**2. Fejlesztői dokumentáció  
2.1 Áttekintés**

A **Producthor** egy oktatási/portfólió célú, tipikus webáruház backend (REST API) Java 17 + Spring Boot ökoszisztémán. A rendszer entitásai: **User**, **Product**, **Category**, **Order**, **OrderItem**, **ShippingData**. A szolgáltatás JWT alapú stateless authentikációt használ, admin/user szerepekkel. A képfeltöltés lokális fájlrendszerre történik, a feltöltött állományok HTTP-n publikálhatók.

A dokumentum célja, hogy szakmai mélységben bemutassa a kivitelezést: használt technológiák, környezet, adatmodell, fő folyamatok/algoritmusok, biztonság, konfiguráció, futtatás, és bővítési lehetőségek.

**2.2 Felhasznált technológiák**

* **Java 17** – LTS, modern nyelvi funkciók (records helyett most DTO-k/Lombok), jó ökoszisztéma.
* **Spring Boot (3.x)** – gyors indítás, autokonfiguráció, Actuator-kompatibilitás (opcionális).
* **Spring Web / MVC** – REST kontrollerek (@RestController), request mappingek.
* **Spring Data JPA / Hibernate** – ORM, deklaratív repository-k, @Transactional.
* **Spring Security** – stateless biztonsági lánc, role alapú jogosultság, JWT integráció.
* **JWT (jjwt)** – aláírt hozzáférési tokenek.
* **PostgreSQL 16** – ACID RDBMS, bő funkciók, megbízható tranzakciókezelés.
* **Maven** – build, dependency management, test futtatás.
* **Lombok** – boilerplate csökkentése (@Getter/@Setter/@Builder/...).
* **MockMvc / Spring Test** – kontrollerek és szolgáltatások tesztjeihez.

**2.3 Fejlesztőkörnyezet**

**2.3.1 Hardver**

* Fejlesztői gép: átlagos modern laptop/desktop (8–16 GB RAM, SSD ajánlott).
* Indoklás: Spring Boot + PostgreSQL fejlesztői környezethez ez teljesen elegendő; a tesztek és lokális DB futtatása kényelmes.

**2.3.2 Szoftver**

* **Java 17 JDK**
* **Docker** (opcionális) PostgreSQL futtatására:  
  docker run --name producthor-postgres -e POSTGRES\_DB=producthor -e POSTGRES\_USER=producthor -e POSTGRES\_PASSWORD=producthor -p 5433:5432 -v producthor\_pg\_data:/var/lib/postgresql/data -d postgres:16
* **IDE**: IntelliJ IDEA / VS Code Java / Eclipse – tetszőleges.
* **Maven**: mvn clean verify / mvn spring-boot:run
* **PostgreSQL kliens** (psql/DBeaver) – sémák/lekérdezések ellenőrzésére.

**2.4 Konfiguráció**

application.properties (kiemelt kulcsok):

spring.application.name=Producthor

server.port=8080

spring.datasource.url=jdbc:postgresql://localhost:5433/producthor

spring.datasource.username=producthor

spring.datasource.password=producthor

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.open-in-view=false

security.jwt.secret=Bcx56JFth71jkl20Bcx56JFth71jkl20

security.jwt.expiration-ms=3600000

spring.servlet.multipart.max-file-size=5MB

spring.servlet.multipart.max-request-size=5MB

app.upload.dir=uploads

app.public.base-url=/files

* **open-in-view=false**: N+1 és lazy init problémák csökkentése service/DAO rétegben megoldott adat-hozzáférést feltételez.
* **CORS**: WebCorsConfig korlátozott eredetre (Angular FE: http://localhost:4200).

**2.5 Architektúra és rétegek**

**Hagyományos, rétegzett felépítés**:

* **Controller**: HTTP végpontok, REST kontraktusok (DTO be/ki).
* **Service**: üzleti logika, tranzakciókezelés, érvényesítések, map-pelések meghívása.
* **Repository**: JPA alapú DAO, deklaratív queryk (pl. findByAvailableTrue()).
* **Mapper**: entitás ↔ DAO/DTO konverziók, update/merge logika.
* **Security**: SecurityConfig, JwtAuthenticationFilter, JwtTokenService.

Stateless API (session nélküli), minden kérés opcionálisan JWT-vel biztosítható.

**2.6 Adatmodell**

**2.6.1 Entitások és kapcsolatok (részletek)**

* **User**
  + username (unikális), password (BCrypt), name, isAdmin (ROLE\_ADMIN/USER).
  + **1–1** *ShippingData* (opcionális) – a felhasználó alapértelmezett címe.
  + **1–N** *Order* (orderHistory).
* **ShippingData**
  + Alap címmezők: postalCode, city, street, houseNumber, additionalInfo.
* **Category**
  + name (unikális), description.
  + **1–N** *Product*.
* **Product**
  + name, description, price, imageUrl, available.
  + **N–1** *Category* (nullable).
  + **ElementCollection** specifications: Map<String,String> külön táblában.
* **Order**
  + **N–1** *User* (nullable – vendég rendelés támogatott).
  + **N–1** *ShippingData* (nem unique, több order hivatkozhat ugyanarra a címre).
  + **1–N** *OrderItem* (cascade + orphanRemoval).
  + totalGross számított mező (service réteg állítja).
* **OrderItem**
  + **N–1** *Order*, **N–1** *Product*.
  + quantity, unitPriceAtPurchase (pillanatnyi ár snapshot).

**Megjegyzés a sémáról**: a korábbi **unique** constraint az orders.shipping\_data\_id-n eltávolításra került, így több rendelés is hivatkozhat azonos címre (reorder, ismétlődő használat).

**2.6.2 DAO-k és DTO-k**

* **DAO**: kliensnek visszaadott, serializálható “view model” (pl. ProductDao, OrderDao).
* **DTO**: bemeneti adatok (pl. ProductDto, OrderDto, AuthenticationDto).

**Mapper-ek** (pl. ProductMapper, OrderMapper, stb.) végzik az oda-vissza átalakítást és “null-skip” frissítést (updateEntityFromDto: csak nem-null mezőket ír felül).

**2.7 Biztonság**

**2.7.1 HTTP végpontok és jogosultság**

* "/api/\*/public/\*\*": nyilvános (pl. login, regisztráció, termék listázás).
* "/api/\*/private/\*\*": authentikáció szükséges (JWT).
* "/api/\*/admin/\*\*": ROLE\_ADMIN szükséges.
* GET /files/\*\* nyilvános (feltöltött képek kiszolgálása).

**2.7.2 JWT lánc**

* **JwtAuthenticationFilter**: kibontja a Bearer tokent, ellenőrzi JwtTokenService-szel, a SecurityContext-be teszi a felhasználót.
* **JwtTokenService**: generálás (sub, roles, exp), ellenőrzés (aláírás, lejárat, subject).
* **Stateless**: nincs HTTP session; minden kérés hordozza a tokent.

**2.8 Fő folyamatok / algoritmusok**

**2.8.1 Autentikáció / bejelentkezés**

1. **Login** (/api/auth/public/login): AuthenticationManager ellenőrzi user/pass-t.
2. Siker esetén **JWT** készül a felhasználó szerepeivel; response: token + meta.

**2.8.2 Képfeltöltés**

* **FileStorageService.store**:
  + Content-Type ellenőrzés (jpeg/png/webp).
  + Biztonságos fájlnév (UUID + kiterjesztés).
  + app.upload.dir alá mentés, majd **publikus URL** visszaadása (/files/{uuid.ext}).
* **StaticResourceConfig**: a uploads mappát HTTP-n publikálja.

**2.8.3 Termék létrehozás / módosítás**

* ProductService.create/update:
  + Kötelező validációk (név, ár).
  + Kategória feloldása (ha adott).
  + Specifikációk teljes felülírása update-nél, ha DTO-ban jöttek.
  + available default: true (ha hiányzik).
  + Persistálás, egyszerű BaseResponse.

**2.8.4 Rendelés létrehozása**

* OrderService.create(OrderDto):
  + Üres lista → ERROR.
  + Shipping kiválasztás:
    - shippingDataId → feloldás; ha nincs, ERROR.
    - ellenkező eset: inline ShippingDataDto → entitássá, mentés.
  + Order összeállítása:
    - minden tételnél Product feloldás (ERROR, ha nincs),
    - unitPriceAtPurchase a termék aktuális ára,
    - quantity ≥ 1 kötelező.
  + totalGross = Σ(unitPriceAtPurchase \* quantity).
  + Mentés, visszaadott OrderDao (items, shipping, total, username).

**Megjegyzés**: A vendég rendelés (user = null) támogatott, authentikált user esetén a username is a DAO része (mapper tölti ki).

**2.8.5 Felhasználói profil frissítése**

* UserService.update(UserDto):
  + Authenticated user feloldása.
  + name frissítés.
  + shippingData:
    - ha hiányzott → új ShippingData entitásból felvétele,
    - ha volt → mezőszintű frissítés (null-skip).
  + Mentés, OK/ERROR visszajelzés.

**2.8.6 Kategória törlése**

* CategoryService.delete:
  + Létezés ellenőrzés,
  + **ProductRepository.clearCategoryFromProducts** (FK megszüntetése a termékeken),
  + Kategória törlése.

**2.9 REST API kivonat (minták)**

* **Auth**
  + POST /api/auth/public/register – { username, password }
  + POST /api/auth/public/login – → { token, isAdmin }
* **Product**
  + POST /api/product/admin/create – Admin
  + PUT /api/product/admin/update/{id} – Admin
  + GET /api/product/public/get?id=...
  + GET /api/product/public/all
  + GET /api/product/public/availables
  + GET /api/product/public/byCategory?categoryId=...
* **Category**
  + GET /api/category/public/getAll
  + POST /api/category/admin/create / PUT /admin/update/{id} / DELETE /admin/delete/{id} – Admin
* **Order**
  + POST /api/order/public/create – vendég is rendelhet (shipping kötelező)
* **User**
  + POST /api/user/private/update – profil mentése
  + GET /api/user/private/get – bejelentkezett user + rendelési előzmények
  + GET /api/user/admin/all – Admin
  + DELETE /api/user/admin/delete/{id} – Admin
* **File**
  + POST /api/file/admin/upload – Admin, 5 MB limit, url-t ad vissza

**2.10 Futtatás & fejlesztői életciklus**

**2.10.1 Adatbázis**

* Dockerrel indítva (PostgreSQL 16): lásd fent.
* Alap beállítások: producthor/producthor, DB: producthor, port: 5433.

**2.10.2 Backend indítása**

mvn clean spring-boot:run

# vagy

mvn clean package && java -jar target/\*.jar

**2.10.3 Alap seed / manuális ellenőrzés**

* **Regisztráció**: POST /api/auth/public/register
* **Login** → JWT
* **Admin tevékenységek**: állíts be egy admin usert (DB-ben is\_admin = true), majd termék/kategória létrehozás.

**2.11 Validáció, hibakezelés**

* Input-validáció a service rétegben (null/blank/ár > 0, stb.).
* Hibák esetén **BaseResponse("ERROR", "...")** mint egységes visszajelzés.
* Az API design a kliens egyszerű kezelését célozza: nincsenek kivételdobások választestben; strukturált code/message érkezik.

**2.12 Teljesítmény, skálázhatóság**

* **Stateless** API → horizontális skálázásra alkalmas.
* **DB indexek**: idegen kulcsokon és gyakori keresési mezőkön (pl. idx\_orders\_user\_id).
* **Képfájlok**: lokális fájlrendszer – kis projekthez megfelelő; éles környezetben CDN/objekttár (S3, GCS) javasolt.
* **Specifikációk**: @ElementCollection(EAGER) – kisméretű mapekhez kényelmes; ha nagyra bővülne, külön entitásra érdemes váltani LAZY fetch-csel.

**2.13 Biztonsági megfontolások**

* **Jelszavak**: BCrypt.
* **JWT**: titok kulcs .properties-ben (demo), éles környezetben **vault / env**.
* **CORS**: célzott eredetek, fejlesztői állapotban localhost:4200.
* **Képfeltöltés**: content-type ellenőrzés, kiterjesztés-becslés, path traversal védelem, random fájlnév.
* **Open redirect / CSRF**: stateless API, CSRF kikapcsolva (cookie nélküli auth), httpBasic tiltva.

**2.14 Bővítési lehetőségek**

* **Rendelés státuszok** (PLACED, PAID, SHIPPED, CANCELLED) + események.
* **Kedvezmények / kuponok**, szállítási díj kalkuláció.
* **Képátméretezés / thumbnail** generálás feltöltéskor.
* **Kereső / szűrő** a backendben (név, kategória, available, árintervallum) — paginált.
* **Audit log**: ki mit módosított (Spring Data Envers, saját audit trail).
* **Email/SMS értesítés** rendelés leadásakor (outbox pattern).
* **Specifikációk** normalizálása (külön táblák, típusok, validációk).

**2.15 Példák (curl)**

**Login:**

curl -X POST http://localhost:8080/api/auth/public/login \

-H "Content-Type: application/json" \

-d '{"username":"john","password":"pw"}'

**Új termék (admin JWT szükséges):**

curl -X POST http://localhost:8080/api/product/admin/create \

-H "Authorization: Bearer <JWT>" \

-H "Content-Type: application/json" \

-d '{"name":"Phone","price":199990,"available":true}'

**Rendelés (vendég):**

curl -X POST http://localhost:8080/api/order/public/create \

-H "Content-Type: application/json" \

-d '{

"items":[{"productId":1,"quantity":2}],

"shippingData":{"postalCode":"4400","city":"Nyh","street":"U","houseNumber":"1","additionalInfo":""}

}'

**2.16 Tesztelhetőség és tesztstratégia röviden**

* **Unit**: mapper-ek, szolgáltatások (mockolt repo-kkal).
* **Slice / MVC**: kontrollerek @WebMvcTest + MockMvc.
* **Integrációs**: @SpringBootTest valós JPA réteggel és in-memory/lokális PG-vel.
* **Represszió**: korábban javított hibákra célzott teszt (pl. shippingData több rendeléshez).

(A részletes teszt-dokumentáció külön anyagban található, már elkészült.)

**2.17 Üzemeltetés, loggolás**

* **Spring Boot logging** (konzol), fejlesztéskor SQL log látható (show-sql=true).
* Élesben ajánlott: strukturált log (JSON), request-id, külön logger kategóriák (security, persistence).

**2.18 Ismert döntések**

* **ShippingData** megosztható több rendelés között → **ManyToOne** Order → ShippingData.
* **Open-in-view=false**: minden adatelérés service rétegben történik (lezárt session mellett nem renderelünk lazy kollekciót a kontrollerekben).
* **Specifikációk** mapként: gyors iterációhoz ideális; ha üzleti igény nő, normalizáljuk.

**2.19 Összegzés**

A Producthor backend tiszta, rétegezett Spring architektúrát követ, jól körülhatárolt felelősségi körökkel és egyszerű, ugyanakkor bővíthető domain modellel. A stateless JWT-biztonság, a szabványos JPA, a szelektív CORS, valamint a fájl-tárolási mechanizmus fejlesztőbaráttá és könnyen üzemeltethetővé teszi a rendszert. A megoldás jól illeszthető a már elkészült Angular front-endhez, és életszerű bővítési irányokkal rendelkezik a későbbi fejlesztési ciklusokra.