Zeneszer Webshop Rendszerdokumentáció

[**1. Bevezetés 4**](#_jeoobffx7a1w)

[**2. Technológiák részletes ismertetése 4**](#_7qg3n154th9)

[Spring Boot: 4](#_t4mozftasmru)

[Fő jellemzők 5](#_r0lofm3airtj)

[Használati területek 5](#_28thzbleb2ls)

[Előnyök és hátrányok 5](#_t4mozftasmru)

[Összegzés 6](#_u61m3v80wfe4)

[Angular: 6](#_wt5elno6jugx)

[Fő jellemzők 6](#_wt5elno6jugx)

[Használati területek 7](#_wt5elno6jugx)

[Előnyök és hátrányok 7](#_wt5elno6jugx)

[Összegzés 7](#_yqvw60cghukc)

[Bootstrap: 7](#_e89syni0dop1)

[Fő jellemzők 8](#_fsiif083zein)

[Használati területek 8](#_fsiif083zein)

[Előnyök és hátrányok 8](#_fsiif083zein)

[Összegzés 9](#_fsiif083zein)

[OpenAPI 9](#_i9kqnzpprk16)

[Fő jellemzők 9](#_fsiif083zein)

[Használati területek 10](#_fsiif083zein)

[Előnyök és hátrányok 10](#_fsiif083zein)

[Összegzés 10](#_ghgqj1r2z5t5)

[**Zeneszer Telepítési Útmutató 10**](#_l1aqh4b02cue)

[Előfeltételek 10](#_otzhk52a8maz)

[1. lépés: JDK 17 telepítése 11](#_4jx5njtd4t46)

[2. Maven telepítése 12](#_t9sunosqsdi1)

[Maven letöltése: 12](#_j124mi271g1z)

[Kicsomagolás és beállítás: 12](#_dtq5wm55msw1)

[Windows: 12](#_q4st4h6ps2k3)

[Linux/macOS: 12](#_9ytm4xwr71ph)

[3. lépés: Node.js és npm telepítése 12](#_tr0plyr6fzuq)

[Node.js letöltése: 12](#_z1vp5ftn7n7m)

[Node.js telepítése: 12](#_yet2715l5k26)

[Telepítés ellenőrzése: 13](#_jy5iy3zesa8y)

[4. lépés: A kódtár klónozása 13](#_vidln4tagdd)

[5. lépés: Frontend függőségek telepítése 13](#_q7ooqt5nxpby)

[6. lépés: Frontend buildelése és futtatása 13](#_togiglockyu2)

[7. lépés: Backend buildelése és futtatása 14](#_g5h5561vl4fo)

[8. Swagger UI 14](#_cebx4xqgsdo1)

[3. Backend Architektúra 14](#_zawwgnrw6yn7)

[A három fő komponens 14](#_f59ax2f7etoo)

[InstrumentResource 18](#_amzdt07o07vo)

[UserResource 19](#_kow9bwvtax5e)

[Összegzés 20](#_rs8eojqhxhsx)

[3. Frontend Architektúra 20](#_6nu4ntlmfhxe)

[Áttekintés 20](#_1hdrf8u0bdix)

[Főbb Komponensek 20](#_m1ugehxkkots)

[Kezdőoldal: 20](#_c59vtld1bd71)

[Főbb jellemzők 20](#_4v5kohugticy)

[1. Üdvözlő szöveg: Nagybetűs, kiemelt címsor üdvözli a látogatókat 20](#_m1ugehxkkots)

[2. Navigációs gomb: "Explore our store" gomb, amely a hangszerlistára vezet 20](#_m1ugehxkkots)

[3. Bemutató kép: Nagy méretű, reszponzív kép egy hangszerboltról 20](#_m1ugehxkkots)

[4. Egyszerű kialakítás: Minimalista megjelenés a tartalomra fókuszálva 20](#_ckxdj3g83fj1)

[Technikai részletek 21](#_w2if7cm65b6e)

[● Komponens: HomeComponent 21](#_m1ugehxkkots)

[● Használt modulok: CommonModule (alap Angular funkciók) 21](#_m1ugehxkkots)

[● Stíluslap: Saját SCSS fájl (home.component.scss) 21](#_m1ugehxkkots)

[● Kép megjelenítés: Bootstrap img-fluid osztállyal reszponzívvá téve 21](#_m1ugehxkkots)

[● Külső hivatkozás: Kép betöltése külső forrásból (Squarespace CDN) 21](#_ozbfkvqzfm0p)

[Hangszer kezelés: 21](#_rtzmsnbcs3xx)

[InstrumentService 21](#_2y2l28vfanqb)

[Funkcionális Komponensek 21](#_3w5te4rgqha0)

[InstrumentListComponent 21](#_vnt52w3nmo8n)

[InstrumentAddComponent 21](#_snbk6d4o1r00)

[InstrumentEditComponent 21](#_8taav77quzhc)

[Technikai Jellemzők 22](#_a5kptbqvzyiq)

[Használati Folyamat 22](#_xw8s1z7tkxkj)

[**Felhasználókezelés 22**](#_9z5sox64n6gj)

[**UserService 22**](#_z83x9ugmvxtt)

[UserAddComponent 22](#_e3n1z99wvvpx)

[Lehetséges Jövőbeli Fejlesztések 23](#)

[Források: 23](#_breu8fxoda9k)

# 

# 1. Bevezetés

A dokumentáció egy teljes körű zeneszer webshop rendszert ír le, amely modern webes technológiákat alkalmaz. A rendszer célja, hogy zenei hangszerek online értékesítését megvalósítsa felhasználóbarát felületen, miközben robusztus backend szolgáltatásokat nyújt. A pre mvp (Minimum Viable Product) képes user kezelésre (regisztráció, login/logout), illetve hangszerek szerkesztésére, létrehozására, illetve, különböző nézetekben való megjelenítésére.

**Főbb jellemzők, használt technológiák:**

* **Full-stack architektúra** (Spring Boot backend + Angular frontend)
* **RESTful API** design Swagger dokumentációval
* **Java** és **Spring Boot** alapú backend
* **Maven** alapú build rendszer
* **H2** in-memory adatbázis fejlesztési környezetben
* **Reszponzív webdesign** mobil és asztali eszközökre
* **Typescript** és **Angular** alapú frontend
* **Bootstrap**
* **OpenApi**

# 2. Technológiák részletes ismertetése

## Spring Boot:

A klasszikus Spring alkalmazások fejlesztése gyakran bonyolult volt a sokféle konfiguráció, XML fájl és beállítás miatt. A Spring Boot ezt a komplexitást csökkenti úgy, hogy „convention over configuration” (azaz az alapértelmezések használata a kézi konfiguráció helyett) szemléletet alkalmaz. Ez azt jelenti, hogy az alkalmazások gyorsan, kevés beállítással is működésbe hozhatók, de természetesen testreszabhatók, ha szükséges.

### Fő jellemzők

* Automatikus konfiguráció: A Spring Boot képes felismerni, milyen függőségek vannak jelen, és ezek alapján automatikusan konfigurálja az alkalmazást.
* Beépített szerverek: Nem szükséges külön Tomcat vagy Jetty telepítése, mert a Spring Boot „önmagát futtató” alkalmazásokat hoz létre, amelyeket egyszerűen el lehet indítani.
* Spring Starter csomagok: Ezek előre összeállított Maven/Gradle függőségcsomagok, amelyek leegyszerűsítik a különböző technológiák (pl. JPA, Web, Security) integrálását.
* Egységes konfiguráció: Egy egyszerű application.properties vagy application.yml fájl segítségével lehet központilag konfigurálni az alkalmazás különböző aspektusait.
* Actuator modul: Rendszerfigyelő végpontokat biztosít az alkalmazás állapotának, metrikáinak és működésének megfigyelésére.

### Használati területek

A Spring Boot különösen népszerű a mikroszolgáltatás-alapú architektúrákban, ahol minden szolgáltatás egy különálló, könnyen telepíthető egységet képez. Emellett webes alkalmazások, REST API-k, háttérrendszerek, adatfeldolgozó alkalmazások és felhőalapú rendszerek fejlesztéséhez is széles körben alkalmazzák.

### Előnyök és hátrányok

Előnyök:

* Gyors fejlesztés és indítás
* Csökkentett konfigurációs igény
* Széles körű közösségi támogatás és dokumentáció
* Integráció a legtöbb modern Java technológiával

Hátrányok:

* Az automatikus konfiguráció néha rejtett komplexitást eredményezhet
* Nem mindig optimális nagyon kis méretű vagy egyszerű projektekhez

### Összegzés

A Spring Boot egy kiváló eszköz azok számára, akik modern, skálázható és gyorsan fejleszthető Java alkalmazásokat szeretnének készíteni. Egyszerűsége és rugalmassága miatt mára az egyik legfontosabb technológia lett a Java ökoszisztémában.

## Angular:

Az Angular egy nyílt forráskódú webalkalmazás-fejlesztő keretrendszer, amelyet a Google fejleszt és tart karban. Az Angular segítségével hatékony, jól strukturált, dinamikus, egyoldalas alkalmazások (SPA – Single Page Application) készíthetők. A jelenlegi Angular (néha Angular 2+ néven is emlegetik) nem azonos az eredeti AngularJS-sel – ez egy teljesen újraírt verzió, amely modern nyelveket és fejlesztési szemléletet alkalmaz, különösen a TypeScript nyelv használatával.

Miért használjuk az Angulart?  
 Az Angular célja, hogy egységes, moduláris és skálázható megoldást nyújtson a komplex webes felületek fejlesztésére. A keretrendszer teljes ökoszisztémát biztosít, így nem szükséges különálló könyvtárakat integrálni az alapvető funkcionalitásokhoz – például routing, űrlapkezelés, HTTP-kapcsolat vagy tesztelés.

### Fő jellemzők

* TypeScript alapú fejlesztés: A JavaScript szigorúbb, típusos változata, ami biztonságosabb és átláthatóbb kódot eredményez.
* Komponensalapú architektúra: Minden felhasználói felületi elem külön komponensként jelenik meg, ez elősegíti a kód újrafelhasználhatóságát és karbantarthatóságát.
* Kétirányú adatbinding: Az adatmodell és a felhasználói felület automatikusan szinkronban marad.
* Beépített routing: Könnyű navigációt biztosít az oldalak és nézetek között egyetlen oldal betöltése után.
* CLI (Command Line Interface): Az Angular CLI gyors és szabványos módot kínál projektek létrehozására, komponensek generálására és fejlesztési szerver indítására.
* Dependency Injection (DI): Segítségével lazán csatolt, jól tesztelhető és skálázható alkalmazások hozhatók létre.

### Használati területek

Az Angular-t elsősorban összetett, nagyvállalati webalkalmazásokhoz használják, ahol fontos a strukturált fejlesztés és a hosszú távú karbantarthatóság. Ugyanakkor kisebb projektekhez is alkalmas, különösen, ha több fejlesztő dolgozik együtt, vagy ha előre meghatározott fejlesztési szabályokra van szükség.

### Előnyök és hátrányok

Előnyök:

* Teljes körű megoldás: nem kell külön könyvtárakat keresni az alapfunkciókhoz
* TypeScript miatt stabilabb és könnyebben hibakereshető kód
* Hosszú távon jól skálázható
* Nagy közösség és sok hivatalos eszköz, dokumentáció

Hátrányok:

* Meredek tanulási görbe, különösen kezdőknek
* Néha túlságosan struktúrált, kisebb projektekhez túl „nehéz” lehet
* A komplex build rendszer miatt a konfiguráció elsőre nehézkes lehet

### Összegzés

Az Angular egy hatékony és professzionális keretrendszer, amely ideális választás lehet komplex webes alkalmazásokhoz. Bár tanulása több időt vehet igénybe, cserébe jól szervezett, stabil és skálázható megoldást kínál, különösen nagyobb fejlesztőcsapatok számára.

## 

## Bootstrap:

A Bootstrap egy nyílt forráskódú front-end keretrendszer, amelyet eredetileg a Twitter fejlesztett ki. Célja, hogy megkönnyítse a modern, reszponzív weboldalak és webalkalmazások készítését egységes dizájnnal és elrendezéssel. A Bootstrap HTML, CSS és JavaScript alapokon nyugszik, és széles körben használt mind a kezdő, mind a haladó fejlesztők körében.

Miért érdemes Bootstrapet használni?  
 A Bootstrap legnagyobb előnye, hogy előre elkészített komponenseket és stílusokat biztosít, így a fejlesztőknek nem kell minden egyes gombot, táblázatot vagy űrlapot egyedileg megtervezniük. Emellett mobilbarát kialakítást kínál, amely automatikusan alkalmazkodik a különböző képernyőméretekhez, ezzel gyorsítva a fejlesztési folyamatot.

### Fő jellemzők

* Reszponzív rácsrendszer (Grid system): 12 oszlopos elrendezési rendszer, amely könnyen alkalmazható bármilyen képernyőmérethez.
* Előre definiált komponensek: Gombok, navigációs sávok, modálablakok, kártyák, űrlapelemek és sok más beépített vizuális elem.
* Testreszabhatóság: A Bootstrap SCSS (SASS) alapon is elérhető, így egyszerűen személyre szabható saját arculathoz.
* JavaScript komponensek: Beépített dinamikus elemek, mint például lenyíló menük, csúszkák (carousel), figyelmeztetések (alerts), vagy collapsible panelek.
* Egységes dizájn: Az egységes vizuális stílus segít abban, hogy az alkalmazás minden része harmonizáljon egymással.

### Használati területek

A Bootstrap kiváló választás weboldalak, admin felületek, portfóliók, landing oldalak és egyszerűbb webalkalmazások fejlesztésére. Gyakran használják gyors prototípus készítésre is, mivel percek alatt összeállítható vele egy működő dizájn.

### Előnyök és hátrányok

Előnyök:

* Egyszerű használat és gyors tanulás
* Mobilbarát és reszponzív kialakítás
* Kiterjedt dokumentáció és nagy közösség
* Időmegtakarítás az előre elkészített elemek révén

Hátrányok:

* Az alapértelmezett stílus könnyen felismerhető, hacsak nincs testreszabva
* Túlzott használata sablonossá teheti a weboldalt
* Egyes esetekben túlméretezett lehet egyszerűbb projektekhez

### Összegzés

A Bootstrap ideális választás azoknak a fejlesztőknek, akik gyorsan szeretnének esztétikus, reszponzív weboldalakat készíteni minimális CSS-írás mellett. Egyszerűsége, megbízhatósága és gazdag funkcionalitása miatt a világ egyik legnépszerűbb front-end keretrendszere.

## 

## OpenAPI

Az OpenAPI Specification (korábban Swagger Specification néven ismert) egy széles körben elterjedt, nyílt szabvány a REST alapú webszolgáltatások leírására. Az OpenAPI célja, hogy gépileg olvasható, mégis ember számára is könnyen értelmezhető formátumban definiálja az API végpontokat, azok működését, adatstruktúráit és válaszait. A specifikáció JSON vagy YAML formátumban íródik, és egyfajta „szerződésként” szolgál az API fejlesztői és felhasználói között.

Miért fontos az OpenAPI?  
 Egy API dokumentálása gyakran kihívást jelent, különösen akkor, ha több csapat vagy rendszer használja. Az OpenAPI lehetővé teszi az automatikus dokumentációk, klienskönyvtárak, tesztesetek vagy akár mock szerverek generálását is. Ez növeli az átláthatóságot, gyorsítja a fejlesztést, és csökkenti a hibák számát.

### Fő jellemzők

* Platformfüggetlenség: Nem kötődik egyetlen nyelvhez vagy keretrendszerhez sem – szinte bármilyen technológiával használható.
* Swagger eszközkészlet: Az OpenAPI-t támogató eszközök, mint például a Swagger UI, Swagger Editor vagy Swagger Codegen, jelentősen megkönnyítik a fejlesztést és a tesztelést.
* Automatizálható dokumentáció: A dokumentációt nem kézzel kell írni, sok keretrendszer (pl. Spring Boot, NestJS, FastAPI) automatikusan képes OpenAPI definíciókat generálni.
* Valósidejű API tesztelés: A Swagger UI segítségével interaktív módon lehet kipróbálni az API hívásokat, ami különösen hasznos a fejlesztők és tesztelők számára.
* Sémaalapú modell: A kimenetek és bemenetek pontosan definiálhatók, típusokkal, kötelező mezőkkel, példákkal, leírásokkal.

### Használati területek

Az OpenAPI ideális bármilyen RESTful API-hoz, különösen ott, ahol a dokumentálás, automatizálás, vagy kliensgenerálás fontos. Nagyvállalati környezetekben gyakori, de startupok és kisebb fejlesztőcsapatok is előszeretettel alkalmazzák.

### Előnyök és hátrányok

Előnyök:

* Automatizálható fejlesztési és tesztelési folyamatok
* Könnyen olvasható és szerkeszthető formátum (YAML/JSON)
* Kiváló támogatottság keretrendszerek és eszközök szintjén
* Egységes kommunikációs alap fejlesztők és felhasználók között

Hátrányok:

* Kezdetben bonyolult lehet a séma írása kézzel
* Csak REST API-khoz használható (GraphQL vagy gRPC esetén nem releváns)
* Nagyobb rendszerek esetén a dokumentum bonyolulttá válhat

### Összegzés

Az OpenAPI egy erőteljes és szabványos eszköz REST API-k dokumentálására, amely megkönnyíti az együttműködést, automatizálja a fejlesztést, és segíti a hibák elkerülését. A mai modern backend- és frontendfejlesztésben szinte nélkülözhetetlen szerepet tölt be.

# Zeneszer Telepítési Útmutató

Ez az útmutató részletesen bemutatja, hogyan telepítheti és futtathatja a Zeneszer alkalmazást. Az alkalmazás Angular-t használ a frontendhez és Spring Boot-ot a backendhez. Az útmutató követésével telepítheti az összes szükséges függőséget, konfigurálhatja a környezetet, és futtathatja az alkalmazást helyileg vagy éles környezetben.  
  
3. **Swagger UI:** -

## Előfeltételek

A telepítési folyamat megkezdése előtt győződjön meg arról, hogy a rendszere megfelel az alábbi követelményeknek:

1. **Operációs rendszer**: Windows, macOS vagy Linux.
2. **Java Fejlesztői Készlet (JDK)**: 17-es verzió szükséges.
3. **Node.js és npm**: Node.js 22.14.0 vagy újabb.
4. **H2 adatbázis**: Beépített, nem igényel külön telepítést (fejlesztési környezethez).
5. **Maven**: A backend fordításához Apache Maven szükséges (ajánlott legalább a 3.8-as verzió).

## 1. lépés: JDK 17 telepítése

1. **JDK 17 letöltése:**Látogasson el az [Oracle JDK Letöltések](https://www.oracle.com/java/technologies/javase-jdk17-downloads.html) vagy az [OpenJDK Projekt](https://openjdk.org/projects/jdk/17/) oldalára, és töltse le a megfelelő verziót.
2. **JDK telepítése:**Futtassa a telepítőt, kövesse az utasításokat. Jegyezze fel a telepítési könyvtárat.
3. **JAVA\_HOME környezeti változó beállítása:**
   * **Windows:**
     + Start menü -> „Környezeti változók” -> „Rendszertulajdonságok” -> „Környezeti változók”.
     + Új rendszerváltozó neve: JAVA\_HOME, értéke pl. C:\Program Files\Java\jdk-17.
     + Path változóhoz adja hozzá: %JAVA\_HOME%\bin
   * **Linux/macOS:**Terminálban szerkessze a ~/.bashrc vagy ~/.zshrc fájlt:  
     export JAVA\_HOME=/path/to/jdk17

export PATH=$JAVA\_HOME/bin:$PATH

Mentés után futtassa:  
source ~/.bashrc

1. **Telepítés ellenőrzése:**

Terminálban vagy parancssorban:  
java -version

A kimeneten ellenőrizhető, hogy a JDK 17 telepítve van.

## 2. Maven telepítése

### Maven letöltése:

Látogasson el az [Apache Maven letöltési oldalára](https://maven.apache.org/download.cgi), és töltse le a legújabb verziót.

### Kicsomagolás és beállítás:

Csomagolja ki a letöltött archívumot egy tetszőleges könyvtárba, például:  
C:\Program Files\Apache\maven vagy /opt/maven  
Állítsa be a MAVEN\_HOME környezeti változót erre a könyvtárra, és adja hozzá a Maven bin mappáját a Path változóhoz.

#### Windows:

"Környezeti változók" ablakban új változó:  
 Név: MAVEN\_HOME  
 Érték: pl. C:\Program Files\Apache\maven

A Path-hoz adja hozzá: %MAVEN\_HOME%\bin

### Linux/macOS:

Tegye be a következő sorokat a ~/.bashrc vagy ~/.zshrc fájlba:  
export MAVEN\_HOME=/opt/maven

export PATH=$MAVEN\_HOME/bin:$PATH

Mentés után frissítse a shellt:  
source ~/.bashrc

## 3. lépés: Node.js és npm telepítése

### Node.js letöltése:

Látogasson el a [Node.js Hivatalos Weboldalára](https://nodejs.org/), és töltse le a legújabb npm lts verziót.

### Node.js telepítése:

Futtassa a telepítőt, kövesse az utasításokat. Az npm is automatikusan települ.

### Telepítés ellenőrzése:

Terminálban vagy parancssorban:

node -v

npm -v

A kimeneten ellenőrizhető, hogy az alkalmazás telepítve van.

## 4. lépés: A kódtár klónozása

Nyisson meg egy terminált vagy parancssort.  
Futtassa az alábbi parancsokat:

git clone <https://github.com/bitoatt-moon42/zeneszer>

cd zeneszer

## 5. lépés: Frontend függőségek telepítése

Navigáljon a frontend könyvtárba:  
cd src/main/webapp

Telepítse a szükséges függőségeket:  
npm install

Ez letölti az összes csomagot, amely a package.json fájlban szerepel.

Indítsd

ng build --watch --configuration development

## 6. lépés: Frontend buildelése és futtatása

Buildelje az alkalmazást:  
npm run build

Ez létrehoz egy éles környezetre kész buildet az Angular alkalmazásból a dist könyvtárban.

Indítsa el a fejlesztői szervert:  
npm start

A frontend elérhető lesz a<http://localhost:4200> címen.

## 7. lépés: Backend buildelése és futtatása

Navigáljon vissza a projekt gyökérkönyvtárába:  
cd ../../..

Buildelje a backendet Maven segítségével:  
mvn clean install

Ez lefordítja a kódot, futtatja a teszteket, és csomagolja az alkalmazást.

Indítsa el a Spring Boot alkalmazást:  
mvn spring-boot:run

A backend elérhető lesz a<http://localhost:8080> címen.

## 8. Swagger UI

API dokumentáció elérhető: `<http://localhost:8080/swagger-ui.html>`

## 3. Backend Architektúra

**MVC minta (Model-View-Controller) – Áttekintés**

Az **MVC (Model-View-Controller)** egy szoftvertervezési minta, amelyet gyakran használnak grafikus felhasználói felülettel (GUI) rendelkező alkalmazások fejlesztésében, valamint webes alkalmazásokban is. A célja, hogy szétválassza az alkalmazás adatkezelését, megjelenítését és a felhasználói interakció kezelését, így elősegítve a kód újrafelhasználhatóságát, karbantarthatóságát és tesztelhetőségét.

### A három fő komponens

1. **Model (Modell)**
   * A modell reprezentálja az alkalmazás üzleti logikáját és adatait.
   * Kezeli az adatbázis-interakciókat, számításokat, validációkat.
   * Nem tartalmaz semmilyen információt arról, hogyan jelenik meg az adat a felhasználó számára.
   * Példa: User osztály egy osztály, ami tartalmazza a felhasználó nevét, e-mail címét és egyéb adatait
2. **View (Nézet)**
   * A nézet felelős az adatok vizuális megjelenítéséért.
   * Csak azt jeleníti meg, amit a modellből vagy a kontrollertől kap.
   * Nem tartalmaz üzleti logikát.
   * Példa: egy HTML oldal vagy felhasználói felület, ami listázza a felhasználókat.
3. **Controller (Vezérlő)**
   * A vezérlő kezeli a felhasználói bemeneteket (pl. gombnyomás, űrlapkitöltés).
   * Frissíti a modellt, és meghatározza, melyik nézetet kell megjeleníteni.
   * Kapcsolatot tart a modell és a nézet között.
   * Példa: egy Spring Boot controller osztály, ami egy HTTP kérésre válaszként lekéri az adatokat és visszaadja a megfelelő nézetet vagy JSON választ.

**Entitás struktúra**

A **Musical Instrument Shop** webalkalmazás két alapvető entitása a User és az Instrument. Ezek az osztályok a rendszer adatmodelljének központi elemei, amelyek az adatbázisban tárolt felhasználói és hangszer-információkat képviselik. Mindkét entitás JPA segítségével valósul meg, és támogatják az automatikus naplózást (auditing), ami lehetővé teszi a rekordok létrehozásának és módosításának időpontjának automatikus nyomon követését.

A **User** entitás a rendszerben regisztrált felhasználókat reprezentálja. Minden felhasználónak van egy egyedi, automatikusan generált UUID azonosítója, valamint egy kötelező name (név) és password (jelszó) mezője. A jelszó tárolása csak titkosított formában javasolt. A dateCreated és lastUpdated mezők segítségével a rendszer automatikusan rögzíti, mikor hozták létre vagy módosították a felhasználói fiókot. Az @EntityListeners és az AuditingEntityListener lehetővé teszi ezeknek a mezőknek az automatikus kezelését.

Az **Instrument** entitás a boltban elérhető hangszereket írja le. A hangszerek szintén rendelkeznek egyedi UUID azonosítóval. A type mező egy felsorolás (enum), amely a hangszer típusát jelöli, például gitár, dob vagy zongora. A brand (márka), name (név) és price (ár) mezők a hangszer alapvető jellemzőit tartalmazzák. Az imageUrl mező opcionálisan lehetővé teszi egy kép URL-jének tárolását, amely vizuálisan reprezentálja a terméket. A price mező BigDecimal típusa pontos pénzügyi számításokat tesz lehetővé. A dateCreated és lastUpdated mezők ebben az entitásban is automatikusan frissülnek a megfelelő műveletek során.

Mindkét entitás jól példázza a Java Spring Boot alkalmazásokban gyakran alkalmazott konvenciókat: az annotáció-alapú konfigurációt, az automatikus mezőkezelést és az auditálási képességeket. Az adatszerkezetek egyszerűek, de bővíthetőek, például kapcsolatok (kapcsolattáblák) hozzáadásával, új mezőkkel vagy validációs szabályokkal. Ez az alap biztosítja, hogy az alkalmazás adatai strukturáltak, biztonságosan kezelhetők és hosszú távon is fenntarthatók legyenek.

**Repository layer**

A repository réteg az adatkezelésért felelős komponens egy többrétegű alkalmazás architektúrában, különösen a Spring Boot keretrendszerben. Feladata az, hogy leválassza az üzleti logikát az adatbázis-kezelés részleteiről, és egységes, könnyen használható API-t biztosítson az adatok lekérdezéséhez, mentéséhez, frissítéséhez és törléséhez.

A Musical Instrument Shop alkalmazásban a repository réteg a JpaRepository interfész kiterjesztésével valósul meg, amelyet a Spring Data JPA biztosít. Ennek segítségével automatikusan rendelkezésre állnak az alapvető **CRUD** (Create, Read, Update, Delete) műveletek anélkül, hogy ezek implementációját külön le kellene írni. A Spring Data intelligens metódusnév-feldolgozást is kínál: amennyiben egy repository interfészben a metódus neve megfelel bizonyos mintáknak (például findBy...), a keretrendszer automatikusan lefordítja azt egy SQL lekérdezéssé.

Az **InstrumentRepository** az **Instrument** entitás kezelését végzi. Alapértelmezett CRUD műveleteken kívül tartalmaz egy egyedi metódust is, amely a hangszereket azok típusa szerint listázza: findByType. Ez a metódus kiegészül egy Sort paraméterrel is, amely lehetővé teszi, hogy az eredményeket például ár vagy név szerint rendezzük. Ez rugalmasságot ad a felhasználói felületen történő listázásokhoz.

A **UserRepository** a **User** entitás perzisztenciájáért felelős. Egyedi lekérdezése a findByNameAndPassword, amely egy adott felhasználót keres meg név és jelszó alapján. Ez a megoldás egyszerű hitelesítési logikákhoz használható, de fontos megjegyezni, hogy a jelszavak nyílt szöveges tárolása és keresése nem biztonságos gyakorlat. Valós alkalmazásban a jelszavakat hash-elt formában kell tárolni, és a hitelesítést ennek megfelelően kell elvégezni.

Összességében a repository réteg az alkalmazás adatbázis-kezelésének sarokköve. Elrejti az adatbázis műveletek részleteit, miközben egyszerű, deklaratív módot biztosít a lekérdezések végrehajtására. Könnyen tesztelhető, bővíthető, és elősegíti a tiszta architektúra kialakítását. A Spring Data JPA pedig nagyban leegyszerűsíti a fejlesztést azzal, hogy a legtöbb adat-hozzáférési logikát automatikusan kezeli.

**Service layer**

A service (szolgáltatás) réteg az alkalmazás üzleti logikájának központi eleme, amely hidat képez a controller (prezentációs) és a repository (adatelérési) réteg között. Célja az adatokkal kapcsolatos műveletek kezelése, logikai feldolgozások elvégzése, valamint annak biztosítása, hogy az alkalmazás üzleti szabályai érvényesüljenek az adatkezelés során. A Musical Instrument Shop alkalmazásban a **UserService** és az **InstrumentService** osztályok képviselik ezt a réteget.

A service osztályok felelőssége az, hogy önálló, jól tesztelhető egységekként működjenek, és ne tartalmazzanak fölösleges technikai részleteket, például HTTP- vagy adatbázis-kezelési logikát. Ehelyett azokkal az entitásokkal és DTO-kkal (Data Transfer Object) dolgoznak, amelyek az adatokat reprezentálják, és átalakításokat végeznek ezek között.

A DTO, azaz Data Transfer Object, egy olyan objektum, amely az adatok továbbítására szolgál az alkalmazás rétegei között, különösen a prezentációs (pl. controller) és a service vagy domain réteg között. Fő célja az, hogy leválassza a külső kommunikációs formátumokat (pl. JSON) az adatbázisban használt entitásokról, és így biztonságosabb, letisztultabb, valamint rugalmasabb adatkezelést tegyen lehetővé.

A Musical Instrument Shop alkalmazásban a InstrumentDTO és a UserDTO osztályok töltik be ezt a szerepet. Ezek a DTO-k reprezentálják a Instrument és User entitások azon mezőit, amelyekre a kliensoldali kommunikáció során szükség van. A DTO-k nem tartalmaznak adatbáziskezelési vagy üzleti logikát – pusztán az adatok hordozására szolgálnak.

A InstrumentDTO például a hangszer típusát, márkáját, nevét, árát és képének URL-jét tartalmazza. A mezőkhöz validációs annotációk is tartoznak, mint például @NotNull, @Size és @Digits, amelyek biztosítják, hogy az API-n keresztül érkező adatok megfeleljenek a kívánt formátumnak és értéktartománynak. A @JsonFormat és @Schema annotációk a JSON-szerializáció és az OpenAPI dokumentáció támogatására szolgálnak.

A UserDTO hasonló módon épül fel, tartalmazza a felhasználó nevét és jelszavát, szintén validációs szabályokkal ellátva. A DTO-k használatával elkerülhető az entitások közvetlen kitettsége az API-n keresztül, így megelőzhetők olyan hibák vagy biztonsági kockázatok, mint az érzékeny mezők (pl. audit adatok) kiszivárgása.

Az InstrumentService az Instrument entitásra épülő műveleteket valósítja meg. A findByType metódus például lehetővé teszi a hangszerek típus szerinti szűrését, valamint rendezi az eredményeket márka és név szerint. A get, create, update és delete metódusok a szokásos CRUD műveleteket fedik le, és mindegyik gondoskodik a megfelelő entitás-DTO konverzióról. Ez biztosítja, hogy a felhasználó felé csak a szükséges és formázott adatok kerüljenek továbbításra, míg az adatbázisban a helyes, konzisztens szerkezetű objektumok tárolódjanak. A mapToDTO és mapToEntity metódusok segítik ezt a kétirányú átalakítást.

A UserService hasonló módon működik, de a User entitásra fókuszál. Egyedi metódusa az authenticate, amely egy egyszerű hitelesítési logikát valósít meg a felhasználónév és jelszó alapján. Itt is megtalálhatók a CRUD műveletek, valamint a DTO-konverziókat végző privát segédfüggvények. Fontos megemlíteni, hogy a jelszavak jelenlegi tárolása és kezelése biztonsági kockázatot jelent, mivel nyílt szövegként vannak kezelve. Egy éles alkalmazásban a jelszavakat minden esetben hash-elni kellene, például BCrypt használatával.

A service réteg legnagyobb előnye, hogy modularizálja és újrahasznosíthatóvá teszi az üzleti logikát. A controller réteg kizárólag ennek az API-ját használja, így a logika centralizált és egységes marad. Ez különösen hasznos a bővítés, új funkciók bevezetése, illetve az egységtesztelés során, mivel a service osztályok könnyen izolálhatóak és tesztelhetőek.

Összefoglalva, a service réteg kulcsfontosságú szerepet játszik egy jól strukturált, skálázható és karbantartható alkalmazás felépítésében. Közvetítőként működik a felhasználói felület és az adatbázis között, miközben megvalósítja az üzleti logikát és biztosítja az adatkezelés integritását.

**Controller layer**

A **REST** (Representational State Transfer) egy elterjedt architekturális stílus, amelyet webes szolgáltatások tervezésére használnak. A REST alapelvei lehetővé teszik az adatok egységes és jól strukturált módon történő elérését HTTP műveleteken keresztül (GET, POST, PUT, DELETE stb.). A Spring Boot keretrendszer kiváló támogatást nyújt RESTful API-k fejlesztéséhez a @RestController, @RequestMapping és más hasznos annotációk révén.

A *Musical Instrument Shop* alkalmazásban a controller réteg (más néven web vagy REST réteg) felel az ügyfél (pl. frontend vagy más külső rendszer) és az alkalmazás backendje közötti kommunikáció kezeléséért. Ez a réteg dolgozza fel az érkező HTTP-kéréseket, hívja meg a megfelelő service metódusokat, majd visszaadja a választ az ügyfélnek JSON formátumban.

### InstrumentResource

Az InstrumentResource osztály a REST végpontokat valósítja meg a hangszer entitások kezeléséhez. Az osztály @RestController annotációval van ellátva, így Spring automatikusan felveszi HTTP vezérlőként, a @RequestMapping pedig meghatározza, hogy az osztály metódusai az /api/instruments útvonalon érhetők el JSON válaszformátummal.

Ez az osztály a következő műveleteket támogatja:

* **GET /api/instruments** – A getAllInstruments metódus opcionálisan szűri a hangszereket típus alapján, majd visszaadja az eredményt listaként.
* **GET /api/instruments/{id}** – Egy konkrét hangszer lekérdezése ID alapján.
* **POST /api/instruments** – Új hangszer létrehozása a kapott InstrumentDTO alapján. A bemenet validálva van, és sikeres művelet esetén a válasz státuszkódja 201 (Created).
* **PUT /api/instruments/{id}** – Meglévő hangszer adatainak frissítése.
* **DELETE /api/instruments/{id}** – Hangszer törlése azonosító alapján, 204-es (No Content) válaszkóddal.

A controller nem tartalmaz üzleti logikát – a InstrumentService komponensre delegálja a műveleteket, ezzel biztosítva a tiszta rétegződést és a karbantarthatóságot.

### UserResource

A UserResource a felhasználókezelés REST végpontjait valósítja meg az /api/users útvonal alatt. A controller támogatja a CRUD műveleteket és egyszerű hitelesítési logikát is kezel.

A végpontjai:

* **GET /api/users** – Az összes felhasználó lekérdezése.
* **GET /api/users/{id}** – Egy konkrét felhasználó adatainak lekérdezése.
* **GET /api/users/{name}/{password}** – Alapvető felhasználóazonosítás név és jelszó alapján. Ez a megközelítés nem biztonságos (ld. megjegyzés lejjebb).
* **GET /api/users/loggedInUser** – A memóriában tárolt „bejelentkezett” felhasználó lekérdezése.
* **GET /api/users/logout** – Kijelentkezés: a loggedInUser mező törlésre kerül.
* **POST /api/users** – Új felhasználó létrehozása, ha az adott név-jelszó páros még nem létezik.
* **PUT /api/users/{id}** – Felhasználói adatok frissítése.
* **DELETE /api/users/{id}** – Felhasználó törlése.

A UserResource egy loggedInUser mezőt használ a bejelentkezett felhasználó ideiglenes tárolására. Ez azonban nem skálázható és nem biztonságos megközelítés – produkciós környezetben helyette autentikációs és autorizációs megoldások, mint a Spring Security vagy OAuth2 ajánlottak.

### Összegzés

A controller réteg célja az, hogy interfészt biztosítson a kliens és a backend üzleti logikája között. A RESTful végpontok a HTTP protokoll műveleteinek megfelelően vannak kialakítva, így az API logikusan strukturált, jól dokumentálható (pl. Swagger/OpenAPI segítségével), és könnyen használható külső rendszerek vagy frontendek számára. A @Valid annotáció révén automatikus bemenet-ellenőrzés is megvalósul, így csökkenthető a hibás adatok kezelése.

A jelenlegi implementáció jól példázza a REST alapelvek alkalmazását, azonban a biztonság növeléséhez szükséges lenne hitelesítési és jogosultságkezelési rétegek beépítése is. Emellett érdemes lehet bevezetni egységes hibatűrést (pl. kivételkezelő osztály), hogy az API még robusztusabb és átláthatóbb legyen.

# 3. Frontend Architektúra

## Áttekintés

Ez az Angular frontend alkalmazás egy teljes körű CRUD (Create, Read, Update, Delete) felületet biztosít a hangszerek kezeléséhez. A rendszer moduláris komponensekre épül, Angular best practice-eket követve, és olyan funkciókat tartalmaz, mint az űrlapvalidáció, hibakezelés és típus-specifikus hangszerek kezelése.

## Főbb Komponensek

### Kezdőoldal:

A kezdőoldal (HomeComponent) az A&D Instrument Store webáruház bemutatkozó felülete, amely egyszerű és letisztult designnal hívja fel a látogatók figyelmét az áruház kínálatára.

#### Főbb jellemzők

## Üdvözlő szöveg: Nagybetűs, kiemelt címsor üdvözli a látogatókat

## Navigációs gomb: "Explore our store" gomb, amely a hangszerlistára vezet

## Bemutató kép: Nagy méretű, reszponzív kép egy hangszerboltról

## Egyszerű kialakítás: Minimalista megjelenés a tartalomra fókuszálva

#### Technikai részletek

## Komponens: HomeComponent

## Használt modulok: CommonModule (alap Angular funkciók)

## Stíluslap: Saját SCSS fájl (home.component.scss)

## Kép megjelenítés: Bootstrap img-fluid osztállyal reszponzívvá téve

## Külső hivatkozás: Kép betöltése külső forrásból (Squarespace CDN)

### Hangszer kezelés:

#### InstrumentService

Szolgáltatás, amely kezeli az API kommunikációt a következő metódusokkal:

* getAllInstruments(): Összes vagy típus szerint szűrt hangszerek lekérése
* getInstrument(): Egy hangszer lekérése ID alapján
* createInstrument(): Új hangszer létrehozása
* updateInstrument(): Meglévő hangszer módosítása
* deleteInstrument(): Hangszer törlése

### Funkcionális Komponensek

#### InstrumentListComponent

* Táblázatos formában jeleníti meg a hangszereket
* Típus-specifikus nézeteket mutat szűréskor (gitárok, dobok stb.)
* Létrehozás/szerkesztés/törlés műveleteket biztosít bejelentkezett felhasználók számára
* Megerősítő párbeszédablak a törlésekhez
* Kezeli az entitások közötti referenciális korlátozásokat

#### InstrumentAddComponent

* Űrlap új hangszerek létrehozásához
* Dinamikus cím a hangszer típusa alapján
* Mezővalidáció (kötelező mezők, maximális hossz, numerikus formátum)
* Típus előzetes kiválasztása típus-specifikus nézetből érkezéskor
* Sikeres/hiba üzenetek

#### InstrumentEditComponent

* Előre kitölti az űrlapot a meglévő hangszer adataival
* Hasonló validáció, mint a létrehozásnál
* Megőrzi a hangszer típus kontextusát
* Visszajelzés sikeres műveletről vagy hibáról

#### Technikai Jellemzők

1. **Reaktív Űrlapok**: Minden űrlap az Angular reaktív űrlap modulját használja egyedi validációval
2. **Típusbiztosság**: Erős típusosság TypeScript interfészekkel
3. **Hibakezelés**: Központosított hibakezelő szolgáltatás
4. **Hitelesítés integráció**: Felhasználói bejelentkezés állapotának ellenőrzése UI vezérlőkhöz
5. **Útválasztás**: Megőrzi a kontextust a nézetek közötti navigáláskor
6. **Internacionalizáció**: Fordítási lehetőségek i18n címkékkel
7. **Reszponzív design**: Bootstrap alapú elrendezés minden képernyőmérethez

#### Használati Folyamat

1. A felhasználók böngészhetik a hangszereket összevont vagy típus szerint szűrt nézetben
2. A bejelentkezett felhasználók CRUD műveletgombokat látnak
3. A létrehozás/szerkesztés űrlapok segítségével hangszerek hozhatók létre, szerkezthetőek, törölhetőek

### Felhasználókezelés

#### UserService

A felhasználókkal kapcsolatos backend kommunikációt kezelő szolgáltatás. Főbb funkciói:

* Felhasználók listázása (getAllUsers)
* Aktuális felhasználó lekérdezése (checkUser)
* Kijelentkezés (logout)
* Bejelentkezés (authenticate)
* Felhasználói CRUD műveletek (létrehozás, módosítás, törlés)

#### UserAddComponent

Kettős funkciójú komponens, ami kezeli:

1. **Bejelentkezési felületet**:
   * Felhasználónév és jelszó mezők
   * Validáció: kötelező mezők, max 255 karakter
   * Sikeres bejelentkezés esetén főoldalra navigál
2. **Regisztrációs felületet**:
   * Ugyanazok a mezők mint bejelentkezésnél
   * Új felhasználó létrehozása
   * Hibakezelés: már létező felhasználó esetén

## Lehetséges Jövőbeli Fejlesztések

* Áttérés perzisztens adatbázisra (pl. PostgreSQL, MySQL)
* Valós szerepkör alapú hitelesítés hozzáadása (Spring Security)
* Jelszó hash-elés
* Valós payment rendszer integrálása (Stripe, paypall, adyen)
* Számlázási rendszer integráció (szamlazz.hu, billingo)
* Kosár funkció
* Teszt automatizálás
* Lista szűrés, lapozás implementálása

## Források:

* Spring Boot: <https://spring.io/projects/spring-boot>
* Angular: <https://angular.io>
* Swagger / OpenAPI: <https://swagger.io/tools/swagger-ui/>
* Bootstrap: https://getbootstrap.com/docs/5.3/getting-started/introduction

## 