Lea

Best Protocol TEAM

*Szoftverfejlesztési projekt (BH-MIT063-L)*

**Készítette:**

**Oláh Dóra (VQ185X)**

**Surányi Krisztina (GL2B0Q)**

**Bondor Dániel (WGFOZD)**

**Szénási Tamás (R0MKN6)**

**Vadász Csaba (D18ZGX)**

**Gazdaságinformatikus alapképzés**

**A képen szöveg, aláírás látható

Automatikusan generált leírás**



****

Fejlesztői Dokumentáció



Tartalomjegyzék

[Rövid összefoglaló 4](#_Toc134170521)

[Fejlesztési módszertan 5](#_Toc134170522)

[Architektúra 6](#_Toc134170523)

[Hosting információ 9](#_Toc134170524)

[API Dokumentáció 10](#_Toc134170525)

[Tesztelési útmutató 12](#_Toc134170526)

[Verziókezelés 14](#_Toc134170527)

[Hibakezelés 16](#_Toc134170528)

[Telepítési útmutató 17](#_Toc134170529)

[Jövőbeni fejlesztési tervek 19](#_Toc134170530)

[Használati útmutató 19](#_Toc134170531)

# Rövid összefoglaló

A Milton Friedman Egyetem számára fejlesztünk egy szoftvert, amely képes protokoll listákat létrehozni ügyfelek kérésére. A szoftver tartalmaz egy Spring backendet (mint mikroszolgáltatás) és egy Angular frontendet. A program 3 nyelven használható: magyarul, angolul és héberül. Modulok: api contract yaml (mind a backend, mind a frontend az endpontot és a modellt hozza létre), tartalmazza az Eureka webszervert és van egy api gateway-e. A modulok: kormányzati szolgáltatás, amely tartalmazza a magyar minisztériumok és vállalatok képviselőit, valamint lehetővé teszi a nem rezidens intézmények tárolását. Tervezzük a felhasználói szolgáltatást, amelyet a felhasználók regisztrálására és bejelentkezésére használnak (a jövőben 2 faktoros hitelesítést tervezünk használni). Tervezzük az eseményeket kezelő szolgáltatást, és lehetőség van képviselők hozzáadására az eseményekhez, és excel fájl generálására az eseményekről.

A fejlesztői dokumentáció célja, hogy részletes információkat nyújtson a best-protocol alkalmazás fejlesztéséről és architektúrájáról. Az alkalmazás a Spring Framework és az Angular használatával készült, és backend és frontend részekből áll. Az API szerződés egy YAML fájlban található, amely leírja az API végpontokat és a kliens számára elérhető adatokat.

Az alkalmazás működő szolgáltatásai a Spring által regisztrált Eureka webszerveren találhatóak, majd továbbításra kerülnek az API gateway-en keresztül. Az egyes modulok terheléselosztóval rendelkeznek, így több példányban is futtathatóak, amelyeket az Eureka szerver automatikusan kezel. Az Angular az API gateway-en keresztül használja a backend szolgáltatásokat.

Az adatbázis MySQL-t használ, amelyben tárolódnak az alkalmazás adatai.

A projekt fő fájlja a "pom.xml", amely az Apache Maven konfigurációját tartalmazza. Az alkalmazás fejlesztése során ajánlott az IntelliJ IDEA vagy a Spring Tool Suite (STS) használata. Az alkalmazás tesztelése során JUnit, Mockito és Selenium WebDriver tesztkeretrendszereket használjuk.

Az alkalmazás forráskódja a GitHub-on található, amelynek címe: <https://github.com/Csaba79-coder/best-protocol>

Az alkalmazás fejlesztése és karbantartása folyamatos, így a dokumentáció a változásoknak megfelelően frissülhet.

# Fejlesztési módszertan

Fejlesztés során az Agilis módszertant alkalmaztuk, a Scrum és a Kanban tábla előnyeit használtuk együttesen, hogy a követelményeket teljesíthessük.

Készítettünk egy teljes backlogot, és user story-kat, alaposan megterveztük a projektet és mindezt egy Kanban Board-on rögzítettük, a projekt nyomonkövetése érdekében.

Scrum pozíciókat kiosztottuk, lásd PAD, valamint két hetes Sprintet állapítottunk meg. Minden Sprint végén tartottunk tartottunk egy Sprint Review-t, Grooming (backlog refinement), valamint egy következő Sprint planning-et. Ennek szerves része volt a Scrum Poker – sztori pontok meghatározása.

Minden este volt egy rövid Daily Stand-up, mely egyik este sem haladhatta meg a 10 perces időtartamot. Ezt tudtuk is tartani. Itt három kérdés hangzott el:

1. Mit csináltál tegnap? (What did you do yesterday?)
2. Mit fogsz ma csinálni? (What will you do today?)
3. Vannak-e akadályok, amik gátolják a munkádat? (Are there any impediments blocking your progress?)

# Architektúra

Az alkalmazásunk fejlesztése során a következő technológiákat és eszközöket használjuk:

Backend:

* Java Development Kit (JDK) 17
* Spring Boot 3
* Spring Framework 6
* API szerződés - OPENAPI - Swagger 3

Frontend:

* Angular CLI 15.1.6
* Angular 15.2.2
* Node 18.14.1
* npm 9.6.4

Az alkalmazás backend része Spring Framework és Angular szolgáltatásokra épül. Az API szerződésünk egy YAML fájlban található, amelyet Swagger 3 segítségével generálunk. Az alkalmazásunkat a Spring által regisztrált Eureka webszerveren találjuk meg, majd továbbításra kerül az API gateway-en keresztül. Az egyes modulok terheléselosztóval rendelkeznek, így több példányban is futtathatóak, míg az Angular az API gateway-en keresztül használja a backend szolgáltatásokat.

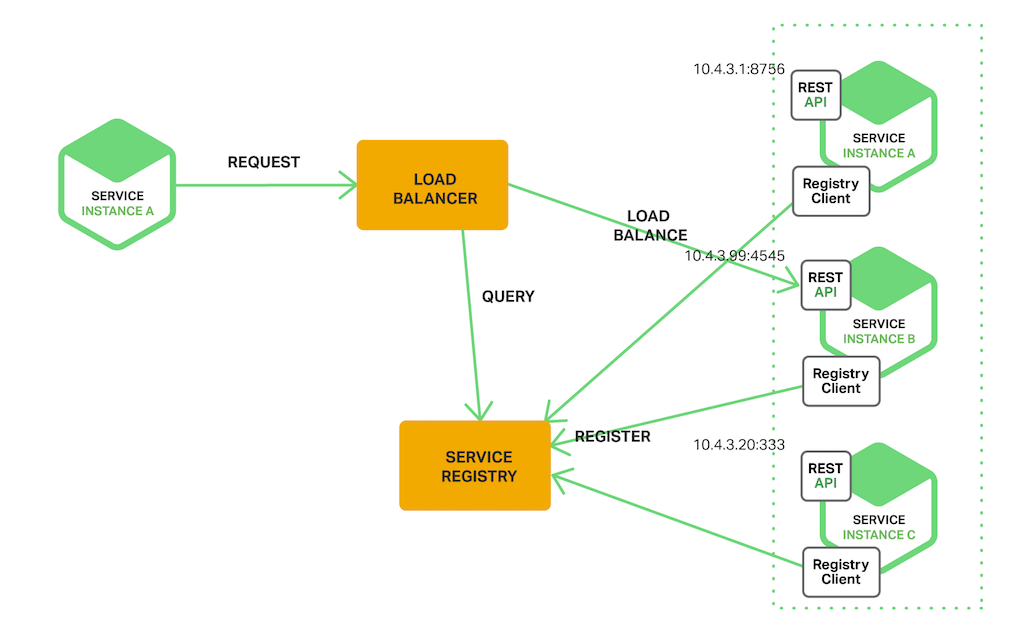
Az API REST API, JSON formátumban történő adatküldéssel. Fotó esetében pedig multipart adattípusban, mely a képet is tudja továbbítani.

Az Angular frontend rész a CLI 15.1.6 verziójával készült, amely az Angular 15.2.2 verzióját használja. A Node verziója 18.14.1, a package managerünk pedig a npm 9.6.4.

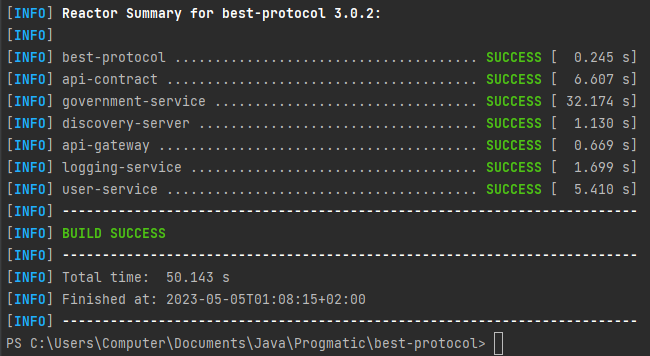
A fejlesztés során fontos, hogy a megfelelő verziószámokat használjuk az eszközöknél és a függőségeknél, hogy biztosítsuk az alkalmazás stabilitását és kompatibilitását.

Ezen technológiák segítségével lehetővé vált egy rugalmas, hatékony és skálázható backend architektúra létrehozása. A Spring Framework és a Spring Boot segítségével könnyedén implementálhatóak a különböző szolgáltatások, melyekhez a megfelelő kódok már előre elkészítve állnak rendelkezésre. Az API szerződés használata lehetővé teszi, hogy a frontend és a backend szorosan kapcsolódjon egymáshoz, a dokumentáció segítségével pedig könnyen megérthetővé válik a backend funkcionalitása és az API-n keresztül történő kommunikáció.

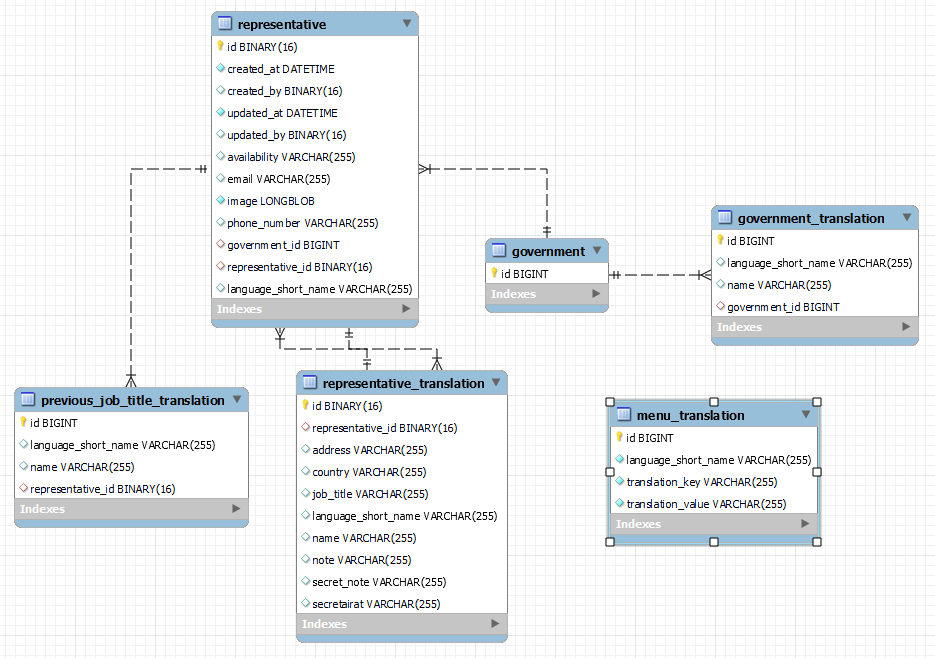
Backend Discovery Service a Microservices Architecturában



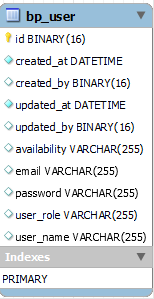
Programunk több modulból, szervízből áll. Egy belépési pontunk van, az API GateWay, ami a kellő konfiguráció mellett képes felismerni, hogy mely szerviz irányába kell a hívást továbbítani. Ez és az összes szerver egy Eureka Webserveren (discovery service) kerül regisztrálásra. A szervízek Load Balancer használatával több példányt is tudnak indítani. A backend az API GateWay-en keresztül kapcsolódik a frorntendhez is.



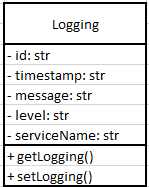
Government-service egy MySQL adatbázishoz kapcsolódik



User-service egy táblából áll (jövőbeni tervünk között szerepel a user-hez tartozó role kiszervezése egy másik adattáblába)



Logging-service egy Mongo DB (non relational database) kapcsolódik, mivel itt nem a táblák közötti kapcsolat, hanem a big data és annak kezelése volt a fókuszban



# Hosting információ

Az alkalmazás üzemeltetése során fontos betartani az alábbi lépéseket a Spring alkalmazások sorrendjének megfelelően: először az api-contract-yaml modult kell futtatni, majd az Eureka webszervert, majd az api-gateway-t, majd végül a különböző szolgáltatásokat, amelyek közül az elsőnek a government-service-t kell futtatni. A helyes sorrend betartása biztosítja az alkalmazás megfelelő működését.

A Spring backend telepítésekor a Maven használata kötelező, amely lehetővé teszi a projekt forráskódjának automatizált fordítását és csomagolását a szükséges fájlokba (pl. jar). Ezeket a jar fájlokat lehet majd futtatni a szerveren.

Az Angular frontend telepítéséhez szükséges a Node.js használata és az npm (Node Package Manager) csomagkezelő telepítése. Az alkalmazás buildelésekor az ng build parancs használható, amely a tartalmat az előre meghatározott dist mappába helyezi.

Az API szerződések YAML fájlokban kerülnek mentésre, és a Maven fordítási folyamatában az io.swagger.codegen swagger-codegen-maven-plugin plugin segítségével generálódnak a Java interfészek és modellek.

Ezek a fájlok, illetve a Spring és Angular alkalmazások összecsomagolásához szükséges egyéb fájlok (pl. konfigurációs fájlok) szintén csomagolásra kerülnek, majd a szerveren kerülnek telepítésre. A Springet mvn clean install paranccsal az Angulart pedig az ng build paranccsal kell előtte sikeresen futtatni. Ezek a file-ok konfigurálhatók majd a szerveren futtathatóra. Fontos, hogy a backend Spring modulok a háttérben fussanak. Ezek után szükséges az adatbázis szerveren történő konfigurálása is.

# API Dokumentáció

Elérhető végpontok:

GET /: Az index oldal letöltése

GET /index: Az index oldal letöltése

GET /index.html: Az index oldal letöltése

GET /{languageShortName}/api/admin/gov-representatives

POST /{languageShortName}/api/admin/gov-representatives

GET /{languageShortName}/api/admin/gov-representatives/governments/{governmentId}

GET /{languageShortName}/api/admin/governments

GET /{languageShortName}/api/admin/governments/{governmentId}

GET /{languageShortName}/api/admin/governments/menu

GET /eureka/web

GET /api/admin/logging-service/log

POST /api/admin/logging-service/log

GET /api/admin/user-service/users

POST /api/admin/user-service/users

GET /api/admin/user-service/users/{userId}

PUT /api/admin/user-service/users/{userId}

Tagek:

* government-representative
* government
* menu
* index
* log
* user

Válaszok:

* 200: Sikeres válasz
* 201: Sikeres művelet (létrehozva)
* 400: Hibás keresés
* 401: Nem engedélyezett
* 403: Tilos művelet
* 404: Nem található
* 405: Validációs hiba

Paraméterek:

* governmentId: A kormány azonosítója, amelyre szeretnénk szűrni
* languageShortName: A kiválasztott nyelv rövid neve
* search: A keresési lekérdezés karakterlánc formájában
* page: A lekérdezés által visszaadott oldalszám, alapértelmezés szerint 0
* size: Az oldalon megjelenő elemek száma, amelyeket a lekérdezés által visszaadunk, 1 és 100 között lehet, alapértelmezés szerint 10
* userId: A user adatbázisban tárolt egyedi azonosítója, mely alapján pl. módosítani vagy lekérni tudjuk!

A page és size paraméter lehetővé teszi backendi és frontendi oldalról a Pagination funkció használatát, ahol meg lehet határozni, hogy egy oldalon mennyi jelenjen meg, és lapozhatóvá válik az oldal.

Request és Response body adatformátumai:

text/html: A visszatérési érték típusa szöveg (HTML formátumban)

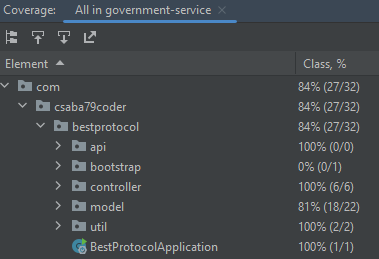
multipart/form-data: itt beérkezhet a szöveg mellett kép is

application/json: itt kizárólag json formában mehet az adat az endpoint-ok között

Fontos megjegyzés: Az API dokumentációban a modell osztályok (generált API modellek) vannak feltüntetve, nem pedig a szerveren tárolt adatbázis entitások. Ez biztonsági okokból történik, hogy az érzékeny adatok ne legyenek nyilvánosan elérhetőek. Kérjük, vegyék figyelembe ezt a különbséget az API használata során.

# Tesztelési útmutató

A program tesztelésére külön tesztkörnyezet nem került kialakításra. Jelenleg a fejlesztés azon fázisában járunk, hogy egység (unit) és integrációs tesztekkel rendelkezik a szoftver, mely a teljes modul min. 80%-át lefedi, ez egy alapelvárás volt a szoftverünk felé. Ez az alapfeltétele annak, hogy a kód GitHub-ra feltölthető legyen.



Fontos megjegyezni, hogy belső használatra került a bootstrap package adatfeltöltés céljából kialakítva, mely jelenleg nincs használatban, a kód comment blockba került, így tesztelni se teszteltük.

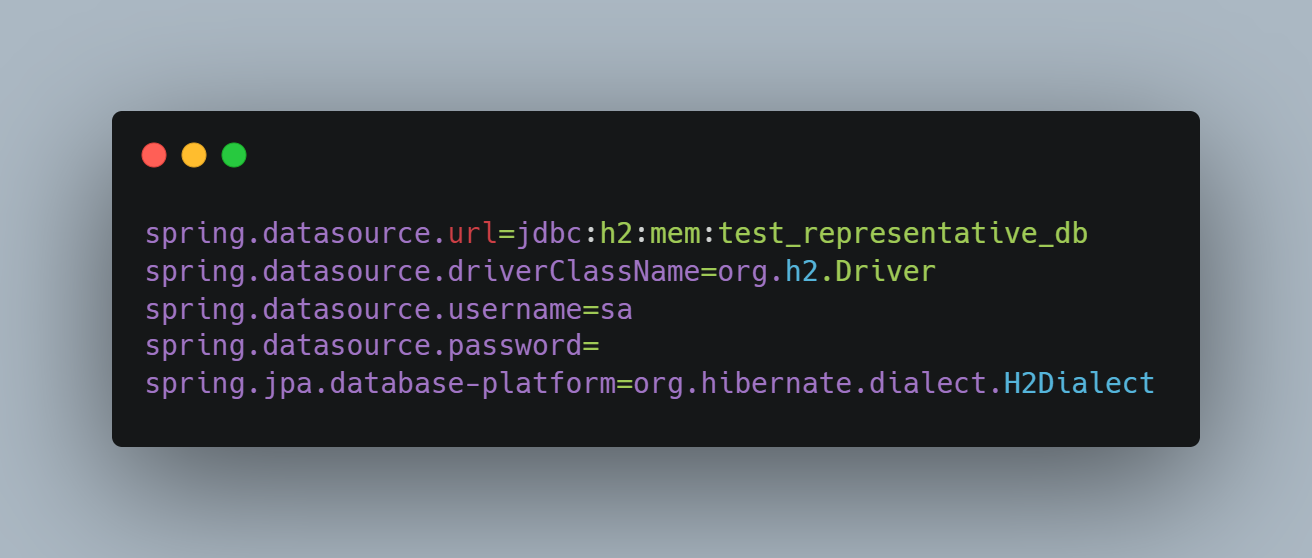
Teszt típusok:

* egységtesztek célja az adott egység (funkció, metódus) működésének ellenőrzése
* integrációs tesztek a különböző egységek közötti kommunikáció és adatcserék ellenőrzésére szolgálnak
* postman (végpont teszt)

A programunk minden végpontot egy integrációs teszt segítségével megfuttat és tesztel, továbbá az egyes funkciók működése unit test (JUnit5) használatával került tesztelérsre.

A végpontokat postman segítségével (mely web környezetet szimulál) szintén leteszteltük.

Az alkalmazás beállításai között találhatóak olyan tulajdonságok is, amelyek biztosítják, hogy az alkalmazás nem az eredeti adatbázist használja, hanem egy H2 in-memory adatbázist. A következő konfiguráció található a tesztelési környezetben:



Ezek a beállítások segítenek abban, hogy a teszt- és fejlesztői környezetek elkülönüljenek egymástól, és ne az eredeti adatbázist használják. Az in-memory adatbázis használata lehetővé teszi, hogy könnyen és gyorsan hozzáadhassuk, módosíthassuk és töröljük az adatokat a tesztelési folyamat során.

Jelenleg ami elérhető:

Az index oldal belépés nélkül egyből a meglévő és működő modulhoz vezérel. Itt a bal oldali nav bar lehetőséget ad az adatok szűrésére, és az Angular egy single page rendering segítségével a teljes listát leszűri a minisztériumokra. Ezt azonnal frissítve be is adja! A lap tetején található search bar pedig lehetőséget ad az adatok szűrésére. Ha a teljes listában vagyunk, akkor csak azon belül szűr, ha valamely minisztériumon belül történik a szűrés, akkor csak azon adatokon belül keres, amit az oldal oda betöltött. Fontos még kiemelni azt is, hogy bal oldalon egy drop down menü segítségével választhatunk nyelve, ez mint egy paraméter bekerül az url-be és azon nyelven adja vissza az adatbázis egyes entitását. Az entitás bármely tulajdonságára szűrhetünk, és a szűrés a kiválasztott nyelven történik. Fontos megjegyezni, hogy a nyelv azonosítója mellett a menühöz translation key-vel kerültek beállításra a fordítások, melyek a menü, vagy a gombok, vagy az oldalon feltűntetett szövegek url-ből érkező fordítását hozza be. Ezen fordítás menü esetében teljesen feltöltve. Az entitás azonban nem biztos, hogy rendelkezik fordítással, ez esetben hozzá csak a fordítást nem igénylő elemek töltődnek be, mint pl. fotó, email cím, telefonszám vagy éppen az adat elérhetősége.

A módosít vagy töröl gomb még fejlesztés alatt!

# Verziókezelés

Sikeresen implementáltunk egy kézi verziókezelési folyamatot, amely során a verziószámokat a Spring keretrendszeren keresztül frissítettük. A projekt történetét és fejlesztési folyamatát a verziószámok jelzik, melyek pontos leírását az alábbi:

1.0.0-RELEASE:

GitHub tároló létrehozása

projekt alapok, beállítások, konfiguráció, README.md, MANUAL.md + .gitignore és LICENCE fájlok

master ág

1.1.0-SNAPSHOT:

developer és api-contract ágak létrehozása

api-contract (végpontok, API interfészek és modellek) létrehozása

1.2.0-SNAPSHOT:

government-service létrehozása (alap adatokkal, egy adatbázisban tárolva, egy táblában)

1.3.0-SNAPSHOT:

Frontend - Angular alapok létrehozása (csak Angular-alapú megjelenítés)

2.0.0-SNAPSHOT verzióra történt az átállás:

környezetek létrehozása a frontend modulban

saját favicon beállítása az Angular eredeti helyére

logó hozzáadása a belső oldalakhoz

modellek és API interfészek létrehozása az api contract alapján

api-contract.yaml refaktorálása (hiányzó miniszteri hivatal nevének hozzáadása) és az modellek nevébe az "Admin" szó beillesztése

fotó feltöltése (helyettesítőkép) ha nincs kép a frontend részről (helyettesítőkép létrehozása az alap adatok megjelenítéséhez) Fotóknak van placeholder minden esetben, tehát ha a frontend nem biztosít fotót, akkor is egy placeholder-t feltölt a Spring backend, így biztos, hogy mindenki esetében érkezik fotó az adatbázisba (null contstrait miatt fontos volt, és az elképzelésünk miatt, megjeleníthetőség, látvány!)

2.1.0-SNAPSHOT:

A kormányzatot külön táblába helyezték, egy másik táblában tárolják a minisztériumokat (Long id)

Nézetek frissítése, formázás, színek megadása.

Minisztériumok osztály létrehozása, lehetőség az ID-ra történő szűrésre

Oldalsáv és útválasztó beállítva (működik)

Extra titkárság mező beállítása!

2.1.0-SNAPSHOT:

Többnyelvű fordítás kialakítása (Angular-ban nyelvváltás URL-al) - az adott nyelv lista URL-jének betöltése!

2.2.0-SNAPSHOT:

Menü idegen nyelvűsítése

Fordító szolgáltatás kialakítása (nem került külön modulba, hogy a fordítás ne zavarjon be az alapfunkcionalitásba az egyes szervízek esetében, később lehet külön module-t erre létrehozni, most vállaltuk egyes menük esetleges duplikált nyilvántartását az adatbázisban – jövőben ha egy modulba kerül, akkor egy helyen kell majd csak módosítani is!)

Backend és frontend részleges fordítása

2.3.0-SNAPSHOT:

Discovery Server: Eureka Discovery Server

2.3.1-SNAPSHOT:

API Gateway

2.3.2-SNAPSHOT:

Government Service Item Search

Keresőmező és keresési funkció

2.3.3-SNAPSHOT:

teszt írás és kód kommentelés (backend - frontend)

backend java doc generálás

3.0.0-SNAPSHOT:

index page és endpoint kialakítás, meglévő működő component url-re irányít autómatikusan!

3.1.0-SNAPSHOT:

log service – MongoDB használata (log esetében nincs szükség relational database-re, itt a big data a fókusz)

endpoint teszt – postman segítségével

jövőbeni teendő: unit és integrációs teszt elkészítése

3.2.0-SNAPSHOT:

user-service (CRUD)

backend lefejlesztve, új user hozzáadás, meglévő módosítás, soft törlés, egy user listázás és összes user kilistázása

A fejlesztési folyamatunk során a master branch szolgál a produkciós verziók tárolására. A fejlesztői branch ebből a master branch-ből ágazik el, és a fejlesztési folyamat során különálló feature branch-eket és api-contract branch-et hoz létre. A fejlesztői branch-ekre történő változtatásokat azután merge-lik be a developer branch-be, majd innen pedig a master branch-be. A fejlesztői branch-ek lehetővé teszik, hogy az egyes funkciókat és fejlesztéseket külön-külön dolgozhassuk ki, és csak akkor vegyük be őket a fő fejlesztői ágba, ha készen vagyunk velük.

# Hibakezelés

Api-contract alapján felmerülő hiba lehetőségek:

* 400: Hibás keresés
* 401: Nem engedélyezett
* 403: Tilos művelet
* 404: Nem található
* 405: Validációs hiba

Logolásra az @Slf4j-t használjuk, mint Spring annotációt, hibát globálisan kezelünk, saját hibakezelésünk is van. Hiba kódokat enum formájában hoztuk létre, szöveget is adunk a hiba dobásához, továbbá megjelöljük, hogy mely funkció, milyen szinten és miért dobott hibát (pl. egyedi id segítségével). Logszervizünk az összes modul logolását, így a hibák logolását is végzi, ehhez pontos dátumbélyegzőt is használ. A terminálban is megjelennek a hibák futáskor, nemcsak az adatbázisba kerülnek be. Jövőbeli tervünk a frontendi hibakezelés megoldása, továbbá saját error page, vagy error kód (user felé természetesen értelmezhető szöveggel)

# Telepítési útmutató

Termékünk egy szerveren fut, amely elérhető a böngészőben egy URL címen keresztül. A program eléréséhez internet kapcsolatra és böngészőre van szükség. Jelenleg a program igazodik a böngésző méretéhez, így minden gond nélkül használható asztali gépről, vagy akár mobilról is. Platformfüggetlen, így speciális rendszerkövetelménye nincs. Terveink között szerepel egy mobilalkalmazás fejlesztése Android és iOS platformokra. (telepíthető alkalmazás).

Szerver oldali konfiguráció javaslat a sikeres futtatás érdekében:

Az alkalmazásunkhoz szükség lesz a backend és a frontend külön-külön történő telepítésére. A backend önálló Spring Boot alkalmazásként telepíthető, a frontend pedig egy web szerverre, például Apache vagy Nginx használatával telepíthető.

Az alábbi lépéseket követhetjük az alkalmazásunk telepítéséhez:

Az Angular alkalmazást a ng build --prod parancs használatával építsük fel. Ez a parancs egy dist mappát fog generálni, ami tartalmazza a frontendhez szükséges összes fordított kódot és erőforrást.

Másoljuk át a dist mappa tartalmát a web szerverünk gyökérkönyvtárába.

Az alkalmazás Spring Boot backendjének építéséhez használjuk a mvn clean install parancsot. Ez egy jar fájlt fog generálni a target mappában.

Másoljuk át a jar fájlt azon a szerveren, ahol a backendet telepíteni szeretnénk.

A backend elindításához futtassuk a java -jar <name\_of\_your\_jar\_file>.jar parancsot.

Az alkalmazásunkat a webböngészőben az általunk megadott web szerver URL-címének megadásával érhetjük el.

Továbbá szükségünk lehet a web szerver konfigurálására annak érdekében, hogy a backend API-t proxyzni tudjuk. Erre egy fordított proxy beállításával van lehetőségünk Apache vagy Nginx használatával.

A ng build parancs létrehoz egy dist mappát, ami tartalmazza mindazt, ami szükséges az Angular alkalmazás futtatásához a webböngészőben. Ez magában foglalja az összes fordított TypeScript és HTML kódot, valamint az összes képet, betűtípust és CSS stíluslapot.

Ha az OpenAPI Generator CLI használatával egy API klienst generáltunk, és a build mappába helyeztük, akkor ezt a mappát is meg kell adnunk a szervernek a dist mappával együtt. Az API kliens kódja külön van az Angular alkalmazás kódjától, és szükséges ahhoz, hogy API-kéréseket küldjünk a backend szolgáltatásunkhoz.

Összefoglalva, mind a dist mappát, mind a build mappát (ami az API klienst tartalmazza) meg kell adnunk a szervernek ahhoz, hogy az alkalmazásunk megfelelően működjön.

# Jövőbeni fejlesztési tervek

log-service

Felkonfigurálása minden szerverre, beállítani mindenhol, hogy elküldje a modul nevét is mikor azt adatbázistáblába menti

Különböző log szintre, hiba szövegre, vagy részletre történő keresés, vagy akár dátum szerinti szűrés implementálása

Frontendi megjelenítése – kizárólag arra jogosult felhasználók részére (sys-admin)

user-service

A user szerviz jelenleg képes kezelni a felhasználókat, role-t beállítani hozzájuk, CRUD elkészült, azonban még a Spring Security-val összhangba kell hozni, és a role-okhoz be kell konfigurálni, hogy mely oldalt, vagy azon mely funkciókat éri el a jogosultsága alapján.

Jövőbeni terv szintén a kétfaktoros azonosítás, valamint a frontend oldali (UI) megjelenítés

happening-service

A három nyelv fejlesztése miatt erről mindenképp le kellett mondanunk az idő rövidsége miatt.

terveink szerint szerepel, hogy létra lehet hozni eseményeket (dátum, kezdés, várható befejezés, esemény neve, esemény pontos helyszíne, kültér-beltér megjelöléssel, lehet-e parkolni a helyszínen és egyéb megjegyzés rovattal ellátra). Ehhez szeretnénk hozzáadni a meghívott vendégeket. A vendégek nevét, email címét, telefonszámát és a fontos megjegyzés rovatot mindenképp szeretnénk itt a listában elhelyezni, és ebből eseményenként vagy akár személyenként excel formájában történő exportált listát szeretnénk készíteni.

# Használati útmutató

Lásd külön mellékletben: Felhasználói specifikáció