



**Софийски университет „Св. Климент Охридски“**

**Факултет по математика и информатика**

**Курсов проект по  
Системи за е-бизнес  
зимен семестър 2025/2026**

**Проект:  
Онлайн магазин за спортни стоки**

Изготвил:

Владимир Григоров, 9MI8000007

# 1. Обща информация

**SportsGearStore** е уеб-базирана информационна система за **магазин за спортни стоки**, разработена като курсов проект по “Системи за е-бизнес”.

Системата реализира:

- **Каталог** от спортни продукти (по категории / отдели).
- **Търсене** в каталога.
- **Потребителски акаунти** (регистрация, вход, изход).
- **Кошница (Cart) и поръчки (Orders)**.
- **Ролева логика**: потребител / администратор.
- **Персонализация / препоръки** чрез въпросник и тагове (Tag-based recommendation).
- **Симулирано плащане** чрез Stripe Checkout в **Test Mode** (външна платежна система, без реални транзакции).

## 2. Използвани технологии

### Backend

- **ASP.NET Core Web API (.NET 8)**
- **Entity Framework Core (Code First)**
- **JWT Authentication** (token-based)
- **Swagger (OpenAPI)** за тест и документация на endpoints

### Database

- **PostgreSQL**
- EF Core миграции за създаване/обновяване на schema

### Frontend

- **Vite** (dev server + build система)
- **TypeScript** + стандартен DOM подход

- Fetch API за комуникация с бекенда
- LocalStorage за съхранение на JWT token + isAdmin

## Външни системи

- **Stripe Checkout (Test Mode)** – симулирано плащане/checkout

## 3. Архитектура на системата

Системата е разделена на 2 основни части:

### 1. Backend (REST API)

- бизнес логика
- достъп до БД
- аутентикация и авторизация
- CRUD операции (вкл. админ функции)

### 2. Frontend

- визуализация на каталог, кошница, поръчки
- login/register
- admin панел (само за админ)
- въпросник и препоръки

## Комуникация

Frontend → Backend чрез HTTP заявки:

- `Authorization: Bearer <JWT_TOKEN>` за защитени endpoints.

## 4. Бизнес роли и права

### 4.1. Потребител (User)

Може да:

- се регистрира
- логва/излиза
- разглежда продукти
- търси продукти
- добавя/маха продукти в кошница
- прави поръчка
- вижда своите поръчки
- попълва въпросник за препоръки
- вижда recommended продукти

### 4.2. Администратор (Admin)

Потребител с поле `IsAdmin = true`.

Може да:

- вижда **Admin Panel** във фронтенда
- създава/трие продукти (CRUD)
- създава/трие категории (CRUD)

## 5. База данни (Модел и таблици)

### 5.1 Основни таблици

#### Users

Съхранява профили за вход в системата.

- Id
- Username
- Useremail
- Password (SHA256 hash)
- FirstName
- LastName
- IsAdmin

#### Departments

Отдели/структура на фирмата

- Id
- Name

#### Categories

Категории на продукти

- Id
- Name

#### Products

Каталог на стоките.

- Id
- CategoryId

- Name
- Description
- Price
- ImageUrl (път до изображение)

## 6. Персонализация: Tags + Questionnaire

### 6.1 Таблица Tags

- Id
- Name (напр. weight\_loss, muscle\_gain, cardio, strength...)

### 6.2 Свързващи таблици

#### ProductTag

Много-към-много:

- ProductId
- TagId

#### UserProfile

Данни от въпросника:

- Id
- UserId
- Age
- Weight
- Height

#### UserProfileTag

Тагове избрани/изчислени от въпросника:

- `UserProfileId`
- `TagId`

### 6.3 Логика за препоръки

Системата сравнява таговете на потребителя с таговете на продуктите и връща топ N продукта, сортирани по съвпадения (`MatchScore`).

## 7. Кошница (Cart) и Поръчки (Orders)

### 7.1 Cart

Кошницата е обвързана с конкретен user (чрез JWT). Поддържа:

- Добавяне продукт
- Премахване продукт
- Актуализация количество
- Визуализация на съдържанието

### 7.2 Orders

Поръчката се създава след checkout/success.

Функции:

- Create order (от текущата кошница)
- List my orders
- Order details (по желание)

## 8. Платежна система (Stripe)

За курсовия проект е реализирано **симулирано плащане** чрез Stripe Checkout:

- Backend endpoint `POST /api/checkout/create-session` създава Stripe Session.
- Frontend redirect-ва към Stripe Checkout.
- След “Pay” (test карта) Stripe връща към `checkout-success/index.html`.
- Оттам се създава order и се изчиства cart.

Stripe работи в **Test Mode** (без истински пари).

## 9. Seed на данни

При първо стартиране се seed-ват:

- Departments (2–3)
- Categories (2–3 на отдел)
- Products (поне 8–10 на категория; в системата има 72 продукта)
- Tags (11 тага)
- Admin user (с `IsAdmin=true`)

## 10. Инсталация и стартиране

### 10.1 Backend

1. Настрой `connection string` в `appsettings.Development.json`
2. Миграции:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

3. Стартирай API:

```
dotnet run
```

### 10.2 Frontend

### 1. Инсталирай зависимости:

```
npm install
```

### 2. Стартирай:

```
npm run dev
```

## 11. Основни API endpoints

### Auth

- POST /api/auth/register
- POST /api/auth/login

### Products

- GET /api/products
- GET /api/products/{id}
- GET /api/products/search?name=...
- POST /api/products (admin)
- PUT /api/products/{id} (admin)
- DELETE /api/products/{id} (admin)

### Categories

- GET /api/categories
- GET /api/categories/{id}
- POST /api/categories (admin)
- DELETE /api/categories/{id} (admin)

### Tags

- GET /api/tags

- `POST /api/product-tags?productId=&tagId= (admin)`
- `DELETE /api/product-tags?productId=&tagId= (admin)`

## **Questionnaire / UserProfile**

- `GET /api/questionnaire`
- `POST /api/user-profile`

## **Recommendation**

- `GET /api/products/recommended (authorize)`

## **Cart**

- `GET /api/cart`
- `POST /api/cart/add`
- `PUT /api/cart/update`
- `DELETE /api/cart/remove`

## **Orders**

- `POST /api/orders`
- `GET /api/orders`

## **Checkout (Stripe)**

- `POST /api/checkout/create-session`

## 12. Use Case секция (Сценарии)

### UC-01: Регистрация на потребител

**Актьор:** Гост (не логнат потребител)

**Цел:** Създаване на нов акаунт.

**Предусловия:** Няма активен login.

**Основен поток:**

1. Потребителят отваря страницата за регистрация.
2. Попълва: username, email, password, firstName, lastName.
3. Frontend изпраща `POST /api/auth/register`.
4. Backend валидира дали username/email са свободни.
5. Backend записва потребителя (паролата се хешира).
6. Frontend показва успешно съобщение и пренасочва към login.

**Алтернативи:**

- Ако email вече съществува → backend връща 409 (Conflict)
- Ако username вече съществува → backend връща 409 (Conflict)

### UC-02: Вход (Login)

**Актьор:** Потребител

**Цел:** Получаване на JWT token и достъп до защитени ресурси.

**Основен поток:**

1. Потребителят въвежда username и password.
2. Frontend изпраща `POST /api/auth/login`.

3. Backend валидира паролата.
4. Backend генерира JWT token и връща `{ token, isAdmin }`.
5. Frontend записва token и isAdmin в LocalStorage.
6. Потребителят се пренасочва към Home.

**Алтернативи:**

- грешна парола или username → 401 Unauthorized

## UC-03: Разглеждане на каталог

**Актьор:** Логнат потребител

**Цел:** Вижда списък с продукти.

**Основен поток:**

1. Frontend извиква `GET /api/products`.
2. Backend връща списък с продукти.
3. Frontend визуализира продуктите.

## UC-04: Търсене на продукт

**Актьор:** Логнат потребител

**Цел:** Търсене по име.

**Основен поток:**

1. Потребителят пише в search полето.
2. Frontend извиква `GET /api/products/search?name=...`
3. Backend връща филтриран списък.

4. Frontend показва резултатите.

## UC-05: Управление на кошница

**Актьор:** Логнат потребител

**Цел:** Добавяне/промяна на количества/премахване.

**Основен поток:**

1. Потребителят натиска Add to Cart.
2. Frontend прави `POST /api/cart/add`.
3. Backend добавя/увеличава quantity.
4. Потребителят отваря cart page.
5. `GET /api/cart` връща текущите items.
6. Потребителят променя quantity (+/-).
7. Frontend праща `PUT /api/cart/update` или `DELETE /api/cart/remove`.
8. Backend обновява кошницата.

## UC-06: Checkout (симулирано плащане)

**Актьор:** Логнат потребител

**Цел:** Симулирано плащане чрез Stripe и приключване на поръчка.

**Основен поток:**

1. Потребителят натиска "Proceed to Checkout".
2. Frontend вика `POST /api/checkout/create-session`.
3. Backend създава Stripe Checkout Session и връща url.
4. Frontend redirect-ва към Stripe Checkout.

5. Потребителят използва test карта (4242...).
6. Stripe връща към `checkout-success` страница.
7. Frontend извиква `POST /api/orders`.
8. Backend създава `order` и изчиства `cart`.
9. Потребителят вижда потвърждение.

## UC-07: Преглед на поръчки

**Актьор:** Логнат потребител

**Цел:** Вижда списък със свои поръчки.

**Основен поток:**

1. Frontend извиква `GET /api/orders`.
2. Backend връща поръчките на текущия `user`.
3. Frontend ги показва.

## UC-08: Попълване на въпросник (персонализация)

**Актьор:** Логнат потребител

**Цел:** Запис на профил + тагове за препоръки.

**Основен поток:**

1. Потребителят отваря `Questionnaire`.
2. Отговаря на въпроси.
3. Frontend изпраща резултата към `backend`.
4. Backend записва/обновява `UserProfile`.
5. Backend записва `UserProfileTags` според отговорите.

## UC-09: Препоръчани продукти

**Актьор:** Логнат потребител

**Цел:** Получава персонализирани продукти.

**Основен поток:**

1. Frontend извиква GET /api/products/recommended.
2. Backend намира тагове на потребителя.
3. Backend намира продуктите с най-много съвпадащи тагове.
4. Връща топ N резултати.
5. Frontend ги показва на Home.

## UC-10: Admin – управление на продукти (CRUD)

**Актьор:** Администратор

**Цел:** Добавяне/триене/редакция.

**Предусловия:** User е логнат и IsAdmin=true.

**Основен поток:**

1. Админът отваря Admin Panel.
2. Вижда списък с продукти (GET /api/products).
3. Създава нов продукт (POST /api/products).
4. Трие продукт (DELETE /api/products/{id}).

**Алтернативи:**

- ако не е админ → 403 Forbid

## UC-11: Admin – upload на снимка

**Актьор:** Администратор

**Цел:** Качване на product image.

**Основен поток:**

1. Админът избира файл от компютъра.
2. Frontend изпраща `POST /api/uploads/product-image (multipart/form-data)`.
3. Backend записва файла в `wwwroot/images/products/`.
4. Връща `imageUrl`.
5. Създаденият продукт използва този `imageUrl`.

## 13. Заключение

SportsGearStore реализира основните функционални модули на е-бизнес система: каталог, администриране, потребителски акаунти, кошница, поръчки, препоръки и интеграция с външна платежна услуга (Stripe). Проектът използва съвременна архитектура (REST API + frontend клиент), EF Core Code First и PostgreSQL, като е разработен с фокус върху покриване на критериите на курсовия проект.