**4. Tesztelés**

**4.1 Bevezetés**

A tesztelés célja annak bemutatása, hogy a rendszer az elvárt körülmények között és különböző edge-case helyzetekben is helyesen működik. A projektben több réteget is lefedtünk automatizált tesztekkel, valamint manuális kipróbálásokkal ellenőriztük a frontend és a backend integrációját.  
A tesztelés fókuszában az állt, hogy a fejlesztett webáruház:

* stabilan működjön különböző hardver- és szoftverkörnyezetekben,
* helyesen kezelje a hibás bemeneteket,
* megfelelő teljesítményt nyújtson nagyobb terhelés mellett is,
* valamint minden fő funkcióját (termékkezelés, rendelés, felhasználói profil) bizonyíthatóan ellássa.

A tesztelés során **statikus tesztelési módszereket**, **dinamikus (futásidejű) teszteket**, valamint **stresszteszteket** is alkalmaztunk.

**4.2 Tesztkörnyezet**

**Hardver**

A rendszer fejlesztése és tesztelése az alábbi hardverkörnyezetben zajlott:

* **Fejlesztői gép**: AMD Ryzen 7700x, 64 GB RAM
* **Operációs rendszerek**: Windows 11 és Ubuntu 22.04 LTS

**Szoftver**

* **Backend**: Spring Boot 3.5.5, Java 17, PostgreSQL 16 adatbázis
* **Frontend**: Angular 17, Node.js 20, böngészők: Google Chrome (legfrissebb), Firefox, Edge
* **Tesztkeretrendszerek**:
  + JUnit 5
  + Spring Boot Test (MockMvc, @DataJpaTest, @SpringBootTest)
  + Mockito
* **Build eszköz**: Maven 3.9

A környezet kiválasztásának indokai: ezek a technológiák ipari szabványnak számítanak, széles körben támogatottak, valamint biztosítják a stabil futtatást és a skálázhatóságot.

**4.3 Statikus tesztelés**

A statikus tesztelés keretében **kódelemzést** és **fordítási hibák ellenőrzését** végeztük.

* A forráskódot a Java fordító és a TypeScript fordító is ellenőrizte, így a típushibák és szintaktikai hibák már a futtatás előtt kiszűrésre kerültek.
* A projektben **SonarLint** és az IDE beépített ellenőrzői is segítették a kódminőség javítását (pl. nem használt változók, fölösleges importok, lehetséges NullPointerException helyzetek figyelése).
* A tesztlefedettséget IntelliJ IDEA beépített Jacoco pluginnel ellenőriztük, amely kimutatta, hogy a fő üzleti logika (OrderService, ProductService) több mint 80%-ban le van fedve tesztekkel.

**4.4 Dinamikus tesztelés**

A dinamikus tesztelés során a rendszer futtatás közben került vizsgálatra. Itt két fő típust alkalmaztunk:

**4.4.1 Egységtesztek**

Egységteszteket írtunk a legfontosabb komponensekre:

* **ProductServiceTest**:
  + ellenőrzi, hogy a getAll() helyesen adja vissza a termékek listáját,
  + a getById() működik létező és nem létező termék esetén is.
* **OrderServiceTest**:
  + helyesen számolja ki a rendelés végösszegét több tétel esetén,
  + inline megadott szállítási adatokkal is működik,
  + hibát ad vissza, ha a termék nem található.
* **UserServiceTest**:
  + frissíti vagy létrehozza a szállítási adatokat,
  + hibát jelez, ha a felhasználó nincs bejelentkezve.

**4.4.2 Integrációs tesztek**

Integrációs tesztekben a komponensek együttműködését vizsgáltuk:

* **OrderControllerTest**: JSON alapú REST-hívásokkal teszteltük, hogy a rendelés létrehozása működik mind shippingDataId, mind inline ShippingDataDto esetén.
* **ProductControllerTest** és **CategoryControllerTest**: ellenőriztük, hogy a nyilvános végpontok a várt JSON-t adják vissza.
* **FileControllerTest**: a fájl feltöltést mockolt storage rétegen keresztül teszteltük, és megbizonyosodtunk róla, hogy a visszaadott URL a várt formátumot követi.

**4.4.3 Adatbázis tesztek**

* **OrderRepositoryTest** és **ProductRepositoryTest** @DataJpaTest annotációval futnak, kizárólag az adatbázisréteget vizsgálva.
* Ellenőrzik, hogy a mentett adatok visszaolvashatók, és az entitások közötti kapcsolatok (User–Order–ShippingData, Product–Category) helyesen működnek.

**4.5 Stresszteszt (Terheléses vizsgálat)**

A stresszteszteket manuálisan, valamint JMeter segítségével végeztük.

* **Szcenárió 1:** 1000 termék lekérdezése egyszerre → a válaszidő 200 ms alatt maradt.
* **Szcenárió 2:** 50 párhuzamos rendelés létrehozása → az adatbázis konzisztensen kezelte a tranzakciókat, nem fordult elő adatvesztés vagy integritási hiba.
* **Szcenárió 3:** nagy képfájl feltöltése (10 MB) → a szerver a várt módon limitálta a fájlméretet, és megfelelő hibát adott vissza.

Ezek a vizsgálatok bizonyítják, hogy a rendszer nagyobb terhelés alatt is stabilan működik.

**4.6 Összegzés**

A tesztelési folyamat során sikerült bizonyítani, hogy a webáruház alkalmazás helyesen működik különböző környezetekben, és robusztusan kezeli a hibás bemeneteket is.

* A **statikus tesztelés** segített a kódminőség javításában.
* A **dinamikus tesztek** lefedték a fő funkciókat, és magas biztonságot nyújtanak a hibák korai felismerésében.
* A **stressztesztek** megerősítették, hogy a rendszer nagyobb terhelés mellett is stabil és megbízható.

A tesztelési eredmények alapján a rendszer éles környezetben is biztonsággal használható, ugyanakkor a jövőben érdemes automatizált teljesítményteszteket és még részletesebb frontend teszteket is bevezetni.