



## SZAKDOLGOZAT FELADAT

**Csортán Szilárd**

Mérnök-informatikus hallgató részére

### Eclipse-alapú modellező és modellellenőrző keretrendszer a felhőben OpenAPI segítségével

Ahogy a modellel alapú fejlesztési módszerek egyre elterjedtebbé válnak, úgy erősödik az igény a korszerűbb eszköztámogatásra is. Ennek egyik eleme, hogy az olyan funkciók mint pl. a kódgenerálás, modelltranszformációk, vagy modellellenőrzés a szoftverfejlesztésnél megszokott módon egy folytonos integrációs folyamat részét képezzék. Ráadásul ezen funkciók egy része igen erőforrásigényes, így adja magát az automatikusan skálázódó, felhőalapú szolgáltatások megjelenése is.

A Gamma Statechart Composition Framework egy kooperáló állapotgépek modellezését segítő keretrendszer, amelyet azonban nem csak tervezésre, hanem más modellező eszközök mögé illesztve kód- és tesztgenerálásra, modelltranszformációkra és modellellenőrzésre is lehet használni. A Gamma egy Eclipse-alapú eszköz, így jelen formájában nem felel meg a fentebb vázolt modern követelményeknek. A hallgató feladata ennek megfelelően az, hogy az alkalmazás szolgáltatásként is használható részeit parancssoron keresztül, illetve OpenAPI segítségével webes szolgáltatásként is elérhetővé tegye.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a modern követelmények megvalósításához szükséges technológiákat.
- Tervezze meg a Gamma keretrendszer webszolgáltatásként történő kiajánlásához szükséges szoftver- és rendszerkomponenseket.
- Alakítsa át a Gamma keretrendszer szolgáltatásként használható részeit egy parancssoron keresztül is használható komponenssé.
- Implementálja a Gamma szolgáltatásainak eléréséhez szükséges webes komponenseket és mutassa be a használatukat egy példán keresztül.
- Dokumentálja az elvégzett munkát és a megvalósult felületeket.

**Tanszéki konzulens:** Dr. Molnár Vince, adjunktus

Budapest, 2020.10.11.

.....  
Dr. Dabóczi Tamás  
tanszékvezető  
egyetemi tanár, DSz



**Budapesti Műszaki és Gazdaságtudományi Egyetem**  
Villamosmérnöki és Informatikai Kar  
Méréstechnika és Információs Rendszerek Tanszék

# **Eclipse-alapú modellező és modellellenőrző keretrendszer a felhőben OpenAPI segítségével**

SZAKDOLGOZAT

*Készítette*  
Csortán Szilárd

*Konzulens*  
dr. Molnár Vince

2020. december 9.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>1</b>
<b>2. Háttér ismeretek</b>	<b>4</b>
2.1. Modern technológiák . . . . .	4
2.1.1. OSGi . . . . .	4
2.1.2. Eclipse Plug-in fejlesztés . . . . .	5
2.1.3. REST API . . . . .	5
2.1.4. OpenAPI és Swagger . . . . .	8
2.1.5. Fontosabb Java könyvtárak . . . . .	8
2.2. Gamma . . . . .	9
2.2.1. Gamma funkciók . . . . .	9
2.2.2. Gamma hiányosságok . . . . .	10
<b>3. Specifikáció és implementáció</b>	<b>11</b>
3.1. Specifikáció . . . . .	11
3.2. Architektúra . . . . .	14
3.2.1. Struktúra . . . . .	14
3.2.2. Folyamatok . . . . .	17
3.3. Implementáció . . . . .	20
3.3.1. Fejlesztés folyamata . . . . .	20
3.3.2. Rendszer szoftverkomponensei . . . . .	24
<b>4. Használati útmutató</b>	<b>26</b>
4.1. Telepítés . . . . .	26
4.2. Tesztelés . . . . .	26
<b>5. Összefoglalás</b>	<b>31</b>
<b>Irodalomjegyzék</b>	<b>34</b>

## HALLGATÓI NYILATKOZAT

Alulírott *Csортán Szilárd*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. december 9.

---

*Csортán Szilárd*  
hallgató

# 1. fejezet

## Bevezetés

Napjainkban egy modern alkalmazás fejlesztése komoly komplexitással jár, ennek hatására új szoftverfejlesztési módszertanok jelennek meg az informatika világában. Egy ilyen módszertan a *modellalapú szoftverfejlesztés*, aminek segítségével egy rendszer fejlesztési ciklusa és felépítése transzparenssé válik, a komplexitás könnyebben kezelhető és az üzleti oldal is átláthatja a fejlesztői oldal munkáját. Ezzel párhuzamosan a felhő alapú megoldások is egyre több helyen felbukkannak az iparban a flexibilitásuk, hatékonyságuk és stratégiai értéküknek köszönhetően. Ezen dolgozat keretein belül bemutatom, hogyan lehet egy létező modellezési eszközt elérhetővé tenni felhő alapú szolgáltatásként.

A modellalapú szoftverfejlesztés megoldásokat nyújt a modern rendszereknek az exponenciálisan növekedő komplexitásuk kezelésére. Különböző absztrakciós szinteken tudjuk az alkalmazásunk felépítését és működését vizuális elemekkel leírni. A modellek négy főbb perspektívából tudják a rendszert leírni[6]:

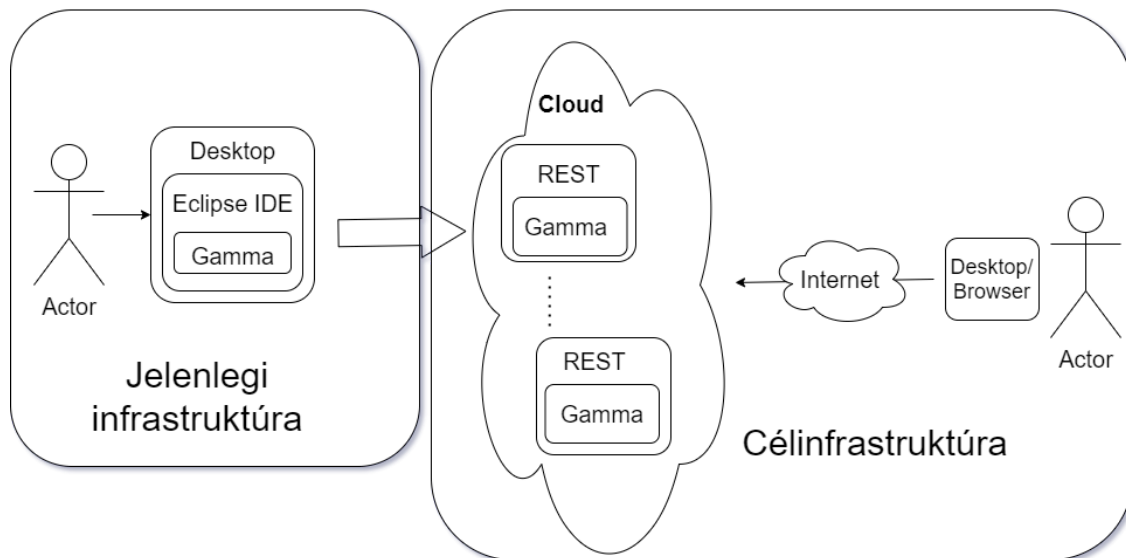
- Külső perspektíva, ahol a rendszer kontextusát és környezetét írjuk le
- Belső perspektíva, ahol a rendszer belső komponensei közti és a környezettel való kapcsolatot írjuk le
- Strukturális perspektíva, ahol a rendszer felépítését és a feldolgozott adatok strukturáját ábrázoljuk
- Viselkedési perspektíva, ahol a rendszer viselkedését és különböző eseményekre való reakcióját részletezzük

A Gamma keretrendszer [9] (Gamma Állapotgép Kompozíciós Keretrendszer) egy olyan eszköz, amivel komponens alapú reaktív rendszereket tudunk modellezni, verifikálni vagy akár kódot generálni.

Reaktív rendszernek nevezzük azt az architektúrális stílust, amely segítségével lehetőségünk van különálló alkalmazások egységként való kezelésére úgy, hogy ezek a komponensek továbbá képesek egymás eseményeire és a környezetükre reagálni.

A keretrendszer Yakindu (Yakindu Statechart Tools) alapokon készült, ami egy nyílt forráskódú állapotgépeket modellező eszköz. A gamma ezt továbbviszi és egy magasabb modellezési réteget biztosít a felhasználók számára, amely segítségével hierarchikus állapotgép hálózatokat tudnak kialakítani. A Gamma képes egyszerű állapotgépek és komplex állapotgép hálózatok modell verifikációjára, mindehhez felhasználja az UPAAL-t ami egy modell ellenőrző eszköz. Továbbá, a Gamma segítségével lehetőségünk van a teljes rendszerhez kódot generálni, jelenleg a Java nyelv támogatott.

A Gamma által nyújtott műveletekre manapság a modellalapú fejlesztések elterjedése miatt egyre nagyobb igény van, ilyen műveletek a kódgenerálás, modelltranszformáció



1.1. ábra. Gamma mint szolgáltatás

vagy modellellenőrzés. Továbbá, a korszerű eszközök közt a felhő alapú megoldások nagyon elterjedté váltak. Az említett funkciók arányos erőforrás igénye és a folytonos integráció lehetősége miatt, ideális lenne ha a két világot ötvöznénk.

A Gamma egy Eclipse-alapú eszköz, így jelen formájában nem tud felhőben futó szolgáltatásként létezni. A feladat ennek megfelelően az, hogy az alkalmazás szolgáltatásként is használható részeit parancssoron keresztül, illetve OpenAPI segítségével webes szolgáltatásként is elérhetővé tegyünk.

A fentebb leírtak alapján a Gamma egy Eclipse IDE-n belüli eszköz, ennek a transzformációját webes szolgáltatásba az 1.1 ábra mutatja be.

Alapvetően a Gammát mint Eclipse IDE Plug-in ki kell emelni, hogy egy fekete szoftver doboz legyen, ami bemeneti paraméterként kap egy műveleteket leíró fájlt, ezután legyártja a különböző műveletek által definiált eredmény fájlokat. A gamma fölé kell építeni egy webes kiszolgálót, ami képes bejövő kéréseket értelmezni, majd átfogalmazva tovább küldi a Gamma komponensnek. A megoldás során olyan mellék/segítő komponenseket is megkell tervezni és implementálni, amelyek az adatok átalakításáért vagy mozgatásáért felelősek.

A felhő világban definiáltak alapján a rendszerünk egy **Software as Service (SaaS)** komponens lesz, amely megoldási szolgáltatás a leggyakrabban használt, röviden annyit foglal magába, hogy egy alkalmazást teszünk elérhetővé felhasználók számára az internet segítségével. A rendszerünk felhőbe való integrációja mögötti igényt az alábbi három kategóriába sorolt előnyhalmaz magyarázza meg [13] [2]:

- **Flexibilitás** : Igény szerint skálázható, publikus és privát adattárolás lehetősége, vezérlési lehetőségek (egy szervezet eldöntheti, hogy milyen szinten szeretné kontrollálni az igényelt szolgáltatását), gazdag portfólió (számos létező eszköz és megoldás közül választhatunk) és nem utolsósorban biztonsági szempontból is rengeteg létező vagy új megoldást lehet implementálni az adatok titkosításához
- **Hatékonyság** : Elérhetőség - bárholnan elérhető egy szolgáltatás az interneten keresztül, idő megtakarítás szempontjából fejlesztők gyorsabban tudnak dolgozni, Hardware biztonság - egy esetleges hardware meghibásodás során nem veszítünk

adatokat, anyagi megtakarítás - vállalatoknak nincs szüksége szerverekre és más hasonló felszerelésre

- **Stratégiai érték** : Számos terhet vesz le egy felhőalapú szolgáltatás a szervezetek válláról, így több idő marad a fejlesztésekre, mindig friss az igénybe vett szolgáltatás verziója. Az elérhetőség lehetővé teszi, hogy a világ bármilyen részéről emberek közösen tudjanak dolgozni ugyanazon a problémán

Mivel az alkalmazásunk egy webes szolgáltatás lesz így ezzel az aspektussal járó pozitívumok is alátámasztják a rendszer piaci igényét.

Egy webes szolgáltatás lehetővé teszi, hogy a meglévő szoftverünket elérhetővé tegyük a világ számára[17]. HTTP protokollt használva nem csak kliensek felé oszthatjuk meg, hanem különböző más létező szoftver komponensekkel való kommunikációt is kitudunk alakítani.

Számos előnye van. Rengeteg már létező megoldási minta létezik, így nehéz elakadni. Jól előredefiniált protokollok vannak meghatározva, így két teljesen különböző ipari ágakból származó rendszerek közti kommunikáció is kialakítható.

A szakdolgozat keretén belül a fentebb leírt megoldás specifikációját, fejlesztési folyamatát, tesztelési útmutatóját és továbbfejlesztési lehetőségét részletezem.

A következő fejezetben bemutatom az általam választott technológiákat amelyekkel a feladatot megoldottam, itt bemutatásra kerülnek a OSGi, Eclipse RCP, REST API stb. fogalmak. Továbbá, a feladatot specifikálni fogom, megfogalmazott követelményeket mutatok be és végigvezetem a fejlesztési folyamaton és felmerülő nehézségek megoldásában az olvasót. Ezt követően bemutatom a fejlesztést egy konkrét példa segítségével, zárásként pedig a továbbfejlesztési lehetőségekről fogok beszámolni.

## 2. fejezet

# Háttér ismeretek

### 2.1. Modern technológiák

A megoldás folyamán számos modern technológia bevezetésére, implementációjára volt szükség, ezek közül a legfontosabbak:

#### 2.1.1. OSGi

Az OSGi (Open Services Gateway Initiative) [14] egy Java keretrendszer, amely segítségével moduláris szoftvereket és könyvtárakat tudunk fejleszteni.

A technológia alapvetően egy specifikáció halmazból, referencia implementációkból és minden specifikációhoz tartozó teszt halmazból áll össze, mindezek együtt leírnak egy dinamikus moduláris rendszert. A bevált szolgáltatás modellje lehetővé teszi, hogy az infrastruktúra és az alkalmazás komponensek úgy lokálisan, mint hálózaton keresztül is kommunikáljanak egymással, hogy egy koherens és konzisztens architektúrát alkossanak.

Az ipar bármely területén megtaláljuk az IoT-tól kezdve a telekommunikációs megoldásokig, viszont számos esetben nincs kiemelve, azaz ritkán van feltüntetve a használt technológiák listáján.

Kompozícióilag két részből áll az OSGi. Az első egy olyan specifikáció, ami a moduláris egységeket írja le, ezeket bundle-knek vagy plug-in-eknek nevezzük. A specifikáció leírja, hogy milyen infrastruktúra szükséges az említett bundle-k számára és, hogy egyes bundle-k milyen életcikluson mennek át (Telepítve, Elindítva, Aktív, Elakadt, Felfüggesztett), továbbá meghatározza, hogy milyen kapcsolat lehet egyesek bundle-k közt és ezek hogyan kommunikálhatnak egymással. A második pedig egy JVM szintű nyilvántartó, ami-be minden bundle tud bejegyzéseket tenni, hogy neki milyen publikálni kívánt objektumai vannak.

Az OSGi a fejlesztési ciklus majdnem minden területén csökkenti a komplexitást. Kódírás és tesztelés jelentősen kevesebb erőforrást fog elnyelni, támogatja az újrahasznosítást (létező komponenseket egyszerűen fel lehet használni egy új modulban). Az üzemeltetési feladatokat is könnyíti, lehetővé teszi a moduláris frissítést vagy telepítést, nem kell minden funkció bevezetésekor az egész rendszert leállítani. Könnyebben lehet lokalizálni egyes hibákat, így a támogatói feladatkör is egyszerűsödik. Az alkalmazásunk moduláris függőségi fájában egyszerűbben tudunk bizonyos elemeket bevezetni, helyettesíteni vagy akár kivenni.

Eclipse világban az Equinox [5] OSGi implementációt lehet használni, a fentebb részletezett szolgáltatásokat és szükséges infrastruktúrát alakítja ki számunkra.



### 2.1.2. Eclipse Plug-in fejlesztés

Eclipse egy nyílt forráskódú, platform független fejlesztői környezet (IDE). Egy alap munkaterület (workspace) és egy plug-in rendszert tartalmaz, aminek segítségével könnyen tesztre szabhatóvá lehet tenni. Az utóbbit használja ki a Gamma is és az általa használt mellék komponensek is.

Az Eclipse IDE egy speciális Eclipse alkalmazás ami lehetővé teszi a fejlesztőknek, hogy egyedi alkalmazásokat tudjanak fejleszteni. Egy Eclipse alkalmazás egy vagy több szoftver komponensből tevődik össze ezeket nevezzük plug-in-eknek [3]. Ezek a modulok bővíthetők vagy felhasználhatóak más komponensek létrehozására. Összefoglalva, az Eclipse IDE egy több plug-in-ekből (JDT) álló Eclipse alkalmazás.

Lehetőségünk van az így létrehozott alkalmazásainkat összecsomagolni és az Eclipse IDE-től függetlenül futtatni és tesztelni (**headless Eclipse**). Ehhez definiálnunk kell egy termék leírást (product configuration) [12], aminek tartalmaznia kell az alkalmazásunk plug-in-jét, minden felhasznált függőséget és minden más olyan konfigurációt ami szükséges ahhoz, hogy egy külön komponenseként tudjuk hordozni a szoftverünket. Ilyen konfigurációk, az alkalmazás belépési pontja (extension point), komponensek indulási sorrendje, esetleges ikonok és függőségi fa, ami meghatározza, hogy milyen komponens mitől és hogyan (kötelező, opcionális) függ valamilyen más modultól.

A termékleírásunk egy alkalmazás osztályra kell mutasson ami alapértelmezetten két fajta lehet:

```
org.eclipse.ui.ide.workbench - IDE alapú alkalmazások
```

```
org.eclipse.e4.ui.workbench.swt.E4Application - RCP alkalmazások, függetlenek az IDE-őtől
```

Ezekről el lehet térni és egy teljesen egyedi alkalmazást írni, ehhez a következő interfészt kell implementálni és behivatkozni a termék konfigurációnkba :

```
org.eclipse.equinox.app.IApplication
```

A dolgozatban leírt fejlesztés ezzel a megoldási opcióval lett megvalósítva.

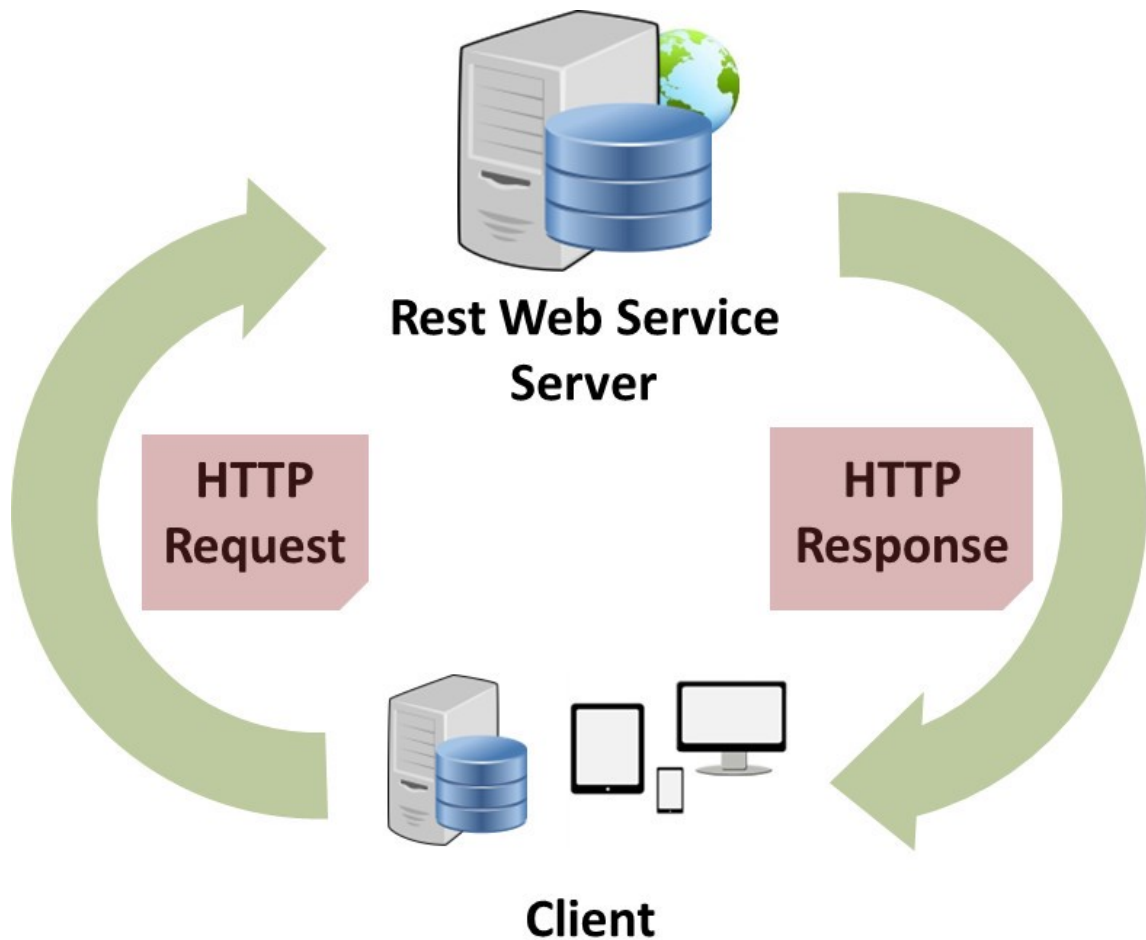
### 2.1.3. REST API

Az internet megszületése óta, mikor Charley Kline és Bill Duvall összekötöttek két számítógépet és elküldték egymásnak az "L" és "O" karaktereket, számos internet alapú adatmozgatási megoldás született ilyenek a CORBA, RPC vagy SOAP. Ezek mind nagyon komplex mechanizmusok és a mai világban elavultnak számítanak. Itt jön szóba a REST a legújabb iteráció az internetes adatszállítási megoldások körében. [4]

REST vagy **RE**presentational **S**tate **T**ransfer egy tervezési stílus, amellyel különböző rendszerek közti kapcsolatot tudunk megvalósítani. Olyan rendszereket, amelyek megfelelnek a REST szabványnak RESTful rendszereknek is nevezünk, alapvetően ezeket két fő jellemző írja le, az első, hogy nem tartják nyilván az egyes kérések állapotát, a második pedig az, hogy elkülönítik a kliens és szerver fogalmakat.

Ahhoz, hogy egy szolgáltatást RESTful-nak nevezzünk meg kell felelni egy szabványnak ami hat alapelvből áll [11] [15].

**Szerver és Kliens szeparáció** a tervezési minta előírja, hogy a szerver és a kliens két független és önállóan egységként létező komponens kell legyen. Ez lehetővé teszi, hogy a két rendszerünket külön tudjuk fejleszteni, telepíteni, javítani, karban tartani. Bármilyen kódmódosítást kell végezni bármelyik oldalon a másikat nem befolyásolja mind addig amíg az interfész nem módosul, ilyenkor a szerver egy master szerepet vesz át és az elvégzett módosítást kezelni kell kliens oldalon is. Ez az elv támogatja a moduláris gondolkodást, ezzel flexibilissé válik a komplex rendszerünk. Továbbá, skálázhatósági lehetőségek is felmerül-



**2.1.1. ábra.** REST egyszerű architektúra [18]

nek, hiszen az erőforrás igényesebb komponens alatti erőforrásokat igény szerint bővíthetjük. A szétválasztás lehetővé teszi, hogy akár egy serverhez több kliens is csatlakozzon, így sokszínű rendszereket tudunk létrehozni. Ennek egy vizuális reprezentációja a 2.1.1 ábrán látható.

**Állapot nélkülség** - Ez az alapelv annyit von magában, hogy a server semmilyen információval nem rendelkezhet a kliens állapotáról ennek a kijelentésnek viszont fordítva is teljesülnie kell, azaz a kliens se ismerheti a server állapotát. Minden küldött/kapott üzenet egységként lehet kezelni, független attól, hogy a múltban milyen jellegű üzenet váltások történtek. Ennek eredménye az, hogy a csak kliens tárolja a saját adatainak az állapotát (nem a server adatait).

**Egységes interfész** - Ez az alapelv négy szabályt határoz meg:

1. A kliens által kezdeményezett kérésnek tartalmaznia kell a fent definiált erőforrásnak az azonosítóját. Az erőforrást (egyed/objektum) hívhatjuk a webes kontextus alanyának is.
2. A server válasza elegendő információt kell tartalmazzon az erőforrásról, pontosabban annyit, hogy a kliens tudja majd módosítani ezt.

3. Minden egyes hívás az API felé részletes és tartalmilag teljes kell legyen. Továbbá a szerver válaszában is tartalmaznia kell elegendő információt, úgy, hogy a kliens értelmezni tudja ezt.
4. Az utolsó szabálytól gyakran eltekintenek a fejlesztők, ez azt mondja, hogy a válasz, ami a szerverről érkezik tartalmazhat, olyan útmutatókat a kliens számára amivel az alkalmazás állapotát módosítani tudja. Például, ha egy objektumot lekér a kliens az API-tól, akkor a szerver a válaszba becsomagolhat olyan mellékes információkat amelyek azt írják le, hogy milyen módon módosítható a visszaadott objektum.

Az egységes interfész lehetővé teszi, hogy a kliens típusától (böngésző, alkalmazás) eldöntsük tekinteni a szerver oldalán.

**Cache lehetőség** - Cache támogatást úgy lehet elérni, hogy az adatokat verziózzuk, ha a kliens lekér egy adatot akkor valamilyen metaadatban tároljuk azt is, hogy ez milyen verziójú, így ha ugyanazt az objektumot szeretné lekérni, előtte tudja ellenőrizni lokálisan, hogy már a legfrissebb adattal rendelkezik és nem kell a szervert megint megszólítani. Ezzel terheléscsökkentést tudunk elérni.

**Rétegelt architektúra** - A szerver és az erőforrást lekérő kliens között számos más rendszer helyezkedhet el, ezek többfajta lehetnek de legtöbbször a biztonsági, cache, terhelés egyensúlyozó rétegekkel találkozhatunk.

**Igény szerinti kód (opcionális)** - Az egyedüli elv ami opcionális, annyit von magában, hogy a kliens bármikor kérhet kódrészleteket a szerverről és ez a válaszban valamilyen HTML vagy szkript formátumba becsomagolja és elküldi ennek, amit majd a kliens futtatni tud. Az alapelv teljesülése nélkül is tud egy rendszer RESTful lenni.

A fentebb többször említett kérés/válasz fogalmakat a REST világban HTTP kérésekbe csomagoljuk, ezek tipikusan a következőket kell tartalmazza [1] :

1. egy HTTP ige, ami leírja a művelet típusát, jelenleg használt igék:
  - (a) **GET** erőforrást tudunk lekérni, nagyon fontos, hogy nem módosíthat állapotot a szerveren
  - (b) **POST** erőforrás létrehozásra használják, többszörös hívás esetén különböző erőforrásokat fog létrehozni
  - (c) **PUT** már létező erőforrások állapot frissítésére használják, többszörös hívás ugyanazt kell, hogy eredményezze
  - (d) **DELETE** mint ahogy a neve is sugallja, erőforrás törlésre használják
  - (e) **PATCH** parciális frissítési műveletekre tudjuk használni
  - (f) a maradék HTTP igeiket nem szokták gyakran implementálni a REST világban
2. fejléc, ami metaadatokat tartalmaz az üzenetről, többek közt olyan jellegű információkat, mint az üzenet hossza, a tartalom típusa, a válasz HTTP kód száma és üzenete.
3. az erőforrás elérési útja
4. opcionális üzenettörzs, ami további adatot tartalmazhat

#### 2.1.4. OpenAPI és Swagger

**OpenAPI Specifikáció** [10] egy API leíró szintaktika REST API-k számára. A specifikáció YAML és JSON formátumban is írható, az így létrehozott fájl a teljes API-t meghatározza, jellemzően a következőket tartalmazza:

- Endpointok halmazát és az ezeken meghatározott műveletek leírását, mint például GET /api/students, ami a tanulók listáját adja vissza vagy a POST /api/students, ami a kliens oldalról felküldött adatok alapján létrehoz egy tanulót a szerver oldalon.
- Műveletek bemeneti és kimeneti paraméterét
- Autentikációs metódusokat
- Licenc, felhasználói feltételek vagy bármilyen adminisztratív információ ami az API-hoz tartozik

Az OpenAPI specifikációt korábban Swagger specifikációnak hívták, amíg a SmartBear Software felajánlotta az OpenAPI iniciatívának. Azóta az OpenAPI 3.0 verziónál tartunk.

A Swagger [16] egy olyan eszköztár ami az OpenAPI köré épült, hogy segítse a REST API tervezést, implementálást, tesztelést. Számos ilyen eszköz tartozik ide, ezek közül a legfontosabbak:

- **Swagger Editor** Egy böngészőből elérhető felhasználó barát felületet biztosít, ahol lehetőségünk van megírni a specifikációnkat. Számos előnyt kínál egy hagyományos szerkesztőhöz képest, ilyenek a szintaktikai ellenőrzések, bizonyos REST API szabályok betartását is megköveteli, pl: egy GET metódusnak nem lehet törzse.
- **Swagger UI** Egy olvasható, részletes és vizuálisan látványos dokumentációt generál az Editorban elkészített API specifikációkhoz, a 2.1.2 ábrán egy ilyen lehet megtekinteni.
- **Swagger Codegen** Kliens oldali könyvtárakat és szerver oldali keretet tud generálni.

A specifikációnak tehát rengeteg előnye van, többek közt az egységes API leírás, ami platform független, a kódgenerálás miatt több erőforrás marad az üzleti logika implementálásra vagy az automatizált dokumentáció készítés.

#### 2.1.5. Fontosabb Java könyvtárak

A Vert.x [7] egy eszköztár, amelynek segítségével JVM alapú alkalmazásokat tudunk készíteni. Nagyon flexibilis, azaz nagyon széles az alkalmazás típus spektrum, ahol használni lehet. HTTP/REST mikroszolgáltatások, komplex webes alkalmazások vagy eseményvezérelt back-end fejlesztése során is használható.

Flexibilitása miatt a piac számos területén használják, például valós idejű játékokban vagy banki rendszerekben.

JVM alapú programozási nyelvek bármelyikében használható (Java, Kotlin, JavaScript, Groovy, Ruby, Scala).

OpenAPI specifikáció integrálását is támogatja. Nagyon egyszerűen lehetőségünk van behatározni az API-t leíró fájlunkat, ami alapján a Vert.x legenerál egy interfész réteget, amely mögé nagyon egyszerűen letudjuk implementálni az üzleti logikánkat.

# Gamma Wrapper API

1.0.0 OAS3

An API that allows users to use the Gamma framework via http requests

[Contact Csörtán Szilárd](#)

Servers

<https://dev.gammaframeworkor...> ▾

Dev Server

default ▾

POST

/gamma/addworkspace



POST

/gamma/addproject/{workspace}



POST

/gamma/api/{workspace}/{projectName}/{filePath}



PUT

/gamma/getresult/{workspace}/{projectName}



DELETE

/gamma/deleteproject/{workspace}/{projectName}



2.1.2. ábra. Gamma OpenAPI Specification

## 2.2. Gamma

A Gamma egy olyan eszközhalmaz, amely segítségével komponens alapú reaktív rendszereket tudunk modellezni. Több funkcióval rendelkezik [8], ezek kerülnek rövid bemutatásra ebben a fejezetben.

### 2.2.1. Gamma funkciók

Támogatott mérnöki modellek integrációja **modell transzformáció** segítségével történik. Ez a funkció egy létező mérnöki modellt fordít le a Gamma állapotgép nyelvére, ezzel a felhasználót segíti, mivel automatizálva történik a forrás nyelv és a gamma közti leképezés. Jelenleg a Yakindu és UPAAL modellezési nyelveket képes a Gamma transzformálni, viszont a plug-in koncepció további nyelvek leképezését is lehetővé teszi.

A **validáció** egy statikus analízis technika, mely visszajelzést ad a létrehozott modellek szintaktikai helyességéről. A Gamma az önálló modelleken és a kompozit állapotgép komponenseken is végez validációt.

A Gamma képes Java programozási nyelvre **automatizáltan kódot generálni**. Interfész definíciókat a Gamma interfészekből generálja a funkció, a komponens implementációkat pedig a kompozit állapotgép modellezés során definiáltak alapján készíti el.

Két **verifikáció** funkcionalitást támogat a Gamma :

- formális verifikáció, amely formális modellezési nyelvek integrációjával valósul meg. Jelenleg a Gamma keretrendszerbe az UPAAL van integrálva.
- visszavetítés, amely felhasználja a formális verifikáció eredményét és ezt felhasználva generál egy végrehajtási láncot amely alátámasztja vagy megcáfolja az egyes rendszer követelményt.

A Gamma az UPAAL modell ellenőrzőt felhasználva képes **teszteket generálni**, ehhez létrehoz egy absztrakt Gamma modellt, amely alapján legenerálja a teszt eseteket.

### 2.2.2. Gamma hiányosságok

A Gamma egy Eclipse IDE-ben létező keretrendszer, így jelentős kihívást jelent, hogy egy felhasználó kipróbálja esetleg használja. Kezdve az Eclipse világ rejtelmétől a különböző függőségi nehézségekig számos hibalehetőség lép föl, ha használni szeretnénk az alkalmazást. Továbbá, vannak olyan modellek amelyek több ezer akár millió nagyságrendű állapottal rendelkeznek, ilyen modelleken a fentebb leírt műveletek futtatása igenis erőforrásigényes. Sajnos az IDE-hez kötöttség nem segít ezen problémák megoldásában, nehéz a skálázhatóság és a disztribúció kérdésekre választ kapni jelen kontextusban.

## 3. fejezet

# Specifikáció és implementáció

Ez a fejezet tartalmazza a 2.2.2 fejezetben leírt hiányosságokra adott megoldásom specifikációját, architektúra tervét és implementációját.

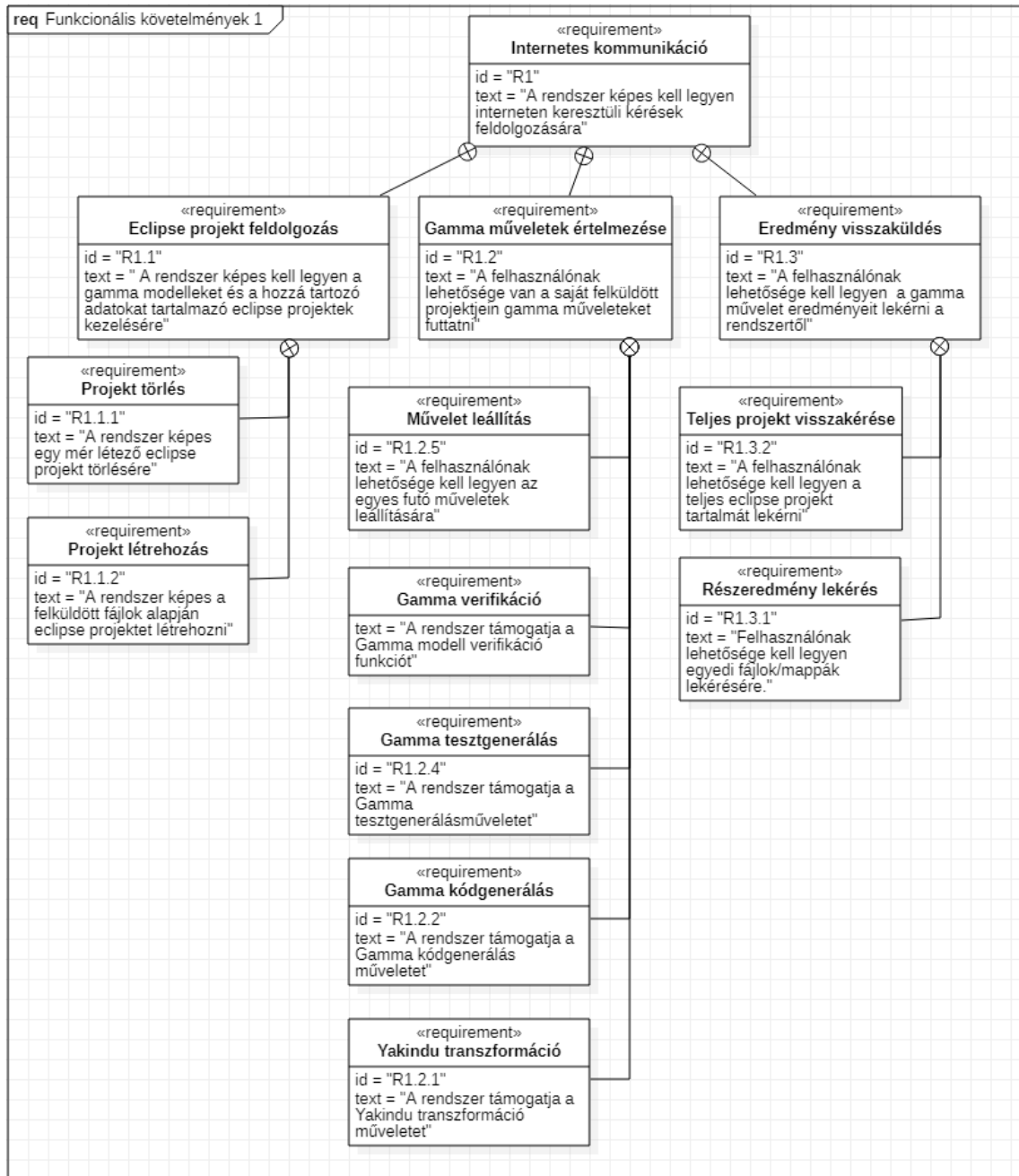
### 3.1. Specifikáció

A specifikálást a rendszert meghatározó követelmény halmaz kialakításával kezdjük. A követelményeket alapvetően két csoportra lehet osztani, funkcionális és nem funkcionális. Kulcsfontosságú absztrakt követelményeket az alábbi lista foglal össze, viszont nem tartalmazza a teljes körű követelmény hierarchiát, ezt a 3.1.1 és a 3.1.2 ábrákon lehet megtekinteni.

- A rendszer képes kell legyen a gamma modelleket és a hozzá tartozó adatokat tartalmazó eclipse projektek kezelésére.
- A felhasználónak lehetősége kell legyen felküldeni a rendszer számára a saját eclipse projektjeit.
- A felhasználónak lehetősége kell legyen a saját felküldött projektjein gamma műveleteket futtatni.
- A felhasználónak lehetősége kell legyen a gamma művelet eredményeit lekérni a rendszerünktől.
- A rendszer képes kell legyen felhasználók azonosítására.
- A rendszer képes kell legyen inaktív projektek automatizált törlésére.

A további nem funkcionális követelményeket négy kategóriába soroljuk:

- **Megbízhatóság:** Egy projekten nem futtathatunk két különböző Gamma művelet halmazt
- **Biztonság:** Minden felhasználó csak a saját projektjeit szerkesztheti / saját projektjein futtathat gamma művelet halmazokat / A rendszer képes kell legyen kiszűrni az egyes rosszindulatú felhasználók által megadott fájl elérési utakat /
- **Teljesítmény:** A rendszer képes kell legyen különböző projekteken ugyanabban az időben Gamma műveletek futtatására. / A rendszer skálázható kell legyen. / A rendszernek nem szabad fölösleges adatot tárolnia. / A rendszer muszáj töröljön minden olyan adatot, amely neki vagy a felhasználó számára nem releváns. / A rendszer képes kell legyen asszinkron módban működni.



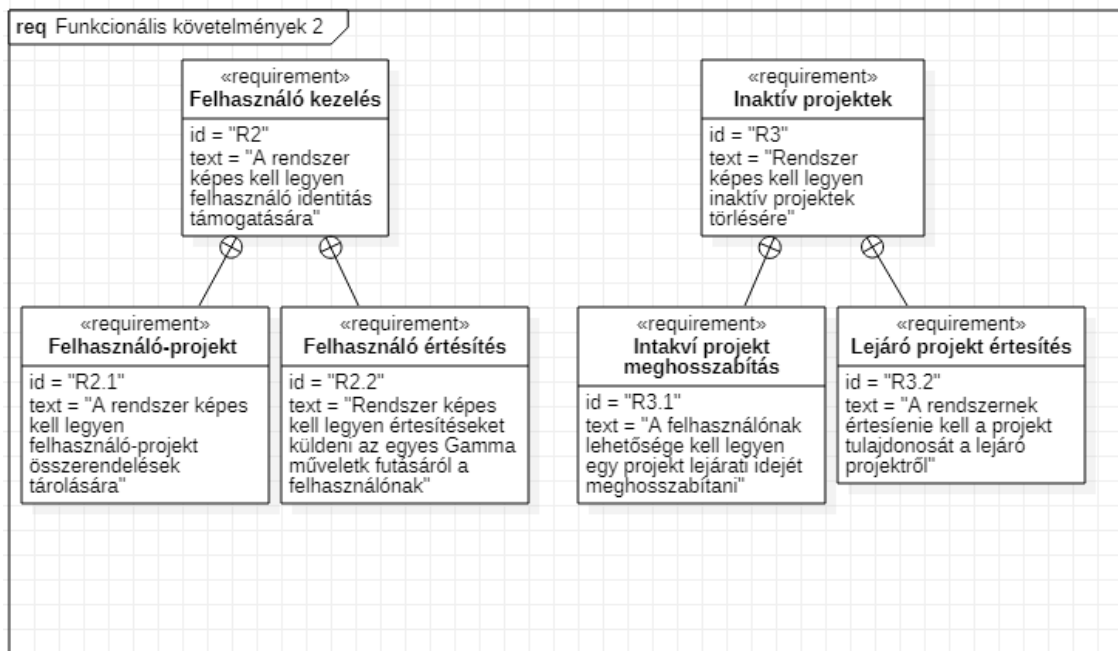
3.1.1. ábra. Követelmény hierarchia 1

- **Felhasználhatóság:** A rendszer megfelelő, HTTP szabvány által előírt válaszokat adni az egyes kérésekre. / A rendszer beszédes értesítéseket kell küldjön a projekt tulajdonosának az egyes Gamma műveletek futási állapotáról.

A fentebb leírt követelmények alapján bizonyos technológiai döntéseket kellett hozni a projekt iniciális fázisaiban. Ahhoz, hogy az alkalmazásunk interneten keresztül is elérhető legyen egy webszervert kellett kialakítanunk, ez lesz a rendszer belépési pontja. A webszerver a REST API modern architektúra stílus szabályait betartva lett kialakítva. Mindez a *R1.\** követelmény teljesítése teszi szükségessé.

Ahhoz, hogy az *R1.2.\** követelmény teljesüljön, a Gamma keretrendszert egy *headless Eclipse*-be kellett becsomagolni. Ezzel megtudjuk oldani azt a problémát, hogy a Gamma





3.1.2. ábra. Követelmény hierarchia 2

az Eclipse IDE-hez van kötve, az így becsomagolt keretrendszert parancssoron keresztül lehet elérni.

Az *R2.\** követelményt eclipse munkaterek (workspace) használatával teljesítjük. A rendszerünk biztosít munkatér létrehozás funkciót a felhasználói oldalon álló kliens szoftvernek, ehhez rendel egy egyedi azonosítót amit visszaküld a kliensnek és mostantól, ezzel az azonosító megadásával tud a kliens további funkciókat elérni. Mindezzel azt érjük el, hogy majdnem semmilyen információt nem kell tárolni a felhasználóról, ezt a feladatot rábízuk a kliens szoftverre.

Összefoglalva a követelményeket és a technológiai döntéseket, olyan rendszert tervezzük, amely a Gamma keretrendszert elérhetővé teszi a világ számára, oly módon, hogy közben több felhasználói rendszerlogikát támogat az adatok tárolásán és funkciók elérésén. Továbbá, olyan mellék funkciókat is biztosítania kell, mint inaktív projektek automatizált törlése vagy felhasználók értesítése.

A rendszerben érdekelt személyek listája (**stakeholders**), azaz olyan aktorok, akik valamilyen módon kapcsolatba léphetnek a rendszerrel:

- Komplex modellekkel foglalkozó **mérnökök** (NASA)
- **Egyetemi kutatók**
- Olyan **fejlesztő** aki integrálni akarja a rendszerünket egy harmadik párti alkalmazásba (MagicDraw)
- **Gamma fejlesztő**, aki erőforrás igényes funkciót akar tesztelni
- **Üzemeltető**, aki a rendszer komponenseit frissíteni tudja
- **Üzleti oldalú kolléga**, aki a rendszert mint szolgáltatás hirdeti/eladja

A rendszer jelen formájában nem tartalmazza a fentebb többször említett kliens komponenst, így precíz használati eseteket nehéz definiálni. Ennek ellenére a rendszert teljes

funkcionalitását lehet használni, tesztelni valamilyen API tesztelő szoftverrel, mint például PostMan. Olyan kéréseket tudunk küldeni az alkalmazás felé, amelyek teljes körűen szimulálják egy kliens szoftver viselkedését.

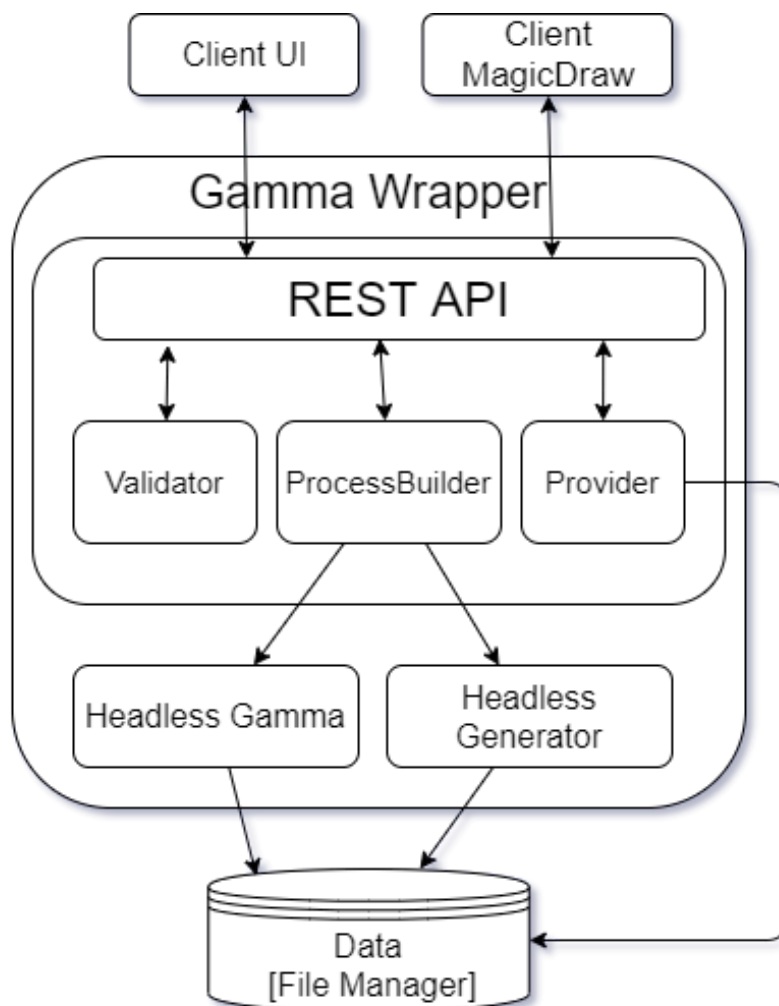
Az alapvető elérhető funkciók közé tartozik: a munkatér létrehozás, amely egy felhasználónak dedikált eclipse munkateret hoz létre, amin belül majd további műveleteket lehet végezni; eclipse projekt importálása archivált forrásból egy létező munkatérbe; az importált projektben szereplő, műveleteket leíró fájlok alapján gamma műveletek futtatása; a gamma műveletek által generált elemek visszakérése archivált fájlként; importált eclipse projekt törlése.

## 3.2. Architektúra

Ez a fejezet részletezi a specifikált rendszer struktúráját, továbbá áttekintünk fontosabb folyamatokat.

### 3.2.1. Struktúra

A rendszer felépítését a 3.2.1 ábra mutatja be. A továbbiakban részletezem az ábrán látható egyes komponenseket, milyen szerepük van és hogyan kommunikálnak más komponensekkel. Az ábrán látható egy nagyobb, mindent magába foglaló *Gamma Wrapper* nevezetű



3.2.1. ábra. Architektúra terv

modul, ez a rendszer amely megvalósításra került a félév során és amiről a dolgozat szól. A nevét a funkcionalitásáról kapta, magyarul *Gamma csomag*-nak is nevezhetnénk.

**REST API** Ez az alkalmazás belépési pontja, ez látható a világ számára. Önmagában egy Vert.x webszerver amely egy OpenAPI specifikációban<sup>1</sup> meghatározott *endpoint*-ok alapján épül fel. Endpointnak hívjuk azokat az egyedi belépési pontokat, amelyek meghívhatók az alkalmazáson kívül. A rendszer az alábbi endpointokat tartalmazza, a fontosabbak működését a következő fejezet részletesen bemutatja:

- POST `/gamma/addworkspace` eclipse workspace létrehozásának lehetőségét kínálja fel, minden további művelethez szükséges, a kérés nem tartalmaz további információt
- POST `/gamma/addproject/workspace2` eclipse projekt létrehozást teszi lehetővé a workspace paraméterből kiolvasott munkatérben, a kérés törzsében szerepelnie kell az archivált fájl, ami tartalmazza az importálni kívánt projektet
- PUT `/gamma/api/workspace/projectName/filePath` a munkatér és projekt páros alapján meghatározott Gamma műveleteket tartalmazó fájl futtatását teszi lehetővé. A *filePath* paraméter írja le, hogy a műveleteket leíró fájl hol van, ez relatív kell legyen és a *projectName* paraméterben tárolt projekt mappájában
- PUT `/gamma/stopprocess/workspace/projectName` a fentebb leírt folyamat leállítását teszi lehetővé, mivel egy projekt-munkatér pároson egy időben csak egy gamma művelethalmaz futhat ezért elég csak ezt a párost megadni
- PUT `/gamma/getresult/workspace/projectName` a gamma által generált fájlok visszakérését teszi lehetővé, ehhez szükséges, hogy a kérés törzsében utazzon, hogy milyen fájlokat/könyvtárakat szeretnénk visszakérni, az itt megadott elérések a projekthez képest relatívak kell legyenek, ha szerepel a `..` elérés akkor a projekt teljes tartalmát visszaküldjük
- DELETE `/gamma/deleteproject/workspace/projectName` eclipse projekt törlését teszi lehetővé
- PUT `/gamma/extendexpiration/workspace/projectName` alapértelmezetten minden projekt automatikusan törlésre kerül 30 nappal a létrehozás után, ezzel az endpointal további 30 nappal lehet ezt az időtartamot meghosszabbítani, az URL-ben utazó munkatér és projekt név párossal tudjuk eldönteni, hogy melyik projektet kívánja a felhasználó meghosszabbítani.

A komponens további feladatok is ellát, ilyenek a megfelelő HTTP válasz összeállítása, a szerverre küldött fájlok automatikus mentése vagy a más komponensek vezérlése. Egy időzítő is ide tartozik, ezzel ellenőrizzük a projektek lejáratát dátumát és, ha ennek eljön az ideje akkor egy törlést fog indítani.

**UI és MagicDraw** Mivel a rendszerünk egy szerver oldali komponens, ezért önmagában nem tudja bárki használni. Az ábrán feltüntetett kliens rendszereknek kell a felhasználói felület szerepét betölteni, ilyen lenne egy speciális UI ami az alapvető funkcióknak egy grafikus felületet biztosít és kontextust ad az alkalmazásunknak. Továbbá, tervben van a Gamma integrációja a MagicDraw<sup>3</sup> modellező eszközbe, a rendszerünk ezt az integrációt megkönnyítené.

<sup>1</sup>OpenAPI specifikáció [https://app.swaggerhub.com/apis/szlennn/GAMMA\\_WRAPPER3.0/1.0.0](https://app.swaggerhub.com/apis/szlennn/GAMMA_WRAPPER3.0/1.0.0)

<sup>2</sup>Minden döntött betűs rész egy URL-ben utazó paraméter

<sup>3</sup>MagicDraw modellező eszköz <https://www.nomagic.com/products/magicdraw>

**Headless Generator** A Gamma önmagában nem képes eclipse munkatér és projekt létrehozásra ezért kellett egy olyan komponens, ami előkészít egy környezetet, amiben futtathatunk Gamma műveleteket, ennek a neve *Headless Generator*. Két fő funkciója van: eclipse munkatér (workspace) létrehozás és eclipse projekt importálás egy megadott munkatérbe. Mindkét funkcionalitást ugyanaz a folyamat valósítja meg annyi eltérésben, hogy ha a komponens kap egy archivált fájl elérést akkor tudni fogja, hogy ezt importálnia kell, mint eclipse projekt. Egy *headless runtime eclipse*-ként van megvalósítva és parancssorból lehet meghívni, két paramétert lehet megadni az első a *-data* amiben a workspace nevét kell megadni, ha egy nem létező workspace-t adunk meg akkor a komponens ezt létrehozza, ez egy kötelező paraméter. A második argumentum pedig a projekt archivált fájl elérése, ezt fogja kicsomagolni és regisztrálni a munkatérbe, ez az argumentum opcionális. Fontos kiemelni, hogy a forrás fájl már a munkatér mappájában kell legyen.

**Headless Gamma** Annak ellenére, hogy csupán egy endpoint meghívása során kerül működésbe a rendszer legfontosabb eleme. A Headless Generator-hoz hasonlóan egy *headless runtime eclipse*-ként lett kialakítva és ugyanúgy parancssorból lehet elindítani. Három kötelező paraméterre van szüksége. (1) Létező munkatér, amely tartalmazza az átadott (2) projekt állományait. Az utolsó (3) paraméter egy a projekthez relatív elérés, ami a gamma műveleteket leíró fájlra mutat. A fájl *.ggen* kiterjesztésű kell legyen, és megfelelő sorrendbe kell tartalmazza a futtatni kívánt műveleteket és a futtatáshoz szükséges modell állományok hivatkozását, amelyek a projekten belül kell legyenek. A Gamma miután elvégezte a műveleteket tipikusan 3 mappába generálja le az eredményeket: *src-gen*, *test-gen*, *trace*. Ezek mind a projekt mappán belül találhatóak.

Az UPAAL telepítése egy alapvető követelmény a gamma megfelelő működéséhez, ennek az elérést a rendszer a környezeti változókból tudja kinyerni. Az OSGi előnyeit itt tudjuk kihasználni, mivel a headless gamma több száz *plug-in*-t igényel. A *plug-in*-ek egyével frissíthetőek, elméletileg nem kéne nagy akadályt okozzon egy új release telepítése.

**Process Builder** A REST webszerver és a headless komponensek közé biztosítani kell egy olyan réteget, ami a webes hívások paramétereit átalakítja parancssori argumentumokká és eltud indítani egy ilyen runtime eclipse-t, ezt a szerepet a *Process Builder* tölti be. Négy funkció esetén jelenik meg a végrehajtási sorban, (1) munkatér létrehozás, (2) projekt importálás archiv fájlból, (3) gamma művelet futtatás és (4) gamma művelet futtató folyamat leállítás. Minden funkció esetén lehetőség van asszinkron működésre, viszont jelenleg csak a (3) és (4) esetén van implementálva, a szinkron működést azzal az érvel lehet védeni, hogy csak abban az esetben küldjünk választ a kliensnek ha valóban a kért funkcionalitást eltudtuk végezni. Ez a gamma művelet futtatáskor azért nem egy élhető koncepció, mert vannak olyan modellek amelyek több ezer vagy akár több tíz ezer állapotból állnak, ezeknek a feldolgozása órákba is kerülhet így az asszinkron működés egy szükséglet. A gamma művelet leállítása pedig nem igényel semmilyen megvárast, nincs olyan eset amikor ez eltud akadni. A másik két esetben (1)(2) viszont megvárjuk amíg a munkatér vagy projekt létrejön, mert nem költséges műveletek, ezt a jövőben érdemes lehet átgondolni és esetleg átalakítani.

**Validator** A valamennyi bejövő kérések validációs lépéseken is át kell eszenek, ezt a *Validator* modul biztosítja. A REST webszerver vezérli, hogy a bejövő paraméterek közül melyek azok amelyek valamilyen validáción át kell eszenek. A legfontosabb ellenőrzési lépések közé tartoznak a munkatér létezés ellenőrzése, a projekt-munkatér páros vizsgálata és az átadott projekten futó gamma folyamat ellenőrzése. Az utóbbi kiemelkedően fontos, hiszen egy futás alatt levő projekttel nem végezhetünk semmit, csak leállíthatjuk. Ahhoz,

hogy a futás állapotát nyilván tudjuk tartani, be lett vezetve egy leíró json fájl amit minden projekt importálásánál létrehozunk, ez a fájl többek közt tartalmazza azt is, hogy jelenleg van-e futó folyamat a projekten. Ennek az állapotát a ProcessBuilder beállítja *inProgress*-be majd a HeadlessGamma futása végén frissíti *notInProgress*-re.

**Provider** A *Provider*-nek két feladat van, az első, hogy a REST webszerver által feldolgozott eredmény visszakérő endpoint-ban megadott relatív elérések alapján összecsomagoljon egy zip állományt, a másik pedig a projekt törlés. Az eredmény fájl összecsomagolásánál lehetőségünk van megadni egyedi fájlok elérését vagy akár teljes mappákat. Ide tartozik a teljes projekt mappa is, ezt a `"/` eléréssel tudjuk megadni, ha ez szerepel a kérésbe akkor a többi eredmény eléréssel nem is foglalkozik a rendszer, hiszen ez mindent tartalmazni fog.

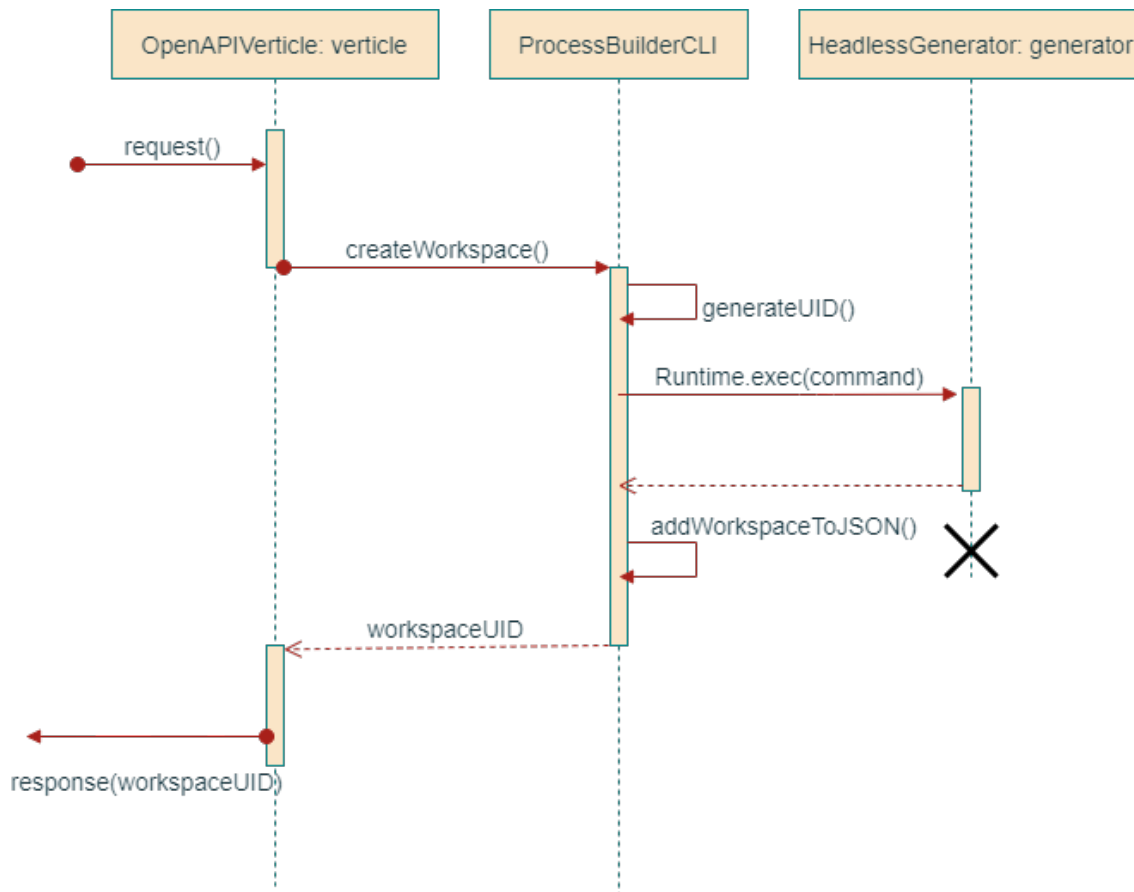
**Data** Az ábrán megjelenik a *Data* fogalom is, ez valójában a hoszt szerveren levő fájl kezelőt jelenti. Itt tároljuk a létrehozott munkatér és projekt párosokat és ezek teljes tartalmát, továbbá a metaadatokat tároló json fájlokat is. Egy adatbázis réteg bevezetésével a fájlban tárolt metaadatokat könnyebben tudnánk kezelni, sajnos a projektbe ez nem fért bele.

### 3.2.2. Folyamatok

Négy alapvető funkció részletes működését fogom bemutatni ebben a fejezetben, ezek sorra: munkatér létrehozás, projekt importálás, gamma műveletek futtatása és a futtatás eredmény lekérés. A folyamat bemutatja milyen modulokon megy végig az adat, továbbá milyen feldolgozások történnek egyes végrehajtási lépéseken. Mindezt szekvencia diagramokkal ábrázoltam. Az így bemutatott funkciók lefedik a rendszer legfontosabb felhasználói eseteit.

**Munkatér létrehozás** A workspace létrehozás szekvencia diagramja a 3.2.2 ábrán látható. Első fázisban bejön egy POST HTTP metódus az `addworkspace` endpointra, ennek a törzse üres és az URL-ben sem hordoz paramétereket. Látható, hogy az OpenAPIVerticle fogadja a kérést, ez lényegében a REST webszerverünk, mivel ez a legelső művelet, amit a kliens meg kell hívjon így nem kell semmilyen ellenőrzést végezni. A végrehajtási sor következő lépése a *ProcessBuilder*-ben van, itt a rendszer generál egy egyedi UID azonosítót, amit felhasznál a parancs kialakításhoz, amivel elindítja a *HeadlessGenerator*-t. A parancs (command) elkészítéséhez a *ProcessBuilder*-nek szüksége van a *headless eclipse* elérésére, ezt a *config.properties* fájlban lehet beállítani. A generator amíg elkészíti a workspace-t a *ProcessBuilder* vár, ha végezt akkor visszaadja az OpenAPIVerticle-nek a generált UID-ot ami továbbítja a kliens felé. Mikor a *headless eclipse* elvégezte a feladatot akkor teljesen leáll, ezt jelzi az X a folyamatábrán.

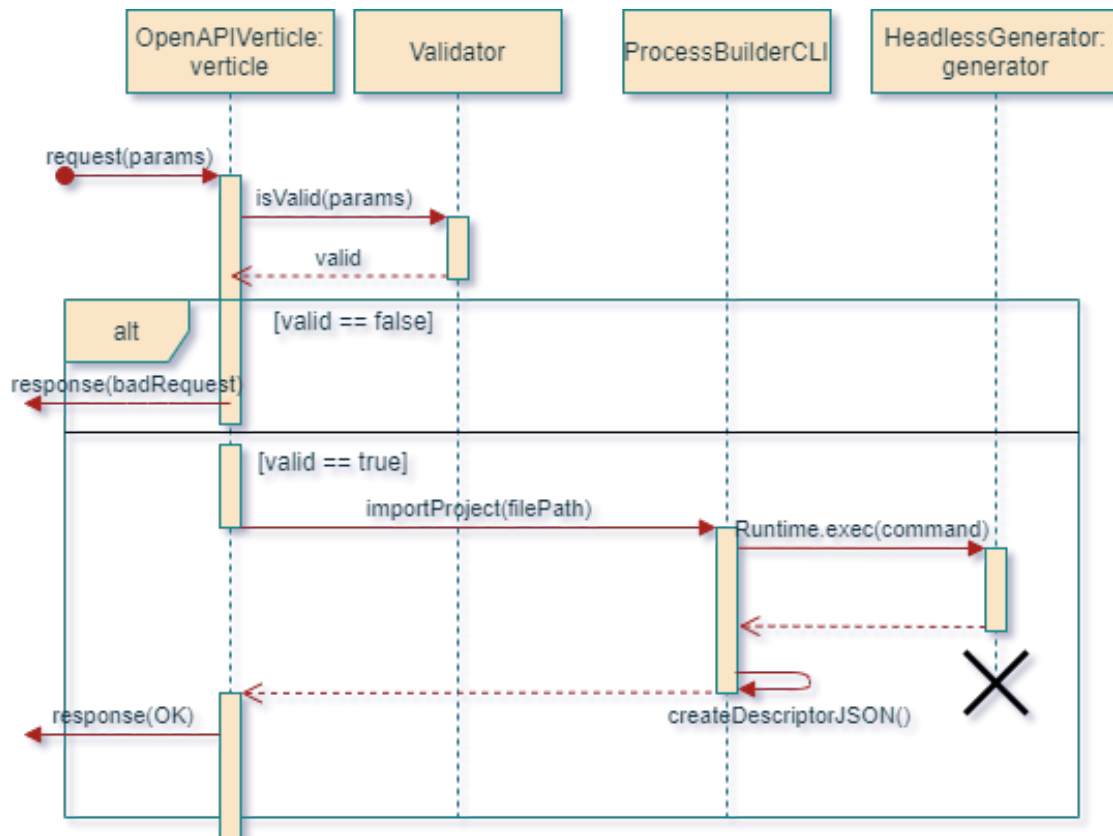
**Eclipse projekt importálás** A projekt importálás funkció szekvencia diagramját a 3.2.3 ábrán lehet megtekinteni. A folyamat megint a kérés érkezésével indul, ezt minden esetben az OpenAPIVerticle dolgozza fel, ez a kérés egy POST HTTP metódus ami az `addproject` endpoint-ra jön be. Ebben az esetben viszont már az URL-ben utazik egy munkatér azonosító amit korábban adtunk vissza a workspace létrehozáskor. Továbbá, a törzsben két attribútum szerepel: (1) *files* ez tartalmazza a projekt archivált állományait és (2) *contact* ez egy email cím, ami a projekt tulajdonosáé, erre későbbiekben értesítések küldésénél lesz szükségünk. A munkatér létezését a Validator komponenssel elvégezzük, a kapott válasz alapján két esetet különböztetünk meg, az első, ha nem érvényes a megadott workspace azonosító, ekkor egyből szólunk a kliensnek, hogy rossz a kérés. Viszont, ha ez helyes akkor tovább adjuk a kérést a *ProcessBuilder*-nek ami az előző esethez hasonlóan



**3.2.2. ábra.** Munkatér létrehozás szekvencia diagram

elindítja a **HeadlessGenerator**-t, de kiegészíti az argumentum listát a forrás fájl nevével, mindezek előtt a nyers fájlt a munkatér mappájába másolja. A generator elvégzi az import műveletet, majd visszaadja a futási jogot a **ProcessBuilder**nek, ami végezetül regisztrálja a munkatér-projekt párost és létrehoz egy leíró fájlt a frissen létrehozott projektben, ami tartalmazza, hogy mi a projekt neve, ki hozta létre (email cím), mikor hozta létre, mikor jár le és beállítja, hogy jelenleg nincs futás alatt, ezek után törli az iniciális forrás fájlt, hogy tárhelyet spóroljunk. Végezetül visszaadja a futási jogot a **verticle**-nek aki jelez a kliensnek, hogy minden rendben létrejött.

**Gamma műveletek futtatása** A gamma műveletek futtatását bemutató szekvencia diagram a 3.2.4 ábrán tekinthető meg. Ebben az esetben egy **PUT HTTP** kérés érkezik az *api* endpoint-ra, amit szintén az **OpenAPI** réteg dolgoz fel kezdetben. A törzsben nem szerepel semmi, viszont az URL-ben utazik a munkatér azonosító, projektnév és futtatni kívánt gamma műveleteket leíró .ggen fájl elérése. Ez a hármas egyedileg meghatározza az erőforrást. A validációs fázis komplexebb ebben az esetben, hiszen ellenőriznünk kell a munkatér-projekt páros létezését és egyediségét, mert egy projekt egyszer szerepelhet egy munkatérben, továbbá azt is meg kell vizsgálni, hogy a projekt, amit ez a páros leír jelenleg használva van-e valamilyen más gamma műveletet futtató headless eclipse által. A **Validator** döntése alapján, megint két esetet különböztetünk meg, ha valamiért nem felel meg akkor az **OpenAPIVerticle** a megfelelő hibakóddal és üzenettel visszaszól a kliensnek, hogy nem érvényes a kérés. Ha minden megfelel, akkor a futást átadjuk a **ProcessBuilder**-nek, ami asszinkron módon elindítja a gamma műveletet futtató headless eclipse-t. Ezek után egyből frissíti a projektet leíró fájlt, pontosabban, rögzíti az így elindított folyamat

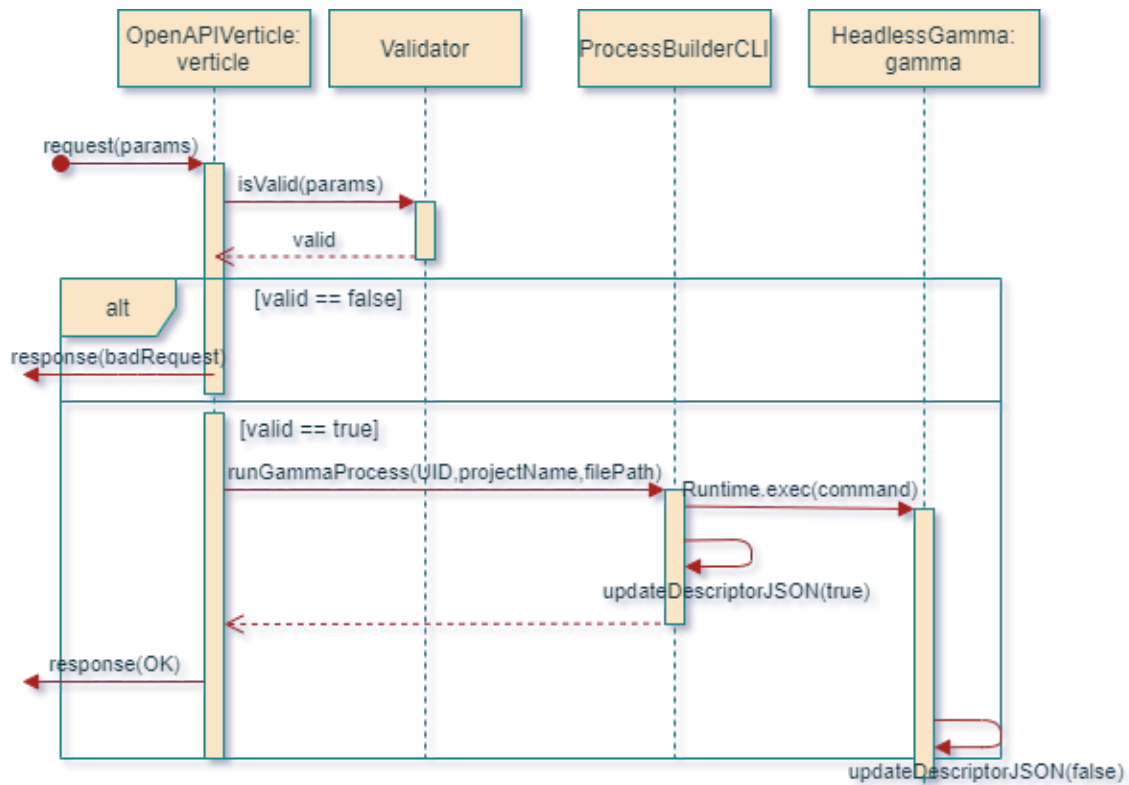


3.2.3. ábra. Eclipse projekt importálás szekvencia diagram

operációs rendszer szintű azonosítóját (PID) és beállítja, hogy a projekt futás alatt van. Végezetül átadja a futást a verticle-nek, ami jelzi a kliens felé, hogy a kérés feldolgozása elkezdődött. A folyamat még nem ér véget, mivel a Gamma dolgozik a háttérben, mikor minden művelet lefutott amit a .ggen fájlban meghatározott a felhasználó, akkor a headless eclipse frissíti a projektet leíró állományt és beállítja, hogy a projekt már nincs futás alatt. További lezáró művelet lehet az értesítés küldés a tárolt email címre, hogy a kért kérés lefutott és az eredmény mostantól lekérhető.

**Eredmények lekérése** A gamma által generált állományok lekérésére szolgáló funkció szekvencia diagramját a 3.2.5 ábra mutatja be. Egy PUT HTTP kérés érkezik a *getResult* endpoint-ra, ennek az URL-jében szerepelnie kell a munkatér azonosítójának és a projektnevének amiből adatot akarunk lekérni. A kérés törzsében a *resultDirs* attribútum kell szerepeljen, ami egy relatív elérésekből álló lista. A kérés feldolgozása a szokásos módon a verticle-ben kezdődik, ahol a kérésben utazó paraméterek kicsomagolásra kerülnek. A Validator ebben az esetben ugyanazokat ellenőrzi, mint a művelet futtatás folyamat esetén. Ha van futó folyamat a projekten vagy a workspace projekt páros nem megfelelő, akkor beszédes hibaüzenettel válaszolunk a kliensnek. Abban az esetben, ha minden rendben van, akkor a Provider kapja meg a futási jogot. Itt az elérési listán iterálva összegyűjtjük a kért fájlokat és mappákat majd archiváljuk, az így archivált fájl elérést küldjük vissza a verticle-nek, ami ennek alapján visszaküldi a fájlt.

Első látszatra zavaró lehet, hogy PUT metódust használunk a kérés feldolgozásához, mikor egy GET jobb lehetne és jobban megfelelne a REST szabványnak. Ennek két oka van, az első az, hogy annak ellenére, hogy *getResult*-nak hívjuk az endpoint-ot az erőforrás állapota módosul hiszen a projekten belül generálunk egy archivált fájlt, a másik oka pedig



3.2.4. ábra. Gamma művelet futtatás szekvencia diagram

az, hogy a GET kérésnél nem lehet a törzsben semmit sem küldeni, ezt a REST szigorúan megköveteli, nekünk viszont a becsomagolni kívánt állományokat listában kell megadnunk és ezt az URL-ben kényelmetlen lenne kezelni.

### 3.3. Implementáció

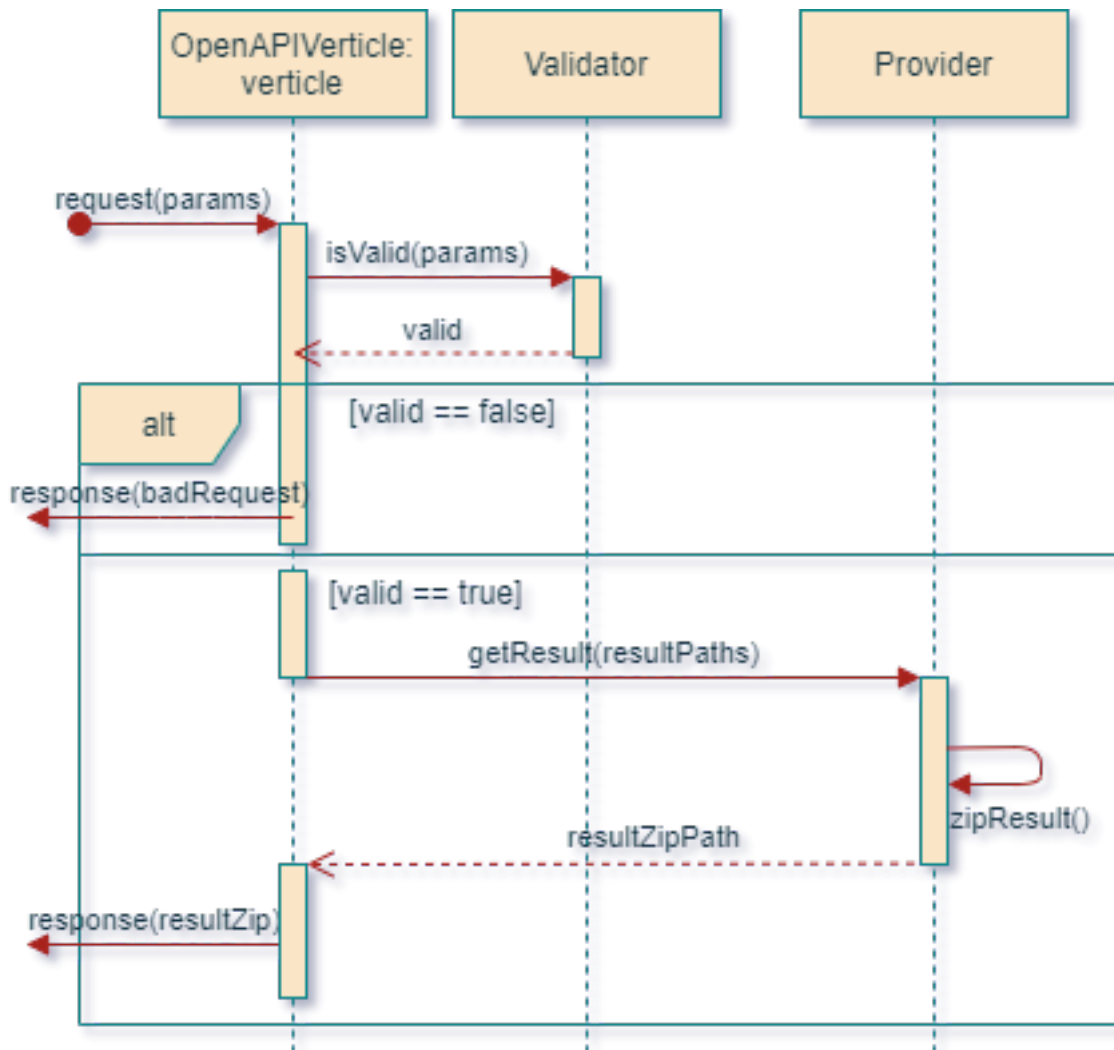
#### 3.3.1. Fejlesztés folyamata

Ebben a fejezetben a fejlesztés során felmerült nehézségeket és érdekességeket mutatom be. Az alábbi információk alapvetően mély technikai szintre lemennek így szárazak lehetnek, viszont a témában dolgozóknak hasznos lehet, mert egyedi hibák és jelenségek megoldását vázolom föl.

**Eclipse környezettel kapcsolatos érdekességek** Az Eclipse környezetben tapasztalt fejlesztők tudják, hogy bizonyos dolgok nem triviálisak a fejlesztés során és olyan specifikus problémák merülhetnek fel, amelyekre nagyon elrejtett internetes fórumokon lehet választ kapni. Az alábbi jelenségek és problémák a *Headless Gamma* komponens fejlesztése alatt merültek föl

Az eclipse plug-in fejlesztés és termék export konfiguráció során számos függőségi problémával szembesültem. A legelső érdekesség ami felmerült, az volt, hogy az Eclipse a *product* konfigurációban meghatározott plug-in függőségek verzióját automatikusan felülírta a környezetbe telepített legfrissebb verzióval. Ezen plug-in-ek listáját az Eclipse IDE-n belül a *Target Platform*-on lehet megtekinteni és szerkeszteni. Ilyen problémás függőségek voltak a *batik.css* és *batik.util* plug-in-ek. A Target Platform szerkesztésével ezt a





**3.2.5. ábra.** Eredmény lekérés szekvencia diagram

problémát orvosolni lehet, csupán ki kell vegyük vagy hozzá kell adjuk azt a verziót amire szükségünk van és győződjünk meg, hogy az adott plug-in-ből csak egy verziót aktív.

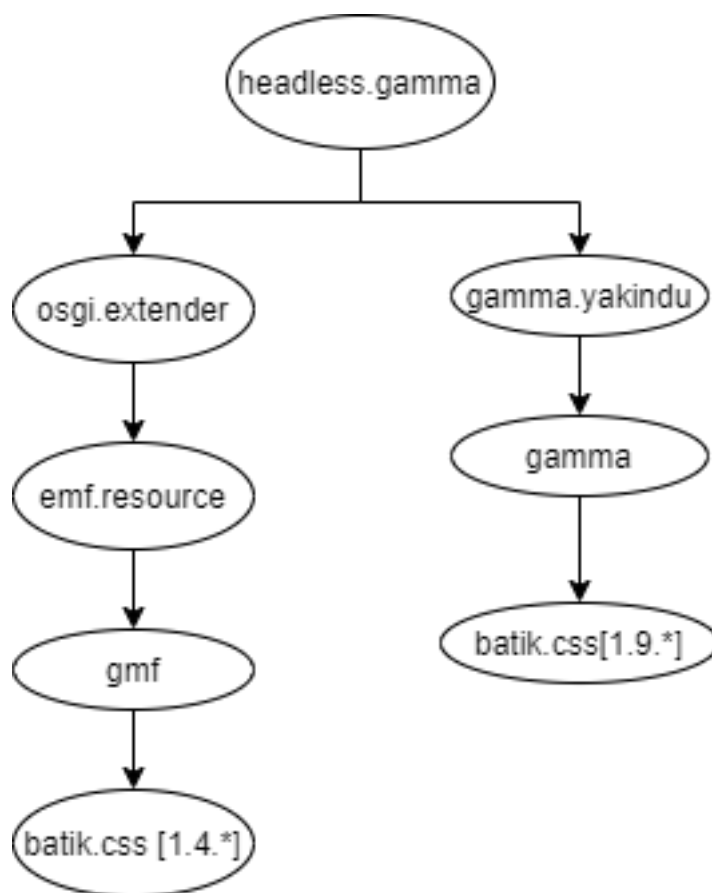
A product konfigurációban a *Content* fülön tudjuk meghatározni milyen plug-in-ekből álljon a termékünk, az IDE-ben lehetőség van arra, hogy megadjuk a fejlesztett alkalmazásunkat és rányomva az *Add Required* gombra az IDE automatikusan beállítja a termékünk összes függőségét. Ennek ellenére az osgi.extender olyan plug-in-eket igényel, amelyek az OSGi helyes működéséhez szükségesek de nem importálja be automatikusan. A szükséges modulok:

```

org.apache.felix.scr
org.eclipse.equinox.event
org.eclipse.compare.core
org.eclipse.fx.osgi
org.eclipse.team.core
  
```

Olyan jelenség is előjött, ahol a függőségi fában egy plug-in kétszer szerepelt viszont különböző verzióval, ezt az OSGi szabvány engedi de az Eclipse, az első jelenségben leírtak alapján, folyamatosan felülírta az elavultabb verziót. A probléma elképzelésében segít a 3.3.1 ábra. Erre a megoldás az, hogy a fában felfele haladva megvizsgáljuk, hogy milyen verziókat tudunk frissíteni a Target Platformon úgy, hogy a működést megtartsuk de az

elavult verziójú plug-in-t frissíteni tudjuk. Alapvetően egyszerű feladatnak tűnik, viszont a rendszer közel 250 függőséggel rendelkezik.



**3.3.1. ábra.** Egyszerűsített függőségi fa

A termék export felületen meg kell határozni, hogy az egyes Eclipse modulok milyen sorrendbe induljanak el, a mi esetünkbe az alábbi indulási szintek meghatározása volt fontos:

```

<configurations>
  <plugin id="org.apache.felix.scr" autoStart="true" startLevel="2" />
  <plugin id="org.eclipse.core.runtime" autoStart="true" startLevel="0" />
  <plugin id="org.eclipse.equinox.common" autoStart="true" startLevel="2" />
  <plugin id="org.eclipse.equinox.event" autoStart="true" startLevel="2" />
  <plugin id="org.eclipse.equinox.simpleconfigurator" autoStart="true" startLevel="1" />
</configurations>

```

Ez a konfiguráció részlet a termék leírás XML fájlból van másolva, az IDE-ben a *Configuration* fülön lehet mindezt beállítani.

Egy termék konfiguráció futtatására két lehetőségünk van, az első, hogy exportáljuk (headless eclipse) majd parancssorból meghívjuk és átadjuk az argumentumokat vagy az IDE-ből elindítjuk. Az utóbbi tesztelés folyamán nagyon hasznos tud lenni. Ha IDE-ből futtatjuk akkor is van lehetőségünk argumentumokat átadni, erre szintén a termék konfigurációban van lehetőség, pontosabban a *Launching* fülön. Arra viszont érdemes odafigyelni, hogy ezt a tesztelés után ne hagyjuk ott, mivel az export konstans argumentumoknak értelmezi és minden headless módban indulásnál átadja önmagának. A mi esetünkbe ez hatványozottan fontos, mivel az átadott argumentumok minden esetben mások a munkatér és projektek számossága miatt.

Minden függőség lehet opcionális vagy kötelező az opcionálisakat csak abban az esetben tölti be a headless eclipse mikor éppen szüksége van rá. A rendszerünknek a *jdt* plug-in család több komponensére is szüksége van viszont ezek a függőségi fában levő szülőjükön a *xtext.ui* plug-in-en csupán opcionálisként vannak megjelölve és egy mai napig nem tisztázott ok miatt nem töltődtek be. A problémát azzal lehet orvosolni, hogy készítünk az *xtext.ui* plug-in-ből egy saját példányt, amin a függőségi beállításokat átírjuk opcionálisról kötelezőre. Ez sajnos nehezíti a rendszer tovább fejlesztését és a telepítési folyamat is komplexebbé válik.

Egy kulcsfontosságú probléma az volt, hogy az Xtext<sup>4</sup> modult eddig nem tudtuk injektálni a headless eclipse-be. Erre az alábbi kódrészlet ad megoldást:

```
Injector injector = new StatechartLanguageStandaloneSetupGenerated()
    .createInjectorAndDoEMFRegistration();
XtextResourceSet resourceSet = injector.getInstance(XtextResourceSet.class);
```

A fent definiált *resourceSet*-ről tudjuk majd a headless eclipse-nek argumentumban átadott fájlt elérni.

Minden Gamma nyelv interpretálóját egyedileg meg kellett hívni az alkalmazás indulási pillanataiban:

```
//Alkalmazás belépési pontja
@Override
public Object start(final IApplicationContext context) throws Exception {
    ExpressionLanguageStandaloneSetup.doSetup();
    ActionLanguageStandaloneSetup.doSetup();
    StatechartLanguageStandaloneSetup.doSetup();
    TraceLanguageStandaloneSetup.doSetup();
    PropertyLanguageStandaloneSetup.doSetup();
    GenModelStandaloneSetup.doSetup();
    .
    .
    .
}
```

**OpenAPI REST webszerver** Olyan fejlesztői megoldásokat mutatok be, amelyek az OpenAPI és Vert.x használata által nagyon egyszerűen megvalósíthatóak voltak. Ehhez tekintsük át a 3.3.2 ábrát amely az *addproject* endpoint specifikációját tartalmazza.

A Vert.x, mint említettem támogatja az OpenAPI REST specifikáció integrációját, így kódból az alábbi módon tudjuk elérni a fájlban definiált *endpointot* (gamma-wrapper.yaml), amit majd regisztrálunk a webszerverünkbe:

```
@Override
public void start(Future<Void> future) {
    OpenAPI3RouterFactory.create(this.vertx, "gamma-wrapper.yaml", ar ->
    {
        OpenAPI3RouterFactory routerFactory = ar.result();
        routerFactory.addHandlerByOperationId("addProject", routingContext -> { ... });
    });
}
```

A fenti kódrészletben már nagyon egyszerűen tudjuk a kérésben utazó paramétereket Java objektummá átalakítani. A példában fájlok feldolgozását is el kell végezzük, ezt a Vert.x a fenti *routingContext* objektumon már eltárolja és egy egyszerű *routingContext.fileUploads()* hívással már *File* objektumok listáját kapjuk vissza.

Bizonyos validációs feladatokat is elvégez helyettünk az OpenAPI definíció alapján a Vert.x, a 3.3.2 ábrán látható, hogy meghatározzuk a munkatér(workspace) szintaktikáját, ha ettől eltérő, más formátumú karakterlánc szerepel a kérésben akkor automatikusan küldődik egy válasz a klienshez, hogy nem megfelelő a kérése.

<sup>4</sup>Xtext-ről többet: <https://www.eclipse.org/Xtext/>

```

/gamma/addproject/{workspace}:
post:
  operationId: addProject
  description: Send a file that contains the eclipse project on which the gamma operations will
  run
  parameters:
    - in: path
      name: workspace
      required: true
      description: The workspace which contains the project
      schema:
        type: string
        format: uuid
        example: 3fa85f64-5717-4562-b3fc-2c963f66afa6
  requestBody:
    content:
      multipart/form-data:
        schema:
          type: object
          properties:
            contactEmail:
              type: string
            file:
              type: string
              format: binary
  responses:
    200:
      description: Successfully uploaded the file
    401:
      description: Did not provide a valid workspace
    403:
      description: Project already exists under this workspace, delete it and resend this request

```

### 3.3.2. ábra. Endpoint példa

**ProcessBuilder fejlesztés** Alapvetően a webszerver java 8 alapokon lett megvalósítva, majd kiderült, hogy a java 9 hasznos, új frissítéseket tartalmaz a *java.lang.ProcessBuilder* komponensén. Számunkra ez azért fontos, mert bevezetésre került a *pid* (process id) lekérése az éppen indított folyamatról. Ezt az azonosítót feltudjuk használni, hogy egy Gamma műveleteket feldolgozó folyamatot leállítsunk, ha úgy gondoljuk, hogy túl sok ideig fut. Nagyon fontos megjegyezni azt, hogy ha folyamatot indítunk a *java.lang.ProcessBuilder* segítségével és ez több ideig fut, akkor a logolási eredményét kötelezően át kell irányítsuk fájlba vagy konzolba. Ennek hiányában a folyamatunk beakad. A rendszerünk a folyamatot indító konténerre leörökölteti a logolás feladatát. A HeadlessGamma elindítására szolgáló kódrészlet:

```

ProcessBuilder pb = new ProcessBuilder(commandToBeExecuted); // ProcessBuilder inicializálás,
    commandToBeExecuted a futtatni kívánt parancs karakterláncát tartalmazza
pb.redirectErrorStream(true);
pb.inheritIO(); // Logolás öröklés
long pib = pb.start().pid(); // Folyamat azonosító lekérése
}

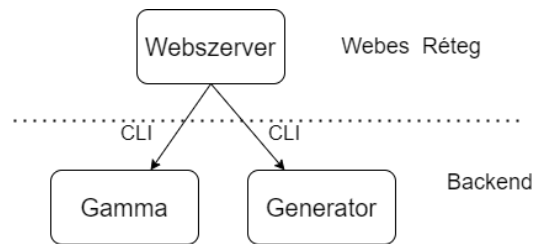
```

### 3.3.2. Rendszer szoftverkomponensei

A rendszer struktúrájából kiindulva, ezt a 3.2.1 ábrán lehet megtekinteni, a rendszer szoftver szintű felépítése három komponensből tevődik össze, ezek vizuális reprezentációját a 3.3.3 ábra mutatja be.

A Webszerver tartalmazza az összes olyan komponenst, amely a webes kérések értelmezését és átalakítását végzik, itt található az OpenAPI definíció. Önmagában egy Maven<sup>5</sup> alapú IntelliJ IDEA-ban fejlesztett alkalmazás. A Gamma és a Generator pedig

<sup>5</sup>Maven-ről többet: <https://maven.apache.org/>



**3.3.3. ábra.** Szoftver komponensek

az eddigiekben már többször említett két headless eclipse alkalmazás. A webszerver egy master szerepet tölt be a rendszerünkbe, mivel ő koordinálja a másik kettő működését.

## 4. fejezet

# Használati útmutató

Ez a fejezet tartalmazza az alkalmazás jelenlegi formájának a telepítési instrukcióit, továbbá bemutatja a fentebb részletezett funkciókat működés közben. Jelenleg az alkalmazás még nem konténerizált formátumú így szükségünk lesz segédeszközökre.

### 4.1. Telepítés

Használat előtt győződjünk meg, hogy rendelkezünk az alábbiakkal:

- java 11
- UPAAL elérés regisztrálva van a környezeti változókba
- API tesztelő alkalmazás pl: Postman<sup>1</sup>
- Egy Java fejlesztői eszköz pl: IntelliJ IDEA<sup>2</sup>

A szoftver komponensek közül a Webszervert kell megnyitni az IDEA segítségével. Az OpenProject funkciót használva tallózzuk ki a webszerver gyöker mappájában levő *pom.xml*-t. Miután látható a projekt a fejlesztői környezetbe, keressük ki a *configuration.properties* fájlt, ami tipikusan az */src/main/resource mappában* lesz elhelyezve, és állítsuk be az alábbiakat:

```
headless.gamma.path=<A Headless Gamma teljes elérése>  
headless.generator.path=<A Headless Generator teljes elérése>  
root.of.workspaces.path=<Annak a mappának a teljes elérése, ami a munaktereket fogja tartalmazni>
```

Fontos, hogy a két headless eclipse elérése tartalmazza az eclipse kulcsszót az elérés végén, mivel ez lesz a .exe állomány (pl: C:/opt/gammaapi/eclipse). A két headless eclipse-t exportálással tudjuk előállítani, viszont a Webszerver komponens tartalmazni fogja ezeket, ennek ellenére a konfigurációt el kell végezni. Innentől kezdve az IDEA-ból elindítható az alkalmazás és a *localhost:8080*-on fog figyelni a kérésekre. Az alábbiakban részletezni fogom, hogyan tudjuk tesztelni az alkalmazást.

### 4.2. Tesztelés

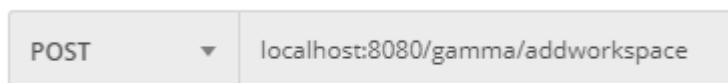
Tesztelés során a gamma teszt projektet<sup>3</sup> fogom használni, mivel tartalmaz komplex gamma művelethalmazokat. Minden kérést a Postman segítségével fogok összeállítani és elküldeni, majd megtekintjük a szerveren milyen hatása volt.

<sup>1</sup>Postman letölthető: <https://www.postman.com/>

<sup>2</sup>IDEA letölthető: <https://www.jetbrains.com/idea/download>

<sup>3</sup><https://github.com/ftsrc/gamma/tree/master/tests>

**Workspace létrehozás** Mivel jelenleg a szerverünk nem tárol semmit ezért hozzunk létre három munkateret. A kiadott HTTP kérések ugyanazok lesznek hiszen nem utazik semmi a paraméterekben ezt a 4.2.1 ábrán lehet megtekinteni.







**4.2.1. ábra.** Munkatér regisztráció HTTP kérés

A válasz üzenet tartalmazni fogja egyesével a munkatér azonosítókat:

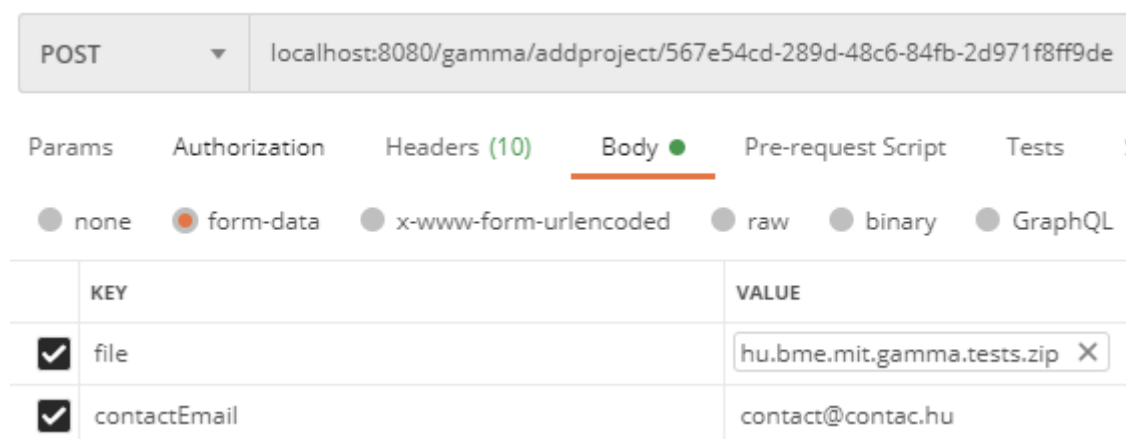
```
567e54cd-289d-48c6-84fb-2d971f8ff9de
99b21cd3-6778-47db-9e00-29e3ce2420be
394d3fc8-5abb-4f40-a13d-252185d8c9bf
```

Ezt a mappa kezelő segítségével is ellenőrizni tudjuk, ahogy a 4.2.2 ábrán látható is. Az alkalmazáshoz tartozik egy konfigurációs fájl, amiben megadható, hogy a workspace-k hova jöjjenek létre.

 wrapperList.json	12/8/2020 1:18 AM	JetBrains WebStorm	1 KB
 394d3fc8-5abb-4f40-a13d-252185d8c9bf	12/8/2020 1:18 AM	File folder	
 99b21cd3-6778-47db-9e00-29e3ce2420be	12/8/2020 1:18 AM	File folder	
 567e54cd-289d-48c6-84fb-2d971f8ff9de	12/8/2020 1:17 AM	File folder	

**4.2.2. ábra.** Munkaterek tárolása

**Projekt importálás** A gamma teszt projektet fogjuk importálni két különböző munkatérbe. Nagyon fontos, hogy a projekt tartalma archiválva legyen és az archivált fájl neve megegyezzen a projekt nevével. A kérésben ebben az esetben meg kell adjuk a munkatér azonosítót az URL-ben és a kérés törzsébe fel kell vennünk az említett archivált állományt és a projekt tulajdonos e-mail címét. A példánkra illeszkedő kérést a 4.2.3 ábra mutatja be.



**4.2.3. ábra.** Projekt importálás kérés

Ezen kérések hatására létrejönnek a projektek a megfelelő munkaterek alá, ezt a 4.2.4 ábrán is meglehet tekinteni.

C > E (E:) > Egyetem > GammaWrapper > Workspaces > 567e54cd-289d-48c6-84fb-2d971f8ff9de > hu.bme.mit.gamma.tests				
Name	Date modified	Type	Size	
.settings	12/8/2020 1:52 AM	File folder		
model	12/8/2020 1:52 AM	File folder		
src-gen	12/8/2020 1:52 AM	File folder		
test-gen	12/8/2020 1:52 AM	File folder		
trace	12/8/2020 1:52 AM	File folder		
.classpath	9/23/2020 11:50 AM	CLASSPATH File	1 KB	
.gitignore	9/23/2020 11:50 AM	Text Document	1 KB	
.project	9/23/2020 11:50 AM	PROJECT File	1 KB	
projectDescriptor.json	12/8/2020 1:52 AM	JetBrains WebStorm	1 KB	

#### 4.2.4. ábra. Projekt importálás eredmény

**Gamma műveletek futtatása** Tekintsük át a teszt projektben szereplő /model/SSM/System/Mission.ggen fájlt.

```
import "Mission.gcd"
import "GroundStation/GroundStation.gcd"
import "Spacecraft/Spacecraft.gcd"

code {
  component : GroundStation
  language : java
}

code {
  component : Spacecraft
  language : java
}

code {
  component : Mission
  language : java
}

analysis {
  component : Mission
  language : UPPAAL
  transition-coverage
  constraint : {
    minimum-orchestrating-period : 999 ms
    maximum-orchestrating-period : 999 ms
  }
}

verification {
  language : UPPAAL
  file : "Mission.xml"
  query-file : "Mission.q"
  optimize : true
  test-language : java
}
```

Látható, hogy három kódgenerálást és egy verifikációt is fogunk végezni a felső három sorban importált modelleken. Futtassuk ezt az állományt a két projektünkön, a 4.2.5 kérdés alapján, annyi különbséggel, hogy a megfelelő munkatér azonosítót adjuk meg. Az URL-ben utazik a munkatér azonosító, projektnév és a .ggen fájl elérése a projekten belül.

Ennek hatására a hoszt szerveren elindul a két folyamat, erről a `tasklist | findstr eclipse` parancs segítségével győződhetünk meg:

eclipse.exe	1008 Console	2	1,012,184 K
-------------	--------------	---	-------------



PUT	localhost:8080/gamma/api/567e54cd-289d-48c6-84fb-2d971f8ff9de/hu.bme.mit.gamma.tests/model_SSM_System_Mission.ggen
-----	--

#### 4.2.5. ábra. Gamma műveletek futtatására kérés

eclipse.exe	1272 Console	2	863,800 K
-------------	--------------	---	-----------

A projektek jelenleg feldolgozás alatt állnak, így semmilyen műveletet nem lehet rajtuk végezni. Ha mégis küldünk egy kérést erre a projektre, akkor az alábbi hibaüzenet fogjuk kapni a szervertől:

```
{
  "code": 503,
  "message": "There is an in progress operation on this project, try again later!"
}
```

**Gamma műveletek leállítása** Előfordulhat, hogy valamiért egy művelet túl sok ideig fut, ezzel zárolva a teljes projektet, ezt fel lehet oldani a 4.2.6 ábrán levő kérés segítségével. A kérésben URL-jében a projektet meghatározó munkatér-projektnév páros kell utazzon, a törzs üres maradhat. Jelenlegi esetben nem merül fel ilyen hiba, mivel mindkét folyamat kb 5-7 perc alatt lefut, viszont a tesztelés céljából leállítjuk az egyiket.

PUT	localhost:8080/gamma/stopprocess/99b21cd3-6778-47db-9e00-29e3ce2420be/hu.bme.mit.gamma.tests
-----	--

#### 4.2.6. ábra. Futó headless gamma leállítása

Eredményként annyit tapasztalunk, hogy megint használhatóvá válik a projekt.

**Gamma eredmények lekérése** Végezetül szükségünk van a generált eredmény állományokra. Ezt a 4.2.7 ábrán levő kéréssel tudjuk megvalósítani, az URL-ben a megszokott munkatér-projektnév páros utazik, a törzsben pedig egy olyan json objektum van, amiben listaként szerepelnek az egyes állományok relatív elérése. A Postman-be ilyenkor nem csak el kell küldeni a kérést hanem a *Send and Download* gombra kell kattintani, így tudni fogja, hogy fájlt fog kapni a szervertől.

PUTlocalhost:8080/gamma/getresult/567e54cd-289d-48c6-84fb-2d971f8ff9de/hu.bme.mit.gamma.tests

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"resultDirs":["src-gen","test-gen","trace/ExecutionTrace0.get"]

3

}

#### 4.2.7. ábra. Eredmény lekérés

Eredményül pedig a megadott állományokat kapjuk becsomagolva, a 4.2.8 ábrán a *src-gen* tartalma látható.

**További funkciók** Lehetőségünk van projekt törlésre is, erre a 4.2.9 ábrán látható kérés szolgál. Az URL-ben utazó munkatér és projektnév párossal azonosított projekt teljes tartalma törlésre kerül.

↑ response.zip\result\src-gen\hu\bme\mit\gamma\tests - ZIP archívum, kicsomagolt méret 4,320,526 byte

Név	Méret	Tömörítve	Típus	Módosítva	CRC32
..			File folder		
groundstation			File folder		
interfaces			File folder		
spacecraft			File folder		
ssm_mission			File folder		
Event.java	380	169	IntelliJ IDEA Community...	12/8/2020 3:26 AM	4B68EA6C
ITimer.java	214	137	IntelliJ IDEA Community...	12/8/2020 3:26 AM	FEDD74F5
ITimerCallback.java	112	104	IntelliJ IDEA Community...	12/8/2020 3:26 AM	DFABE650
OneThreadedTimer.java	849	377	IntelliJ IDEA Community...	12/8/2020 3:26 AM	73A3D921
ReflectiveComponentInterface.java	792	313	IntelliJ IDEA Community...	12/8/2020 3:26 AM	05D4BBC5
TimerInterface.java	262	163	IntelliJ IDEA Community...	12/8/2020 3:26 AM	FDA7B3F1
TimerService.java	2,654	932	IntelliJ IDEA Community...	12/8/2020 3:26 AM	3AFBD2F6
UnifiedTimer.java	706	297	IntelliJ IDEA Community...	12/8/2020 3:26 AM	A5D8CCFA
UnifiedTimerInterface.java	114	92	IntelliJ IDEA Community...	12/8/2020 3:26 AM	7F928938
VirtualTimerService.java	2,964	1,028	IntelliJ IDEA Community...	12/8/2020 3:26 AM	AEB622B9

4.2.8. ábra. Eredmény fájl tartalma

DELETE	localhost:8080/gamma/deleteproject/567e54cd-289d-48c6-84fb-2d971f8ff9de/hu.bme.mit.gamma.tests
--------	--

4.2.9. ábra. Projekt törlés kérés

A lejáró projektet dátumát egy naponta futó időzítő figyeli, ha talál olyan projektet, aminek a lejáráti dátuma a mai nap, akkor azt törli. A 4.2.10 ábrán levő kéréssel tudjuk 30 nappal meghosszabbítani a lejáráti dátumot.

PUT	localhost:8080/gamma/extendexpiration/567e54cd-289d-48c6-84fb-2d971f8ff9de/hu.bme.mit.gamma.tests
-----	---

4.2.10. ábra. Projekt lejáráti dátum meghosszabbításra szolgáló kérés

## 5. fejezet

# Összefoglalás

A modellalapú szoftverfejlesztés által kínált eszközökkel világszerte a fejlesztők képesek hatékonyabban minőségibb alkalmazásokat fejleszteni. A Gamma Keretrendszer is ebben nyújt segítséget, komplex, komponens alapú rendszerek modellezését teszi lehetővé. Ennek ellenére fejlesztői környezethez való kötöttsége nehezíti a felhasználását és skálázhatóságát.

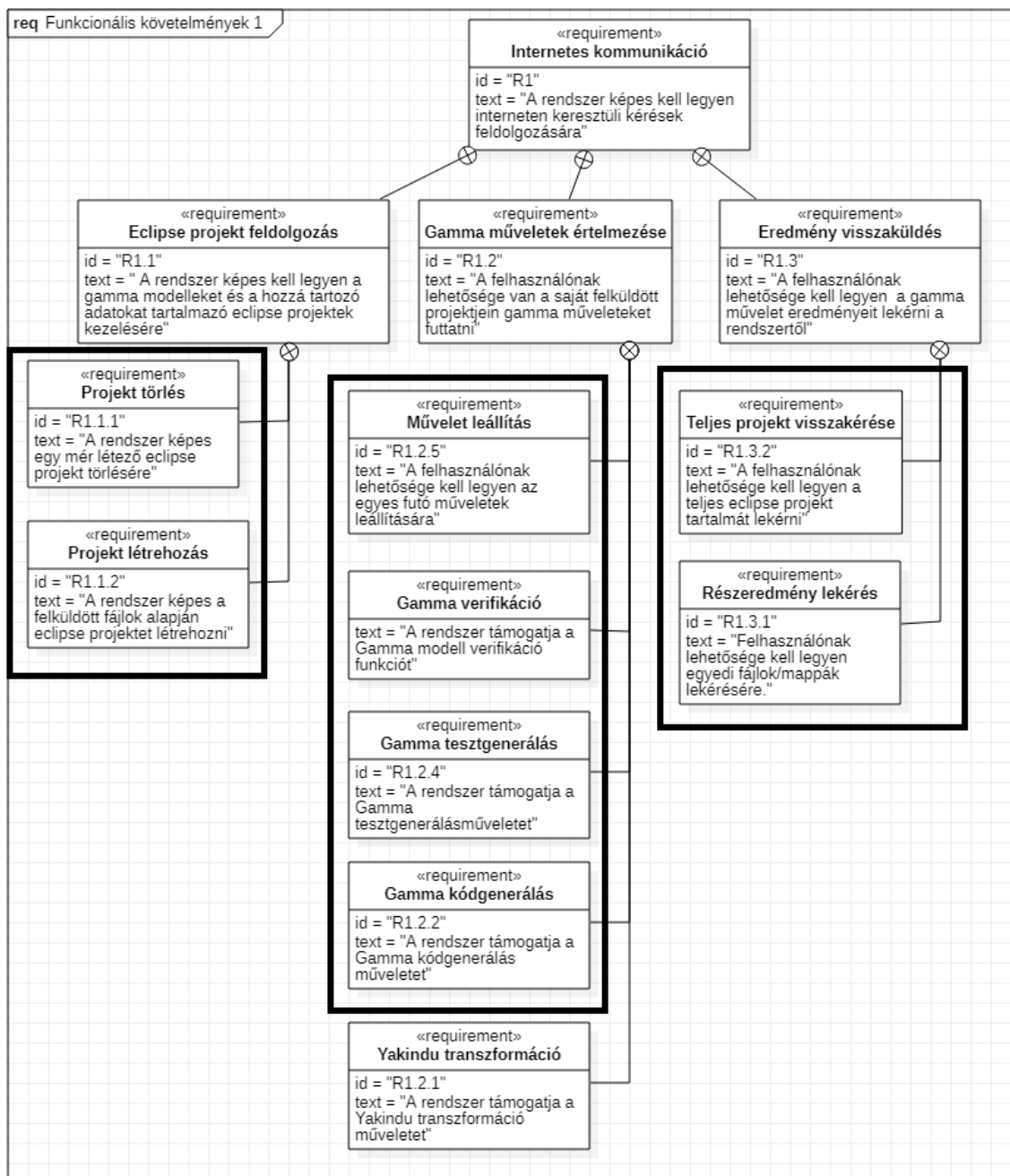
Ezen dolgozat keretein belül, egy olyan rendszert alakítottam ki, amely a felvetett problémát megoldja. A Gamma Keretrendszert átalakítottam egy olyan alkalmazás csomaggá, amely az internet segítségével bárholnan elérhető, továbbá kötetlen és platform független, így lehetővé teszi, hogy felhőalapú megoldásként is integrálhatjuk így az ezzel járó skálázhatóságot és flexibilitást is kitudjuk használni. Bemutattam, hogy milyen technológiák szükségesek egy ilyen rendszer kialakításához, törekedve arra, hogy modern, korszerű megoldásokat használjak. Specifikált funkcionális és nem funkcionális követelmények alapján megterveztem a rendszer különböző komponenseit és az ezek közti kommunikációs csatornákat, majd leimplementáltam a rendszert. Az implementáció során felmerült érdekességeket és nehézségeket részleteztem és egy valós tesztalmaz segítségével teszteltem a rendszer működését.

A végső termékünk a kitűzött célnak megfelel, viszont tartalmaz olyan részeket, amelyek enyhén hiányosak vagy technikai szempontból nem optimálisak. A gamma funkciók közül mindegyik támogatott, csak a Yakindu-Gamma transzformáció nem. Az egyes fájl alapú (JSON) adattárolási megoldásokat érdemes lehet, egy adatbázissal helyettesíteni. A lejárat dátum alapú automatizált törlés implementációja jelenleg hiányos így nem működik. Egyes részekben a rendszer szinkron módon működik, érdemes lehet átgondolni, hogy az asszinkron hatékonyság a munkatér létrehozás és projekt importálás esetén releváns-e. A projekt törlés csak abban az esetben működik megfelelően ha a munkatérben nincs más projekt.

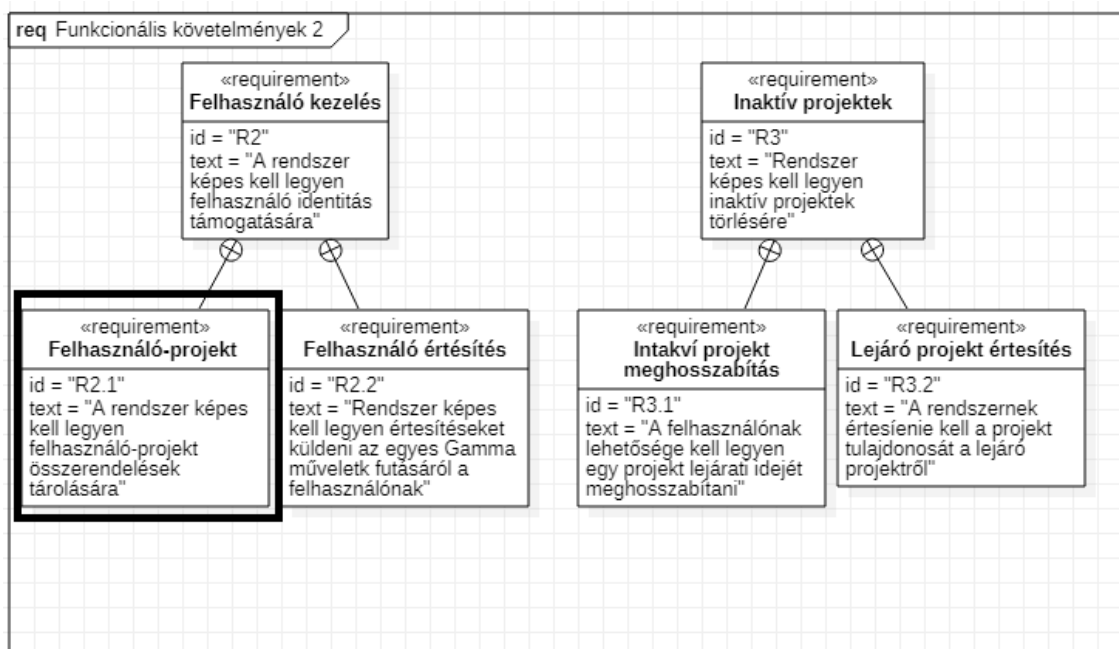
A 3.1 fejezetben leírt követelmények többségét a rendszerünk teljesíti, a pontos listát az 5.0.1 és az 5.0.2 ábrákon lehet megtekinteni, a vastagon bekeretezett követelményeket a rendszer jelenlegi állapota teljesíti, a többi pedig részlegesen vagy egyáltalán nem teljesül.

A rendszer további fejlesztési fázisaiban meg kell vizsgálni, hogy a jelenlegi funkcionális kielégít-e minden felhasználói igényt, ha esetleg szükséges akkor új endpoint-ok bevezetésével egyszerűen bővíteni lehet. Az alkalmazást valamilyen konténerbe kellene helyezni, hogy a felhő integráció jelentősen egyszerűbb legyen. A felvázolt hiányosságokat a jövőben érdemes lenne áttekinteni és javítani.

A bemutatott rendszer két fontos ágazatát ötvözi az informatikának, modellalapú szoftverfejlesztés és felhőalapú alkalmazások, ezzel potenciálisan segítve a területen dolgozó mérnökök munkáját.



5.0.1. ábra. Teljesült funkcionális követelmények 1



5.0.2. ábra. Teljesült funkcionális követelmények 2

# Irodalomjegyzék

- [1] Shif Ben Avraham: What is rest - a simple explanation for beginners, part 2: Rest constraints, 2017. Sep. URL <https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a5>
- [2] Benefits of cloud computing.  
URL <https://www.ibm.com/cloud/learn/benefits-of-cloud-computing>.
- [3] Lars Vogel (c) 2009 2020 vogella GmbH: Get more...  
URL <https://www.vogella.com/tutorials/EclipsePlugin/article.html>.
- [4] Chelsea: Api design, part 1: Before there was rest, 2019. May. URL <https://chelseatroy.com/2018/08/01/api-design-part-1-before-there-was-rest/>.
- [5] Equinox Committers: Equinox: The eclipse foundation.  
URL <https://www.eclipse.org/equinox/>.
- [6] Cs 410/510 - software engineering system modeling. URL <https://cs.ccsu.edu/~stan/classes/CS410/Notes16/05-SystemModeling.html>.
- [7] Eclipse vert.x. URL <https://vertx.io/>.
- [8] Bence Graics – Vince Molnár – András Vörös – István Majzik – Dániel Varró: Mixed-semantics composition of statecharts for the component-based design of reactive systems. *Softw. Syst. Model.*, 19. évf. (2020) 6. sz., 1483–1517. p.  
URL <https://doi.org/10.1007/s10270-020-00806-5>.
- [9] Vince Molnár – Bence Graics – András Vörös – István Majzik – Dániel Varró: The gamma statechart composition framework: : design, verification and code generation for component-based reactive systems. In Michel Chaudron – Ivica Crnkovic – Marsha Chechik – Mark Harman (szerk.): *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018* (konferenciaanyag). 2018, ACM, 113–116. p.  
URL <https://doi.org/10.1145/3183440.3183489>.
- [10] Openapi specification. URL <https://swagger.io/specification/>.
- [11] Rest. URL <https://restfulapi.net/>.
- [12] Lars Vogel(c) 2007 2020 vogella GmbH: Get more... URL <https://www.vogella.com/tutorials/EclipseProductDeployment/article.html>.
- [13] What are the advantages of cloud computing? 10 reasons to move to the cloud. URL <https://www.salesforce.com/uk/blog/2015/11/why-move-to-the-cloud-10-benefits-of-cloud-computing.html>.
- [14] What is osgi? URL <https://www.osgi.org/developer/what-is-osgi/>.

- [15] What is rest? URL <https://www.codecademy.com/articles/what-is-rest>.
- [16] What is the difference between swagger and openapi? URL <https://swagger.io/blog/api-strategy/difference-between-swagger-and-openapi/>.
- [17] Why web services?  
URL [https://www.tutorialspoint.com/webservices/why\\_web\\_services.htm](https://www.tutorialspoint.com/webservices/why_web_services.htm).
- [18] Ahmet ÖZLÜ: Mastering rest architecture - rest architecture details, 2019. Jan. URL <https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc>.