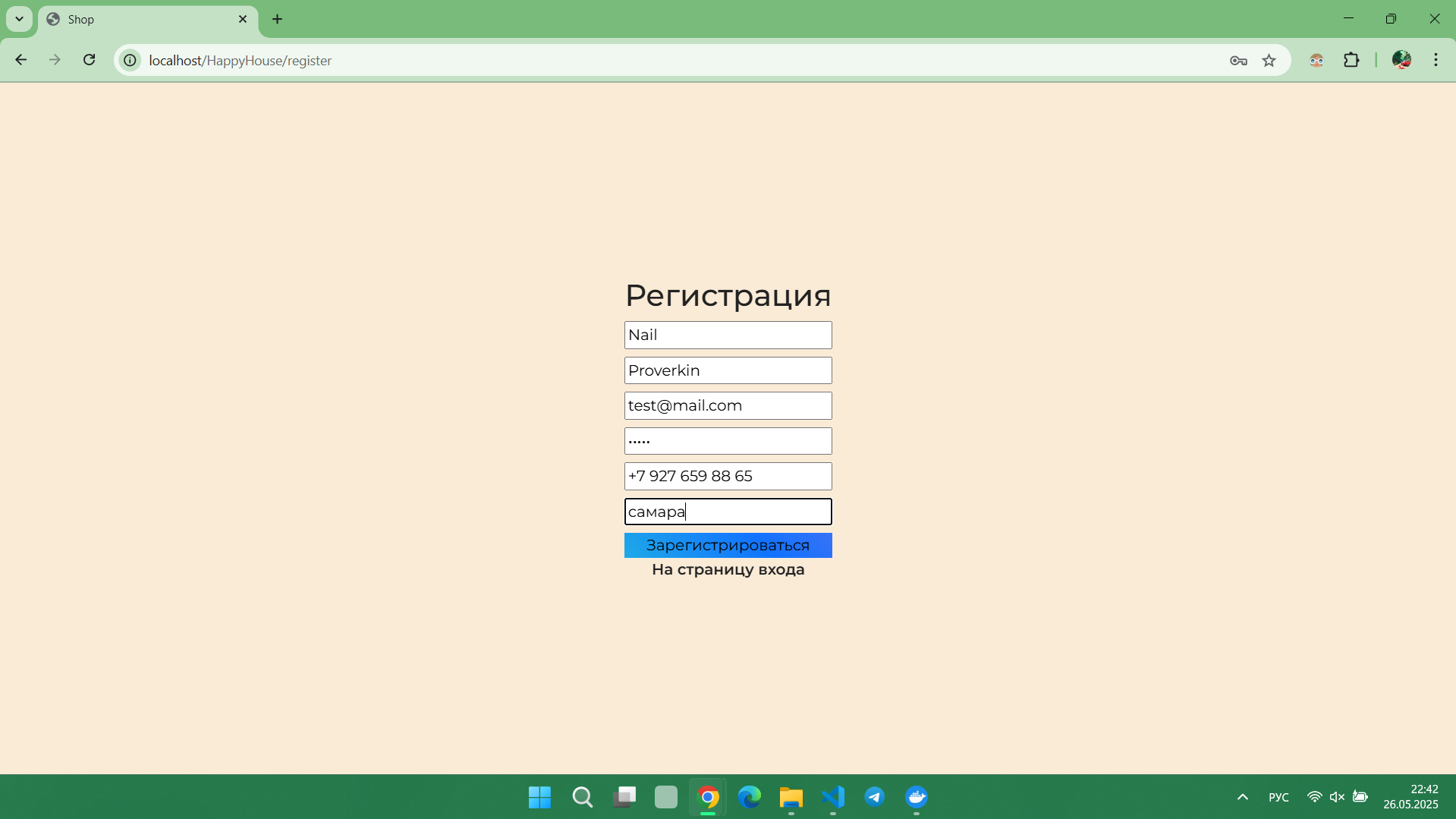
Приложение собрано и запускается с помощью docker-compose.yml, который объединяет:

* Контейнер с бэкендом.
* Контейнер с фронтендом.
* Контейнер с базой данных PostgreSQL.

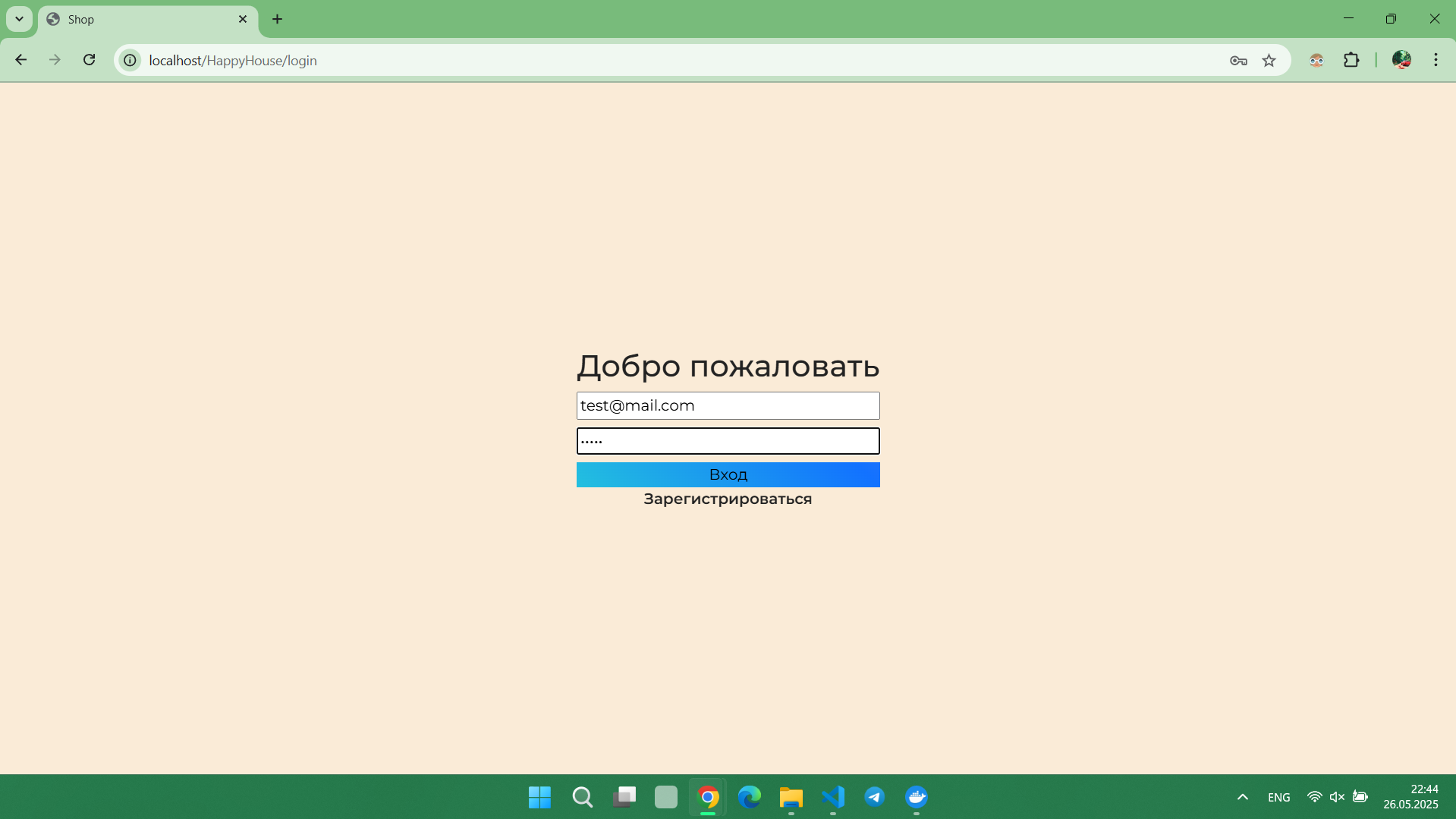
Все сервисы взаимодействуют по внутренней сети Docker, и приложение готово к запуску в продакшн-среде. Сборка фронтенда осуществляется через Node.js, а сервер собирается Maven и запускается в контейнере OpenJDK.

# **2 Пример работы приложения**

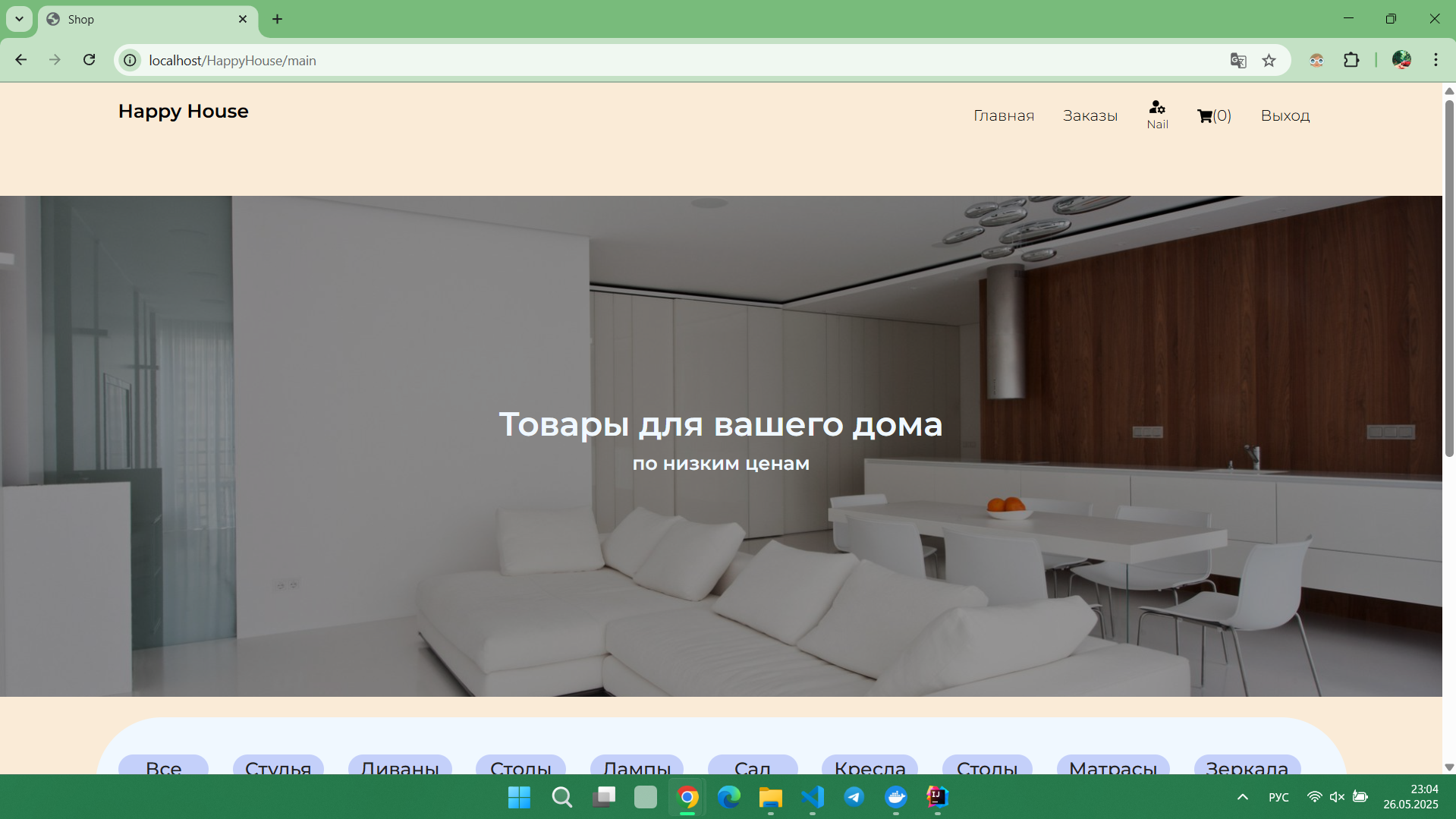
1. Зарегистрируем нового пользователя:



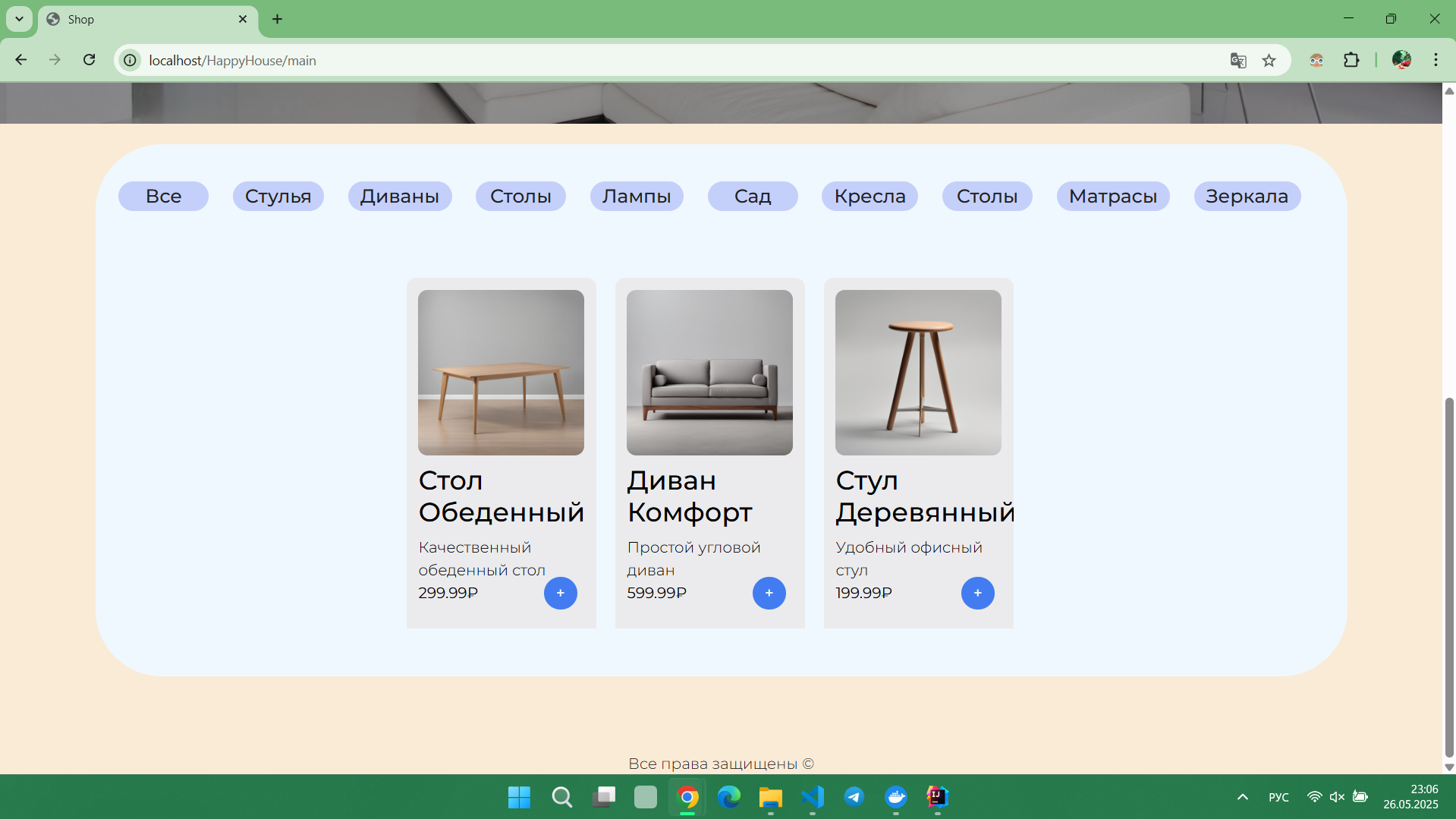
1. После регистрации пользователю открывается страница входа в личный кабинет:



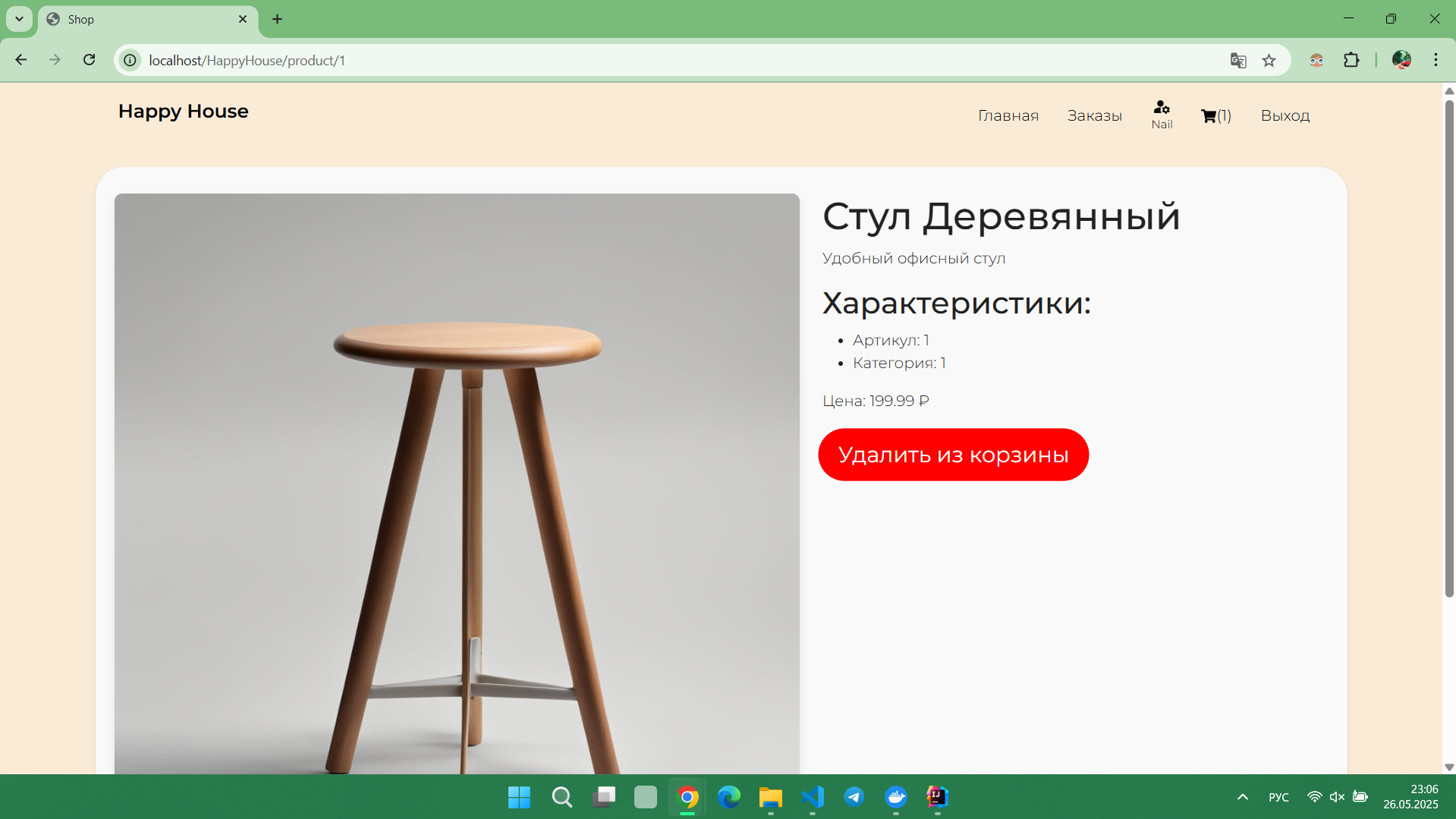
1. После успешного входа открывается главная страница магазина на которой можно просматривать товары и добавлять понравившиеся в корзину. Сверху находится панель навигации с главными страницами.



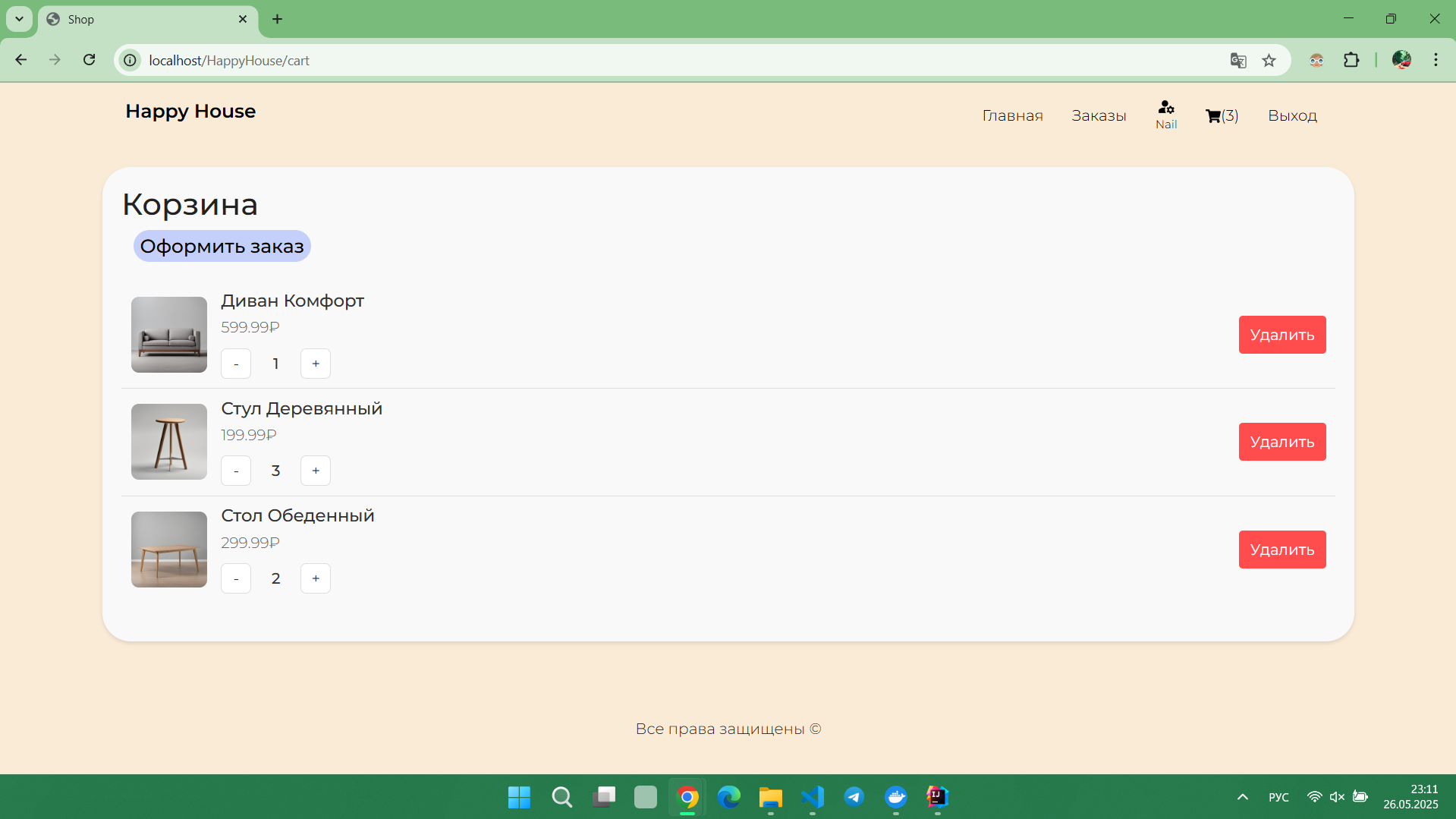
1. На страницу товара можно перейти нажав на товар.



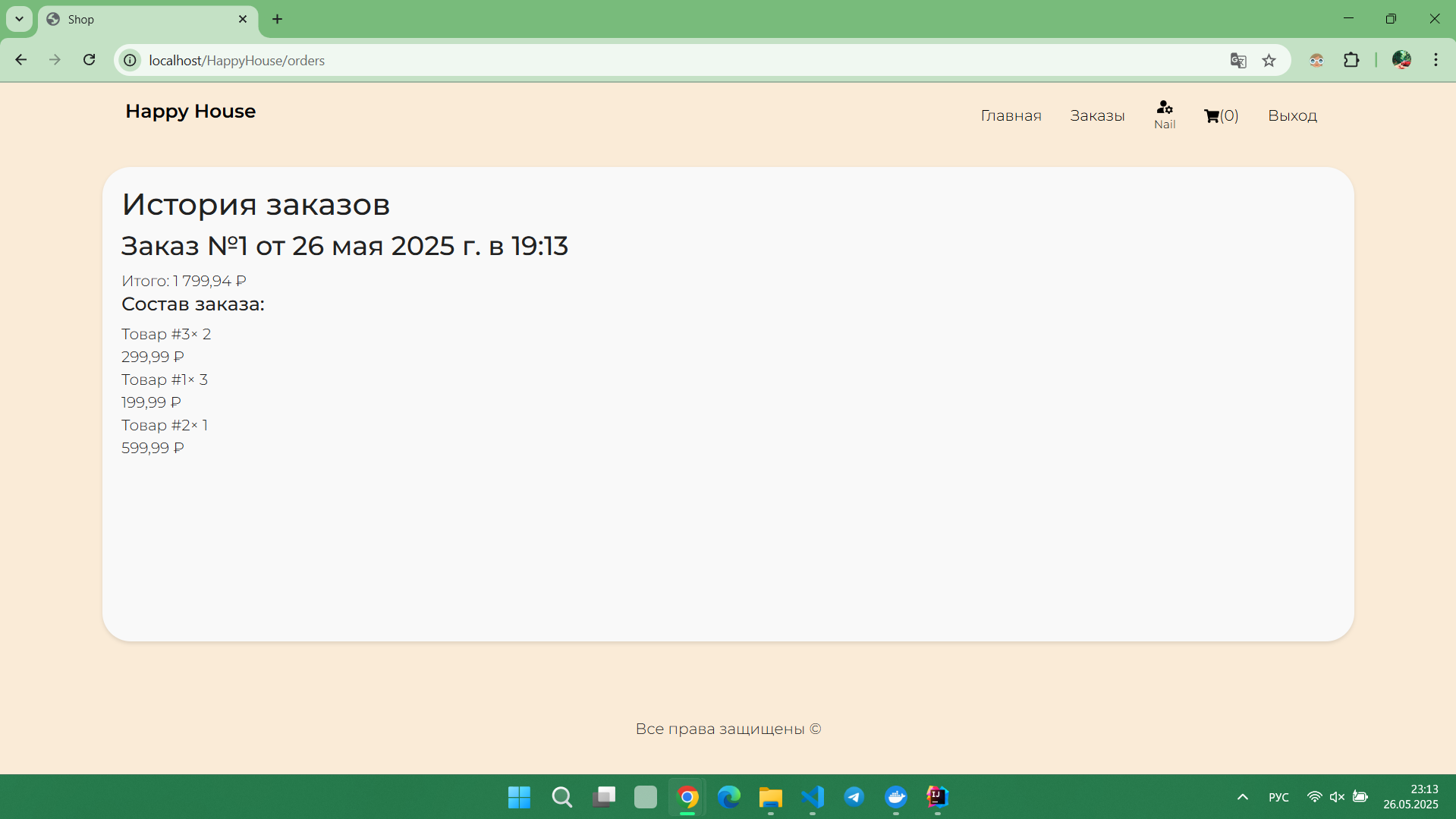
1. Также можно добавить товар в корзину на странице товара, нажав на кнопку “добавить в корзину”, после этого кнопка станет красной и изменить надпись на “удалить из корзины”:

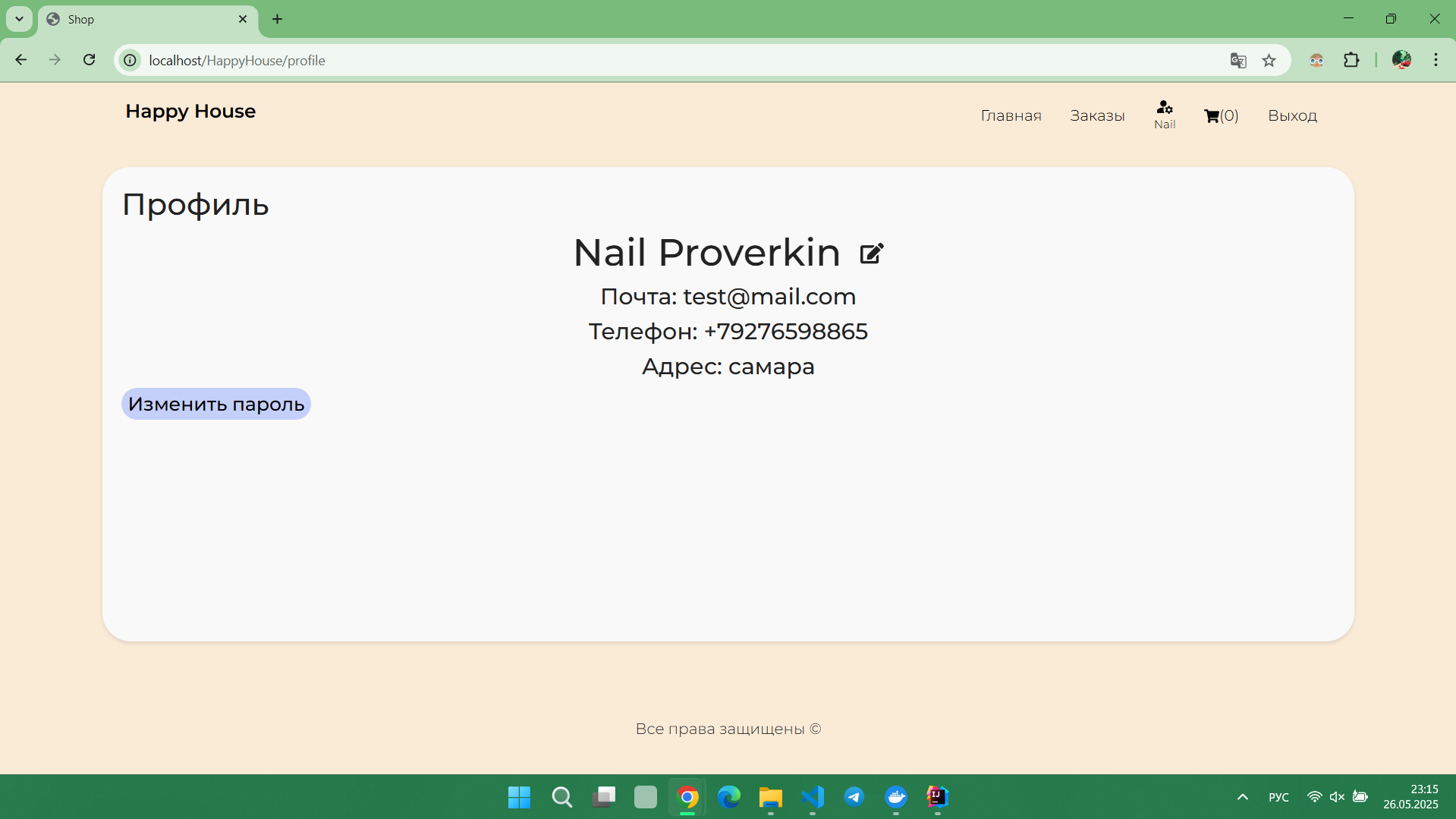


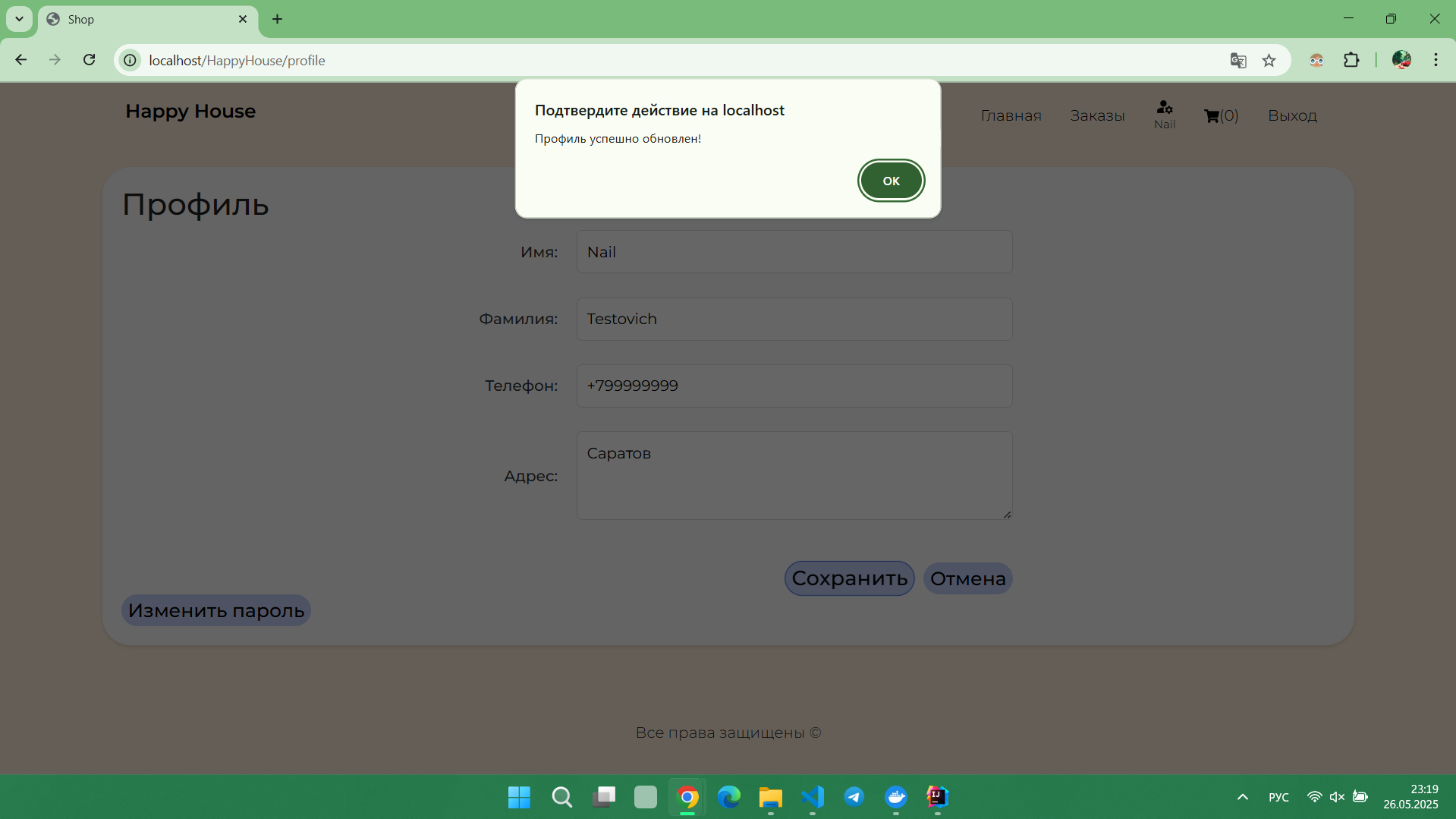
1. В корзине можно изменять количество товаров, либо удалить товар вовсе. При нажатии на кнопку “Оформить заказ” товары в корзине стираются, а заказ добавляется в БД:



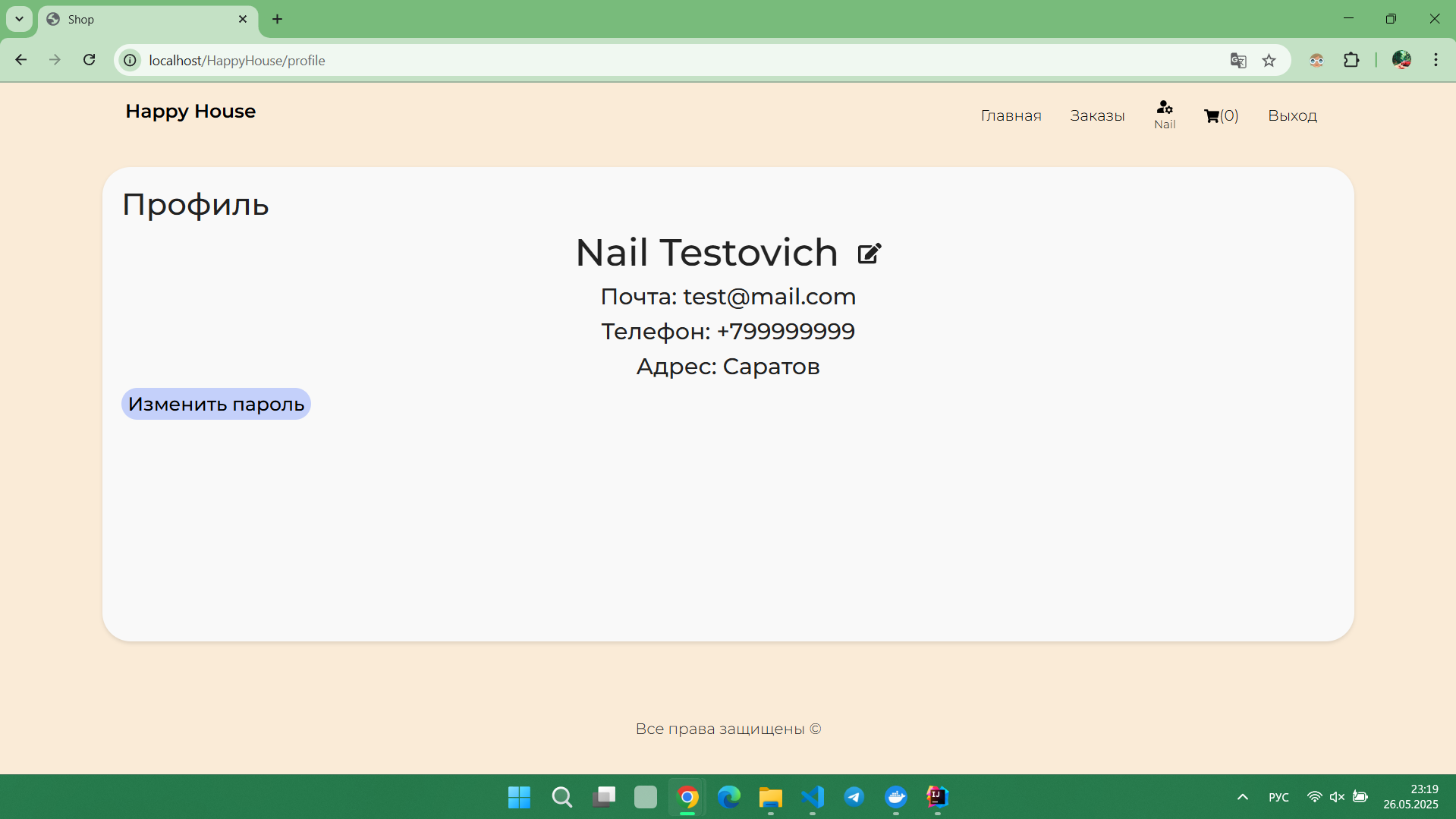
1. Теперь перейдем на страницу заказов:



На странице профиля можно изменить пароль либо другие данные пользователя, нажав на “изменить пароль” или на кнопку редактирования (в виде ручки с листочком):  




1. После, данные сразу же отображаются обновленными:



1. При нажатии на кнопку “выход” срабатывает функция, стирающая данные из localstorage(userId, access/refresh токены). И перед пользователем открывается страница входа.