**Projekt dokumentáció**

**Céges számlafeldolgozó folyamatot támogató alkalmazás**

Készítette: Adonics Nikolett

Beosztás: Szoftverfejlesztő gyakornok

Csoportvezető: Bíró Ádám

Helyszín: Szeged

Időtartam: 2022.07.14 – 2022.10.25

Munkaórák: 320 óra

**Tartalomjegyzék**

[**Bevezetés** 4](#_Toc117753814)

[**A dokumentum célja** 4](#_Toc117753815)

[**A dokumentum tartalma** 4](#_Toc117753816)

[**Fogalomtár** 4](#_Toc117753817)

[**Mellékletek, vonatkozó dokumentumok** 4](#_Toc117753818)

[**Témakiírás** 5](#_Toc117753819)

[**Követelmény analízis** 6](#_Toc117753820)

[**Funkcionális követelmények:** 6](#_Toc117753821)

[**Nem funkcionális követelmények:** 6](#_Toc117753822)

[**Áttekintés, solution design** 7](#_Toc117753823)

[**Use-case diagram** 7](#_Toc117753824)

[**State Machine diagram** 8](#_Toc117753825)

[**Architecture diagram** 9](#_Toc117753826)

[**Entity Relation diagram** 10](#_Toc117753827)

[**Interfészek szerepe** 10](#_Toc117753828)

[**Kommunikáció** 17](#_Toc117753829)

[**Komponensek** 18](#_Toc117753830)

[Component diagram 18](#_Toc117753831)

[User / Browser 18](#_Toc117753832)

[Windows Server 19](#_Toc117753833)

[Resource Server 19](#_Toc117753834)

[**Technikai leírás** 20](#_Toc117753835)

[**Database Relationship diagram** 20](#_Toc117753836)

[**Class diagram** 21](#_Toc117753837)

[**JWT** 21](#_Toc117753838)

[**Bejelentkezés** 21](#_Toc117753839)

[**Számla jóváhagyási folyamata** 22](#_Toc117753840)

[**Jelszó titkosítása** 23](#_Toc117753841)

[**Problémák** 23](#_Toc117753842)

[**Bugok** 23](#_Toc117753843)

[**Errorok** 23](#_Toc117753844)

[**Security rizikók** 23](#_Toc117753845)

[**Továbbfejlesztési lehetőségek** 23](#_Toc117753846)

[**Használhatóság (Angular Server):** 23](#_Toc117753847)

[**Védelem (Spring boot Server):** 24](#_Toc117753848)

[**Funkcionalitás (Spring boot Server):** 24](#_Toc117753849)

[**PostgreSQL Server** 24](#_Toc117753850)

[**Használt szoftverek, architektúrák** 25](#_Toc117753851)

[**Munkakörnyezet specifikációja** 25](#_Toc117753852)

[**Szoftverek** 25](#_Toc117753853)

[**Használt technológiák** 25](#_Toc117753854)

[**Referenciák** 26](#_Toc117753855)

[**Angular** 26](#_Toc117753856)

[**Fullstack** 26](#_Toc117753857)

[**JPA és PostgreSQL database server** 26](#_Toc117753858)

[**Spring Boot és Spring Security** 27](#_Toc117753859)

[**Blob kezelés** 27](#_Toc117753860)

# **Bevezetés**

## **A dokumentum célja**

A dokumentumban bemutatásra kerül a projekt leírásától az elkészült weboldalig a készítés folyamata. Részletesen tárgyalja az architektúra felépítését és a fontosabb részek technikai részleteit is ismerteti, a komponensek szerepét, valamint hogy hogyan lehetne továbbfejleszteni a projektet. Számos diagram szemlélteti a weboldal működését, amik segítenek annak megértésében.

## **A dokumentum tartalma**

A projekt leírása, a konfigurációk, az architektúra, a komponensek, a technikai leírások, a problémák és a továbbfejlesztési lehetőségek.

## **Fogalomtár**

* JWT – JSON Web Token
* UI – User Interface
* BLOB – Binary Large OBject
* SQL – Structured Query Language
* HTTP – HyperText Transfer Protocol
* REST – Representational State Transfer
* API – Application Programming Interface
* CORS – Cross-Origin Resource Sharing
* CSRF – Cross Site Request Forgery
* AOP – AspektusOrientált Programozás

## **Mellékletek, vonatkozó dokumentumok**

* diagramok
  + Use-case diagram
  + State Machine diagram
  + Application Architecture diagram
  + Entity Relation diagram
  + Component diagram
  + Class diagram
  + Database diagram
* dokumentumok
  + [api-docs](docs/api-docs.yaml) – OpenAPI dokumentáció
  + [component-docs](docs/component-docs.yaml) – OpenAPI dokumentáció
* (Megjegyzés: A diagramok megtalálhatók a diagram mappában)

# **Témakiírás**

A feladat egy Spring Boot és Angular alapú céges számlafeldolgozó folyamatot támogató webes alkalmazás elkészítése.

Ez alapján az alkalmazásnak három fő elemből kell állnia:

1. Spring Boot alapú back-end: a fájlkezelésért, a folyamatért és jogosultságkezelésért, a frontend kiszolgálásáért felel
2. SQL adatbázis: a felhasználók és a számlák tárolására, PDF fájlokat BLOB-ként vagy tetszőleges fájlrendszeri mappában tárolva
3. Angular keretrendszerű webes front-end

A front-end és a back-end közötti kommunikáció HTTP REST interfészen keresztül kell, hogy történjen.

Az alkalmazásban szerepelnie kell egy bejelentkezési lehetőségnek. Ennek jogosultság szerint kell megtörténnie.

A jogosultságok:

* pénzügyi ügyintéző (financial clerk)
* pénzügyi controller (financial controller)
* manager (manager)
* adminisztrátor (administrator)

A számlafeldolgozás lépései jogosultság szerinti felosztásban:

A pénzügyi ügyintéző felelősége a számla feltöltése PDF formátumban, a számlához tartozó metaadtatok kitöltése és az engedélyezési folyamat indítása.

A pénzügyi controller felelősége a számla elsőszintű, pénzügyi elfogadása, visszautasítása, korrekcióra visszaküldése.

A manager felelőssége a számla másodikszintű, végleges elfogadása, visszautasítása, korrekcióra visszaküldése.

Az adminisztrátor ezek közül bármit megtehet és ezeken felül törölheti is a számlákat.

Bármelyik jogosultsággal rendelkező bejelentkezett felhasználó jogosult a számlák listázására, a listában keresésre, szűrésre és a számla letöltésére.

A felhasználók biztonsága érdekében szükség van biztonságos jelszókezelésre is. A jelszavak hash értékeit és a felhasználókat és jogosultságaikat egy titkosított adatbázisban kell eltárolni.

Opcionális feladatok:

* feltöltés során történő alapvető validálás -> csak PDF-et lehessen feltölteni
* ha tetszőleges fájlrendszeri mappában tároljuk a PDF-et, akkor a mappához való egyéb, applikáción kívüli hozzáférés tiltása
* password policy

# **Követelmény analízis**

## **Funkcionális követelmények:**

Bejelentkezési lehetőség a felhasználó jogosultságának megfelelően és minden funkciót ezen jogosultságok alapján jelenítsen meg az alkalmazás. Valamint ha a felhasználónak nincs már szüksége az alkalmazás használatára, legyen lehetősége kijelentkezni.

Legyen lehetőség új számlák feltöltésére PDF formátumban és mellette a számla metaadatainak megadására, valamint, ha rossz adatokat adtak meg, akkor legyen lehetőség módosítani a megadott metaadatokat. A feltöltött számlákat lehessen listázni, keresni közöttük és szűrni rájuk, valamint le lehessen tölteni őket PDF formátumú fájlba. A számlák törlésére is legyen lehetőség.

A számlák státuszát lehessen állítani. Az alapján legyen erre lehetőség, hogy a felhasználónak van-e jogosultsága az adott művelet elvégzésére. Az első státusz a „New”, a feltöltés utáni első státusz. Ezután lehetőség van a számlák jóváhagyására első és második szinten: az „Accepted by financial controller”, az első jóváhagyási szint; az „Accepted by manager”, a második jóváhagyási szint; az „Accepted by administrator on the first level” és az „Accepted by administrator on the second level”, az adminisztrátor általi első- és másodikszintű jóváhagyás. A számlák elutasítására is van lehetőség: „Rejected by financial controller”, „Rejected by manager”, „Rejected by administrator”. Valamint vissza lehet küldeni a számlákat korrekcióra: „Returned by financial controller”, „Returned by manager”, „Returned by administrator”. Ezután, ha javítva lett a számla, akkor az „Updated” státuszba kerül.

A felhasználók jelszavait biztonságosan kell kezelni, úgy hogy a jelszavak hash értékeit tárolom el és nem magukat a jelszavakat.

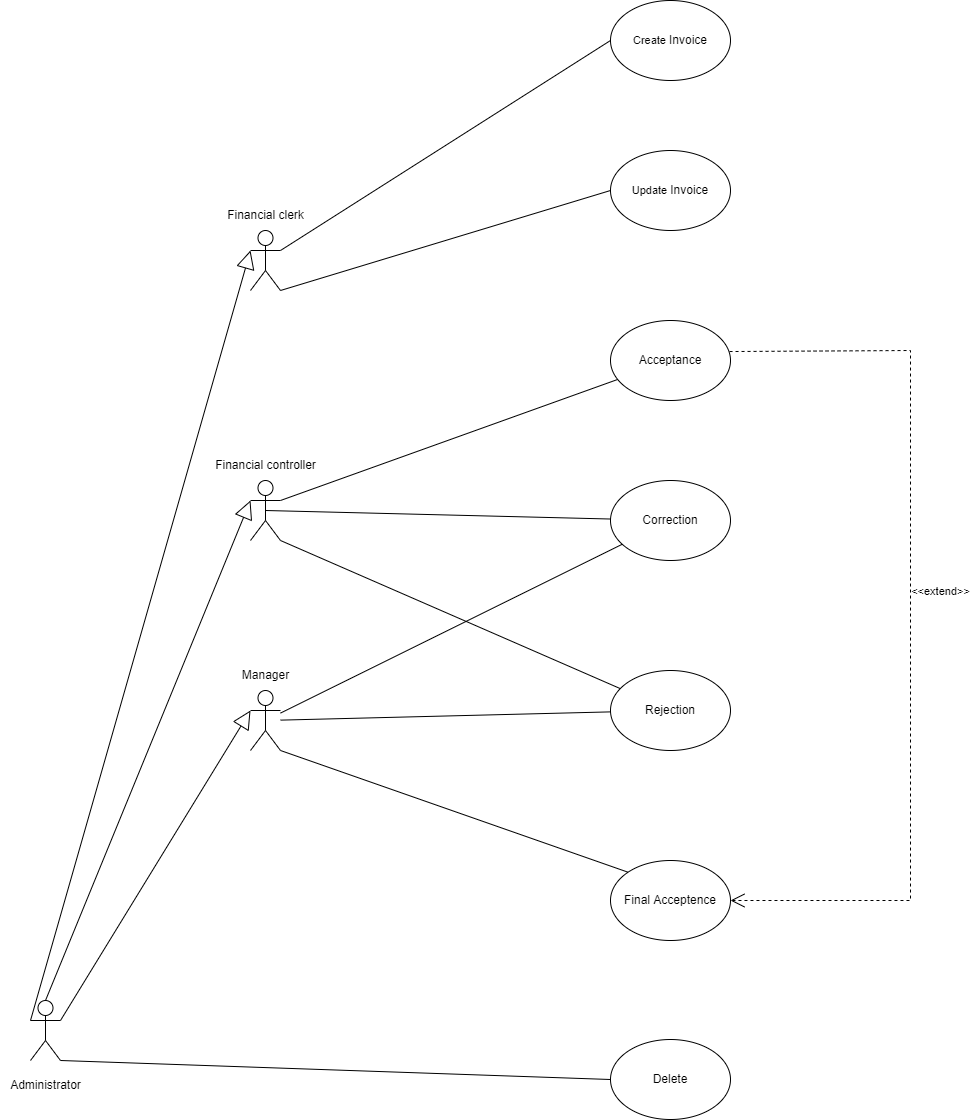
További opcionális követelmények az új felhasználók regisztrációja, a jelszavakra bizonyos megkötések adása. valamint a fájlok validálása feltöltéskor, hogy csak PDF formátumú fájlokat lehessen feltölteni.

## **Nem funkcionális követelmények:**

Nincs meghatározva nem funkcionális követelmény.

# **Áttekintés, solution design**

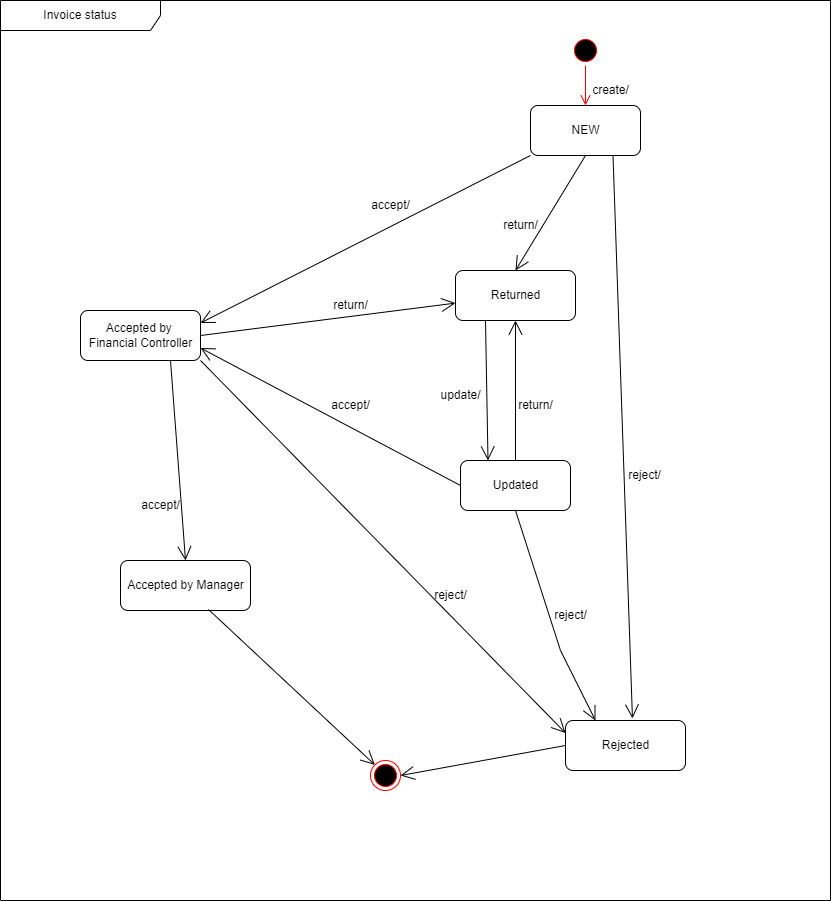
## **Use-case diagram**



A diagram a felhasználók feladatait mutatja be jogosultságaik szerint, ezzel egy számla feldolgozásának állomásait mutatja be.

Láthatók a négy jogosultsági körnek megfelelő felhasználók. A financial clerk feladatai közé tartozik a számlák feltöltése és módosítása. A financial controller feladatai a jóváhagyás első szinten, ezen kívül el is utasíthatja a számlát vagy visszaküldheti korrekcióra a financial clerk-nek. A manager második szintű jóváhagyást adhat a számlákra, és ezen kívül ő is visszaküldheti vagy elutasíthatja a számlát. Az adminisztrátor a másik három jogosultsági kör feladatainak elvégzésére jogosult, de ezeken felül még törölheti is a számlákat.

## **State Machine diagram**



A fenti diagram bemutatja, hogy milyen állomásokon, státuszokon megy keresztül egy számla, amíg végleges elfogadásra vagy elutasításra nem kerül.

Első lépésként létre kell hozni, fel kell tölteni a számlát, ekkor kerül a „New” státuszba. Ezután három lehetőség van: az ’accept’, a ’reject’ és a ’return’. Ha az ’accept’-et választjuk, akkor az „Accepted by Financial Controller” státuszba kerül, ha a ’reject’-et választjuk, akkor „Rejected” státuszt kapja. Ha pedig a ’return’-t, akkor a „Returned”-be.

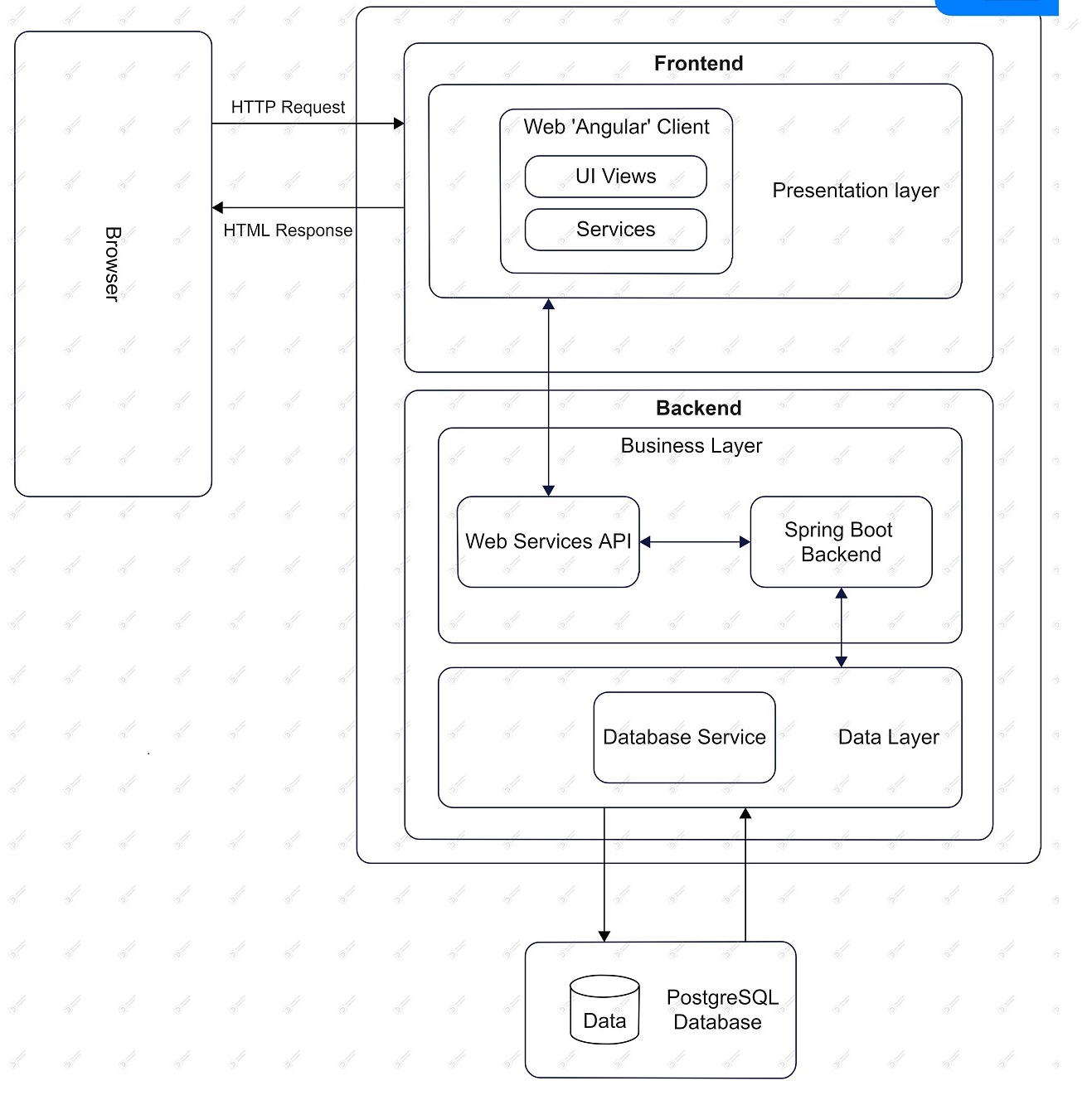
Ha „Accepted by Financial Controller” státuszt kap, akkor szintén három lehetőség van, amik megegyeznek a „New” státusz lehetőségeivel, csak egy kis eltéréss van. Ha az ’accept’-et választjuk, akkor a „Accepted by Manager” státusba kerül, ahonnan a végállapotba kerülünk, mivel megtettük a kétszintű jóváhagyás lépéseit.

A „Rejected” státuszból végállapotba kerülünk, mivel ha visszautasítjuk, akkor nincs már lehetőségünk a számlával további műveletek végzésére.

Ha „Returned” státuszt kap, akkor innen az ’update’ lehetőség van, amivel az „Updated” státuszba kerülhet.

Az „Updated” státuszból szintén három lehetőség van, melyek megegyeznek a „New” státuszból induló lehetőségekkel.

## **Architecture diagram**

****

A fenti diagram egy magasszintű diagram, ami a szoftver szerkezetét mutatja be: a szoftverkomponenseket, azok kapcsolatait és tulajdonságait.

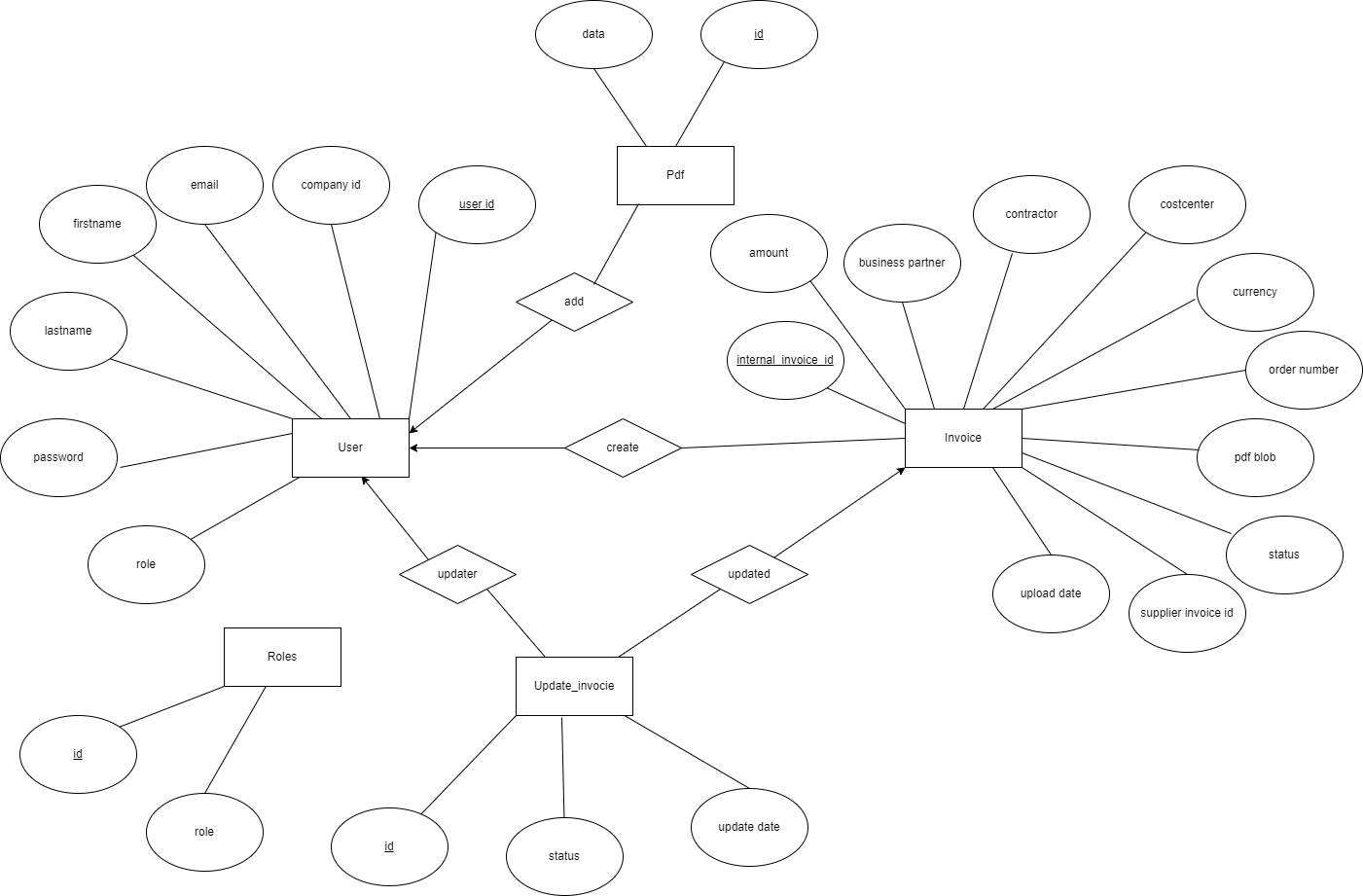
A szoftver egy Frontendből és egy Backendből áll. A Frontend-hez tartozik egy Presentation Layer, ami az Angular alapú webes klienst tartalmazza. Ez a kliens felel a UI kinézetéért és a kommunikációért a browser-rel. A Backendhez tartozik a Business Layer és a Data Layer. A Business Layer-ben található a Web Services API, ami a Frontend és a Backend közötti kommunikációért felelős. A Business Layer-ben szerepel még a Spring Boot alapú Backend, ami a Data Layer-rel való kommunikációért felel, és a Frontend felöl a Web Services API-n keresztül érkező kérésekre az adatbázisból lekér adatokat, amiket aztán visszaküld a Frontend felé szintén a Web Services API-n keresztül. A Data Layer kommunikál az adatbázissal és kér le vagy tölt fel adatokat a postgreSQL adatbázisból.

A diagramon szerepelnek külső komponensek is, ezek a felhasználói interfészek, vagyis a böngészők, browser-ek és az adatbázis, ami ebben az esetben PostgreSQL Database.

Tehát a browser HTTP kéréseket indít a Frontend-nek, ami ezt továbbküldi a Backend-nek a Web Servises API-n keresztül. A Backend közvetlen kapcsolatban áll az adatbázissal és onnan kér le információkat, amiket aztán válaszként visszaküld az API-n keresztül a Frontend-nek, ami átalakítja HTML weboldallá és ezt adja a felhasználónak.

## **Entity Relation diagram**

Az alábbi diagram az egyes egyedeket és a köztük fennálló kapcsolatot ábrázolja. Az egyedek a User, a Pdf, az Invoice, az Update\_invoice és a Roles. Mindegyiknek vannak tulajdonságai és egy ezek között egy olyan tulajdonság, ami egyértelműen beazonosítja, egy id. Az egyedek között lehetnek a kapcsolatok, relációk. Ilyen például a User és az invoice közötti create reláció, ami egy 1:N-hez kapcsolat, tehát egy User több Invoice-t is létrehozhat, de egy Invoice-t csak egy User hozhat létre.

****

## **Interfészek szerepe**

openapi: 3.0.1

paths:

  /api/user/update\_user/{id}:

    put:

      tags:

      - user-controller

      operationId: updateUser

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/User'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

  /api/user/update\_password/{id}:

    put:

      tags:

      - user-controller

      operationId: updatePassword

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      requestBody:

        content:

          application/json:

            schema:

              type: string

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

  /api/invoice/update\_status/{id}:

    put:

      tags:

      - invoice-controller

      operationId: updateStatus

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/UpdateStatus'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

  /api/invoice/update\_invoice/{id}:

    put:

      tags:

      - invoice-controller

      operationId: updateInvoice

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/Invoice'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

  /api/user/add\_user:

    post:

      tags:

      - user-controller

      operationId: addUser

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/User'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                $ref: '#/components/schemas/User'

  /api/upload:

    post:

      tags:

      - pdf-controller

      operationId: uploadFile

      requestBody:

        content:

          application/pdf:

            schema:

              type: object

              properties:

                short:

                  type: integer

                  format: int32

                char:

                  type: string

                int:

                  type: integer

                  format: int32

                long:

                  type: integer

                  format: int64

                float:

                  type: number

                  format: float

                double:

                  type: number

                  format: double

                direct:

                  type: boolean

                readOnly:

                  type: boolean

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: string

  /api/invoice/add\_invoice:

    post:

      tags:

      - invoice-controller

      operationId: addInvoice

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/Invoice'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                $ref: '#/components/schemas/Invoice'

  /api/auth/signup:

    post:

      tags:

      - auth-controller

      operationId: addUser\_1

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/SignupRequest'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

  /api/auth/signout:

    post:

      tags:

      - auth-controller

      operationId: logoutUser

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

  /api/auth/login:

    post:

      tags:

      - auth-controller

      operationId: login

      requestBody:

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/LoginRequest'

        required: true

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

  /api/user/list\_user:

    get:

      tags:

      - user-controller

      operationId: list

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/User'

  /api/user/find/{id}:

    get:

      tags:

      - user-controller

      operationId: findInvoiceByUserId

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                $ref: '#/components/schemas/User'

  /api/invoice/list\_invoice:

    get:

      tags:

      - invoice-controller

      operationId: list\_1

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/Invoice'

  /api/invoice/find/{id}:

    get:

      tags:

      - invoice-controller

      operationId: findInternalInvoiceById

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                $ref: '#/components/schemas/Invoice'

  /api/user/find/delete/{userId}:

    delete:

      tags:

      - user-controller

      operationId: deleteUser

      parameters:

      - name: userId

        in: path

        required: true

        schema:

          type: integer

          format: int32

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

  /api/invoice/find/delete/{id}:

    delete:

      tags:

      - invoice-controller

      operationId: deleteInvoice

      parameters:

      - name: id

        in: path

        required: true

        schema:

          type: integer

          format: int32

      responses:

        "200":

          description: OK

          content:

            '\*/\*':

              schema:

                type: object

                additionalProperties:

                  type: boolean

## **Kommunikáció**

A kommunikáció HTTP REST interfészen keresztül történik. Az interfészen keresztüli kommunikáció nem titkosított.

A regisztráció során biztonságos jelszókezelés történik, hashelt és sózott jelszókat tárolunk az adatbázisban. Bejelentkezéskor a beírt jelszót szintén hasheljük és sózzuk ugyanazzal a sóval, ugyanannyi körben és az eredményt összehasonlítjuk az adatbázisban tárolt jelszóval, amiknek meg kell egyezniük, ha megfelelő a jelszó.

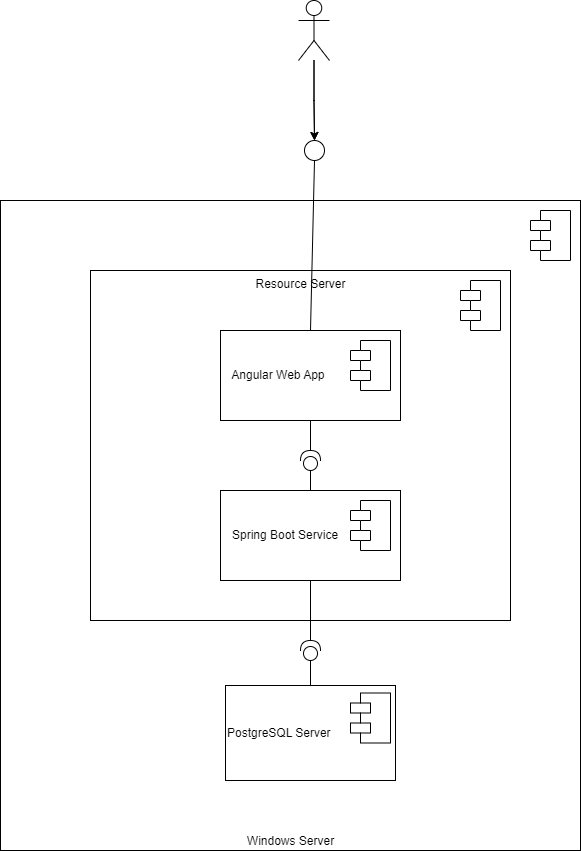
A browser HTTP request-eket küld a Frontend-en és az API-n keresztül a Backend-nek. A mejelenítéseknél HTTP GET response-ban kapja vissza a kér adatokat, amiket a Frontend segítségével megjelenít a browser, például a számlák listáját. Az új adatok hozzáadásánál HTTP POST request-et küld a browser és a request body-jában elküldi az új adatokat a Backend-nek, majd kimenti őket az adatbázisba. Az adatbázisban szereplő elemek módosítására HTTP PUT request-et használunk, aminek a path-jában elküldjük a módosítani kívánt elem id-ját és a body-jában a módosításokat. Az elemek törlésére a HTTP DELETE request-et használjuk.

## **Komponensek**

### Component diagram

Az alábbi diagram a szoftver komponenseit és a köztük fennálló kapcsolatokat mutatja be. A User / Browser számára az Angular Web App biztosít interface-t az alkalmazás eléréséhez.

A Resource Server komponensben található az Angular Web App és a Spring Boot Service komponensek. Az Angular Web App megköveteli a Spring Boot Service által biztosított interface-t, amin keresztül kér le adatokat az adatbázisból. A Spring Boot Service megköveteli a PostgreSQL Server által biztosított interface-t, így eléri az adatbázisban tárolt adatokat.

****

### User / Browser

Minden User rendelkezik jogosultsággal, ami alapján bejelentkezhet és műveleteket végezhet.

### Windows Server

A Windows Server-en fut a maga az alkalmazás.

### Resource Server

Tartalmazza magát az alkalmazást.

#### Angular Server

Lehetővé teszi, hogy a felhasználók bejelentkezzenek és bejelentkezés után a jogosultságaiknak megfelelő adatok megjelenítését az adatbázisból.

Ha a felhasználónak nincs jogosultsága megtekinteni az egyes adatokat, akkor nem láthatja bejelentkezés után sem.

Amire a felhasználónak van jogosultsága, annak a megtekintéséhez, alkalmazásához megjelennek a megfelelő gombok.

#### Spring Boot Server

Ez felel a PostgreSQL Server-ről érkező adatok feldolgozásáért és a frontend felöl érkező kérésekre válaszként a megfelelő szolgáltatást biztosítja, ha a kérést indító felhasználó jogosult rá. Ha nem jogosult rá, akkor hibaüzenetet küld.

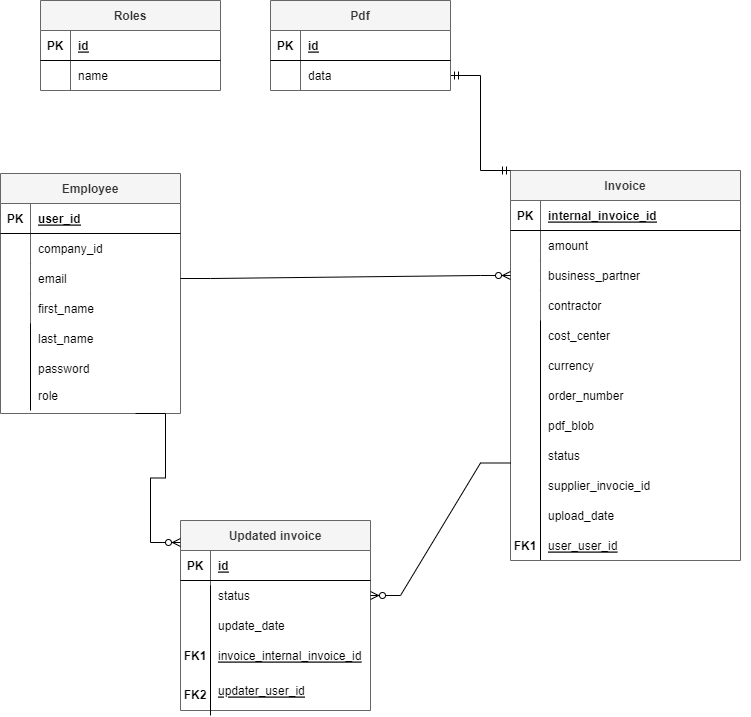
Több végponttal rendelkezik az elvégezhető feladatoknak megfelelően, amiket fentebb már tárgyaltunk.

#### PostgreSQL Server

Tartalmaz egy ’ManagementDB’ nevű adatbázist. Az adatbázis 5 táblát tartalmaz. Ezek a táblák az ’Employee’, ami a felhasználók adatait tárolja; az ’Invoice’, ami afeltöltött számlákat tárolja; a ’Pdf’, ami ideiglenesen tárolja a feltöltött pdf formátumú számlák dokumentumait, amíg hozzá nem adjuk azt a megfelelő számlához az ’Invoice’ táblában. Van még egy ’Roles’ tábla, ami felhasználó szerepköreit tartalmazza. Az utolsó tábla az ’Updated\_invoice’, ami a módosított számlákra és a módosító felhasználókra mutató referenciát tartalmazzák, a módosítás dátumával és a módosított adattal.

# **Technikai leírás**

## **Database Relationship diagram**



A fenti diagramon az adatbázis táblák és kapcsolataik szerepelnek. Öt tábla szerepel az adatbázisban, az ’Employee’, az ’Invoice’, a ’Pdf’, az ’Updated Invoice’ és a ’Roles’. Minden táblának van Primary Key-e, egy id-je, ami alapján azonosítható egy egyed.

Az ’Employee’ és az ’Invoice’ tábla között egy ManyToOne kapcsolat van, ami alapján Egy ’Invoice’ példányhoz egy ’Employee’ példány tartozhat és egy ’Employee’ példányhoz több ’Invoice’ példány tartozhat. Ez az ’Employee’ példány az ’Invoice’ példány létrehozója, aki feltöltötte az adatbázisba a számlát.

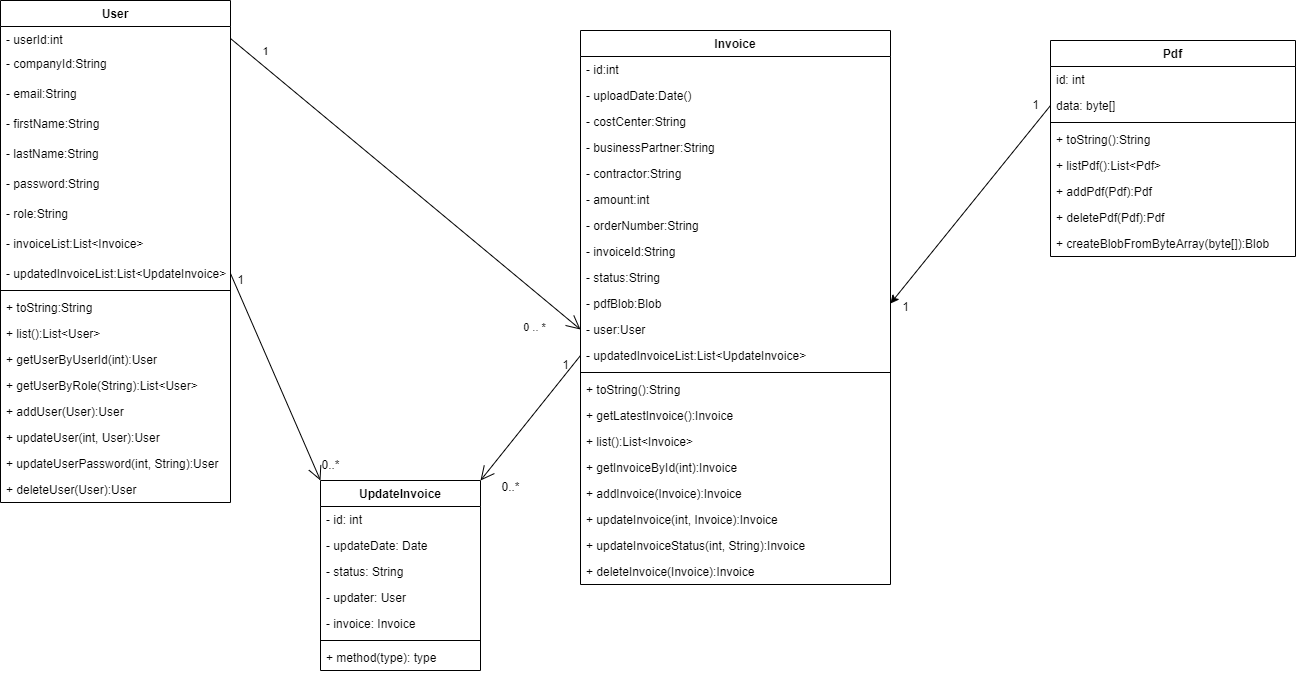
Az ’Invoice’ és a ’Pdf’ között OneToOne kapcsolat van. Egy ’Invoice’ példányhoz egy ’Pdf’ példány tartozik. Ez a számla létrehozásánál használatos. A pdf formátumú számlát feltöltjük a ’Pdf’ táblába, aztán feltöltjük a számla adatait és a ’Pdf’ táblába mentett dokumentumot az ’Invoice’ táblába. Végül töröljük az adatbázisban szereplő ’Pdf’ tábla tartalmát, mivel ide csak ideiglenesen mentettük ki.

Az ’Employee’ és az ’Updated Invoice’ tábla, valamint az ’Invoice’ és az ’Updated Invoice’ tábla között OneToMany kapcsolat áll fenn. Egy ’Updated Invoice’-hoz csak egy ’Employee’ és egy ’Invoice’ példány tartozhat, de egy ’Employee’ és egy ’Invoice’ példányhoz több ’Updated Invoice’ példány is tartozhat. Ez az ’Invoice’ tábla példányainak módosítása esetén használt.

A ’Roles’ táblát az egyes felhasználók szerepeinek megadásánál hasznájuk.

## **Class diagram**

Az alábbi diagramon az alkalmazásban szereplő osztályok kerültek feltüntetésre az adattagjaikkal és metódusaikkal. Ezekből az osztályokból készültek a fent látott adatbázis táblák.



## **JWT**

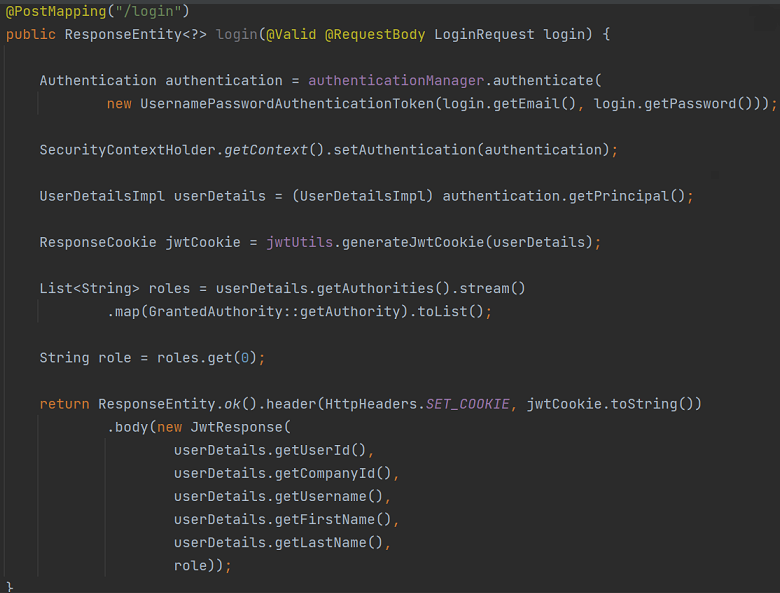
A JSON Web Application biztonságos információ átadásra szolgál JSON formátumban. A JWT-t az authorizációnál használt token, ami három részból áll: Header, Payload, Signature (aláírás).

A ’generateJwtCookie(UserDetailsImpl userPrincipal)’ metódussal generálunk egy JWT-t tartalmazó Cookie-t

* a generateJwtToken(String username) segítségével, a username-ből, a dátumból, a lejáratból és a titkosításból
* A jwtCookie-t az ’application.properties’-ben adtuk meg
* A ResponseCookie-t a jwtCookie-ból és a generált JWT Token-ből képezzük, és megadjuk még, hogy milyen útvonalon működjön, hogy mennyi ideig legyen használható és csak HTTP-n keresztül legyen küldhető
* A titkosítás egy magánkulccsal történik, amit szintén az ’application.properties’-ben adtuk meg és a hozzá tartozó publikus kulccsal lehet feloldani, ami a USER\_KEY

## **Bejelentkezés**

Bejelentkezéskor a Frontend elküldi a Backend-nek a felhasználónevet és a jelszót, amit megadtak. Ezek hitelesítése az AuthenticationManager authenticate metódusának segítségével történik. Majd ez alapján beállítjuk az authentication-t. Ebből lekérjük az adatokat a UserDetailsImpl-be és létrehozzuk a JWT-t tartalmazó Cookie-t, aztán lekérem külön a user role-ját is. Végül visszaküldöm a Frontend-nek válaszként a header-ben a Cookie beállítását, body-ban pedig a user adatait.



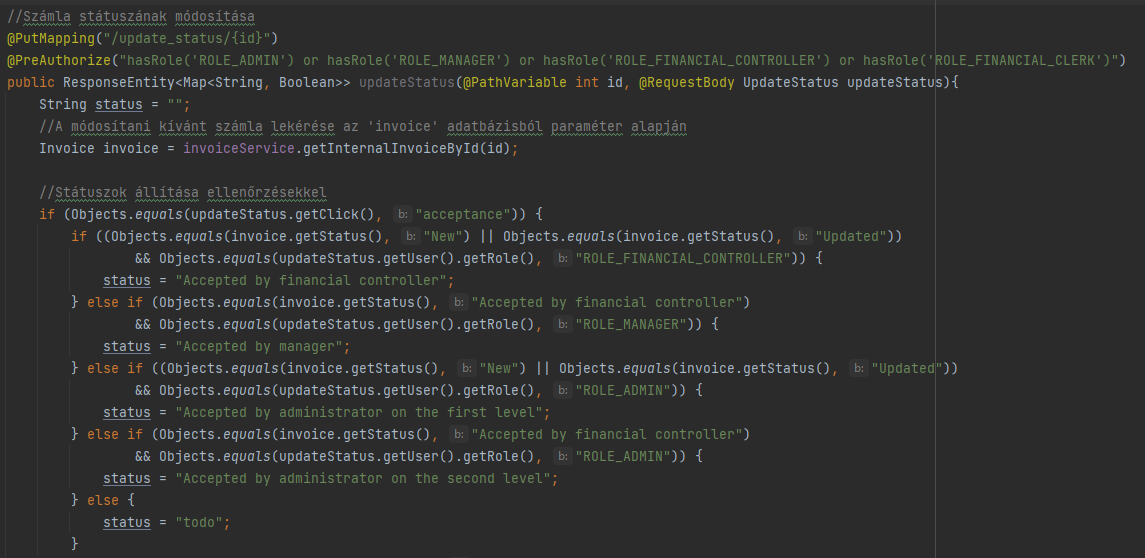
A user role-ja alapján történik az információk, lehetőségek megjelenítése a jwtCookie-ból lekért role-ok alapján. Valamint a Backend-en is szerepelnek megkötések a szerepkörök szerint. Pl.: @PreAuthorize("hasRole('ROLE\_ADMIN') or hasRole('ROLE\_FINANCIAL\_CLERK')")

## **Számla jóváhagyási folyamata**

A számlák jóváhagyása hardcode-olva készült el. Az alább látható függvény egy részlete mutatja, hogy hogyan készült az implementáció.

A Frontend-től Backend megkapja a módosítandó invoice internal\_invoice\_id-ját, az hogy melyik gombra kattintottak (Add Invoice/Accept/Reject/Return/Updated) és hogy ki tette ezt.

Először egy if-elseIf-fel ellenőrizzük, hogy melyik gombra kattintottunk, ha ez megvan, akkor szintén if-elseIf-ekkel ellenőrizzük, hogy az Invoice aktuálisan megfelelő státuszban van-e a státusz állításához és hogy jogosult-e rá a user, hogy módosítsa, tehát lekéri a role-ját is. Ha minden megfelel, akkor a státusz módosításra kerül a következő státuszba. Valamint a módisítás részleteit kimentjük az Updated\_Invoice táblába az adatbázisba.



## **Jelszó titkosítása**

A jelszó titkosítása a bcrypt jelszó hashelő függvény segítségével történik. A hashelés mellett sót is alkalmaz, ami nagyobb védelmet biztosít a szivárvány-tábla alapján indított támadások ellen, később is növelhető az iterációk száma, ami még inkább véd a tádamádos ellen.

Ehhez a ’PasswordEncoder’ interfészt imlementáló ’BcryptPasswordEncoder’ osztályt használja az alkalmazás, ami megvalósít BcryptPasswordEncoder konstruktorokat különböző paraméterezéssel. Itt állítja be a hashelés verzióját, a hashelés erősségét és egy random generált értéket. Itt található még egy encoder, egy salt-ot generáló függvény és egy egyezést vizsgáló.

# **Problémák**

## **Bugok**

* Nincsenek bugok, valószínűleg az alapos tesztelés hiánya miatt

## **Errorok**

* Szűrésnél, ha egy elemet akarunk törölni a szűrések listájából
* Szűrésnél bizonyos esetekben nem működik
* Hibás felhasználói adatok megadásakor bejelentkezésnél
* Felugró ablakoknál, ha az ablak mellé kattintunk

## **Security rizikók**

* Könnyen kitalálható felhasználói jelszavak
* Regisztáció és bejelentkezés esetén nem titkosított kapcsolaton küldött adatok
* JWT nem titkosított
* SQL injekciózás kockázatok

# **Továbbfejlesztési lehetőségek**

Az alkalmazás továbbfejlesztésére sok lehetőség áll még fenn mind a használhatóság megkönnyítésére, mind védelem szempontjából, mind funkcionalitás terén.

## **Használhatóság (Angular Server):**

* Az adatok megjelenítésének átalakítása, hogy jobban átlátható legyen
* Egyes oszlopokra szűrésnél egyes szűrt oszloptípusok külön törlése
* Date formátumú elemek formázása
* Görgetés nélkül látható az összes gomb és adat

## **Védelem (Spring boot Server):**

* https protokol használata a bejelentkezésnél és a regisztrációnál
* JWT titkosítása

## **Funkcionalitás (Spring boot Server):**

* Több felhasználó egyszerre használja, többszálon fusson
* Hibakezelések bővítése
* Ismétlődő kódok kiszervezése függvényekbe
* Hibás bejelentkezési adatok esetén hibajelzés
* Felurgó ablakoknál sima kilépés, ha mellékattintunk
* Egy adminisztrátor csak egyszer módosíthassa a számlák státuszát
* Számlák módosítása esetén, a számlák pontos adatainak kimentése
* Password complexity rules megadása
* Az invoice currency megadásakor egyéb lehetőségbe input mező beállítása
* Csak PDF formátumú elemeket lehessen feltölteni
* Új user hozzáadása esetén email-ben jelszó küldése a megadott e-mail címre, majd a user módosíthassa a jelszavát

## **PostgreSQL Server**

* ’Pdf’ tábla megszüntetése és egyből az ’Invoice’ táblába mentés
* Magának a pdf dokumentumnak a törlése az adatbázisból
* ’Roles’ tábla törlése és az ehhez tartozó kódok átírása

# **Használt szoftverek, architektúrák**

## **Munkakörnyezet specifikációja**

Fujitsu Lifebook U747

CPU: Intel Core i5-6200U

RAM: 8 GB

SSD(M.2): 250 GB

Operációs rendszer: Windows 10 Pro

## **Szoftverek**

Google Chrome (Version 105.0.5195.102)

Windows Services (Version 10.0.19041.746)

IntelliJ IDEA Community (Version 2022.1.2)

Visual Studio Code (Version 1.72.2)

Postman (Version 10.0.32)

pdAdmin 4

## **Használt technológiák**

PostgreSQL database

HTTP REST API

Java Spring Boot

Java Spring Security

Angular

# **Referenciák**

## **Angular**

* <https://ng-bootstrap.github.io/#/getting-started>
* <https://getbootstrap.com/docs/5.2/getting-started/introduction/>
* <https://material.angular.io/components/autocomplete/overview>
* <https://www.bezkoder.com/angular-14-jwt-auth/>
* <https://stackblitz.com/edit/angular-advanced-search-box?file=src%2Fapp%2Fupload-search-box%2Fupload-search-box.component.ts,src%2Fapp%2Fupload-search-box%2Fupload-search-box.component.html>
* <https://dev.to/idrisrampurawala/creating-a-search-filter-in-angular-562d>
* <https://rxjs.dev/deprecations/subscribe-arguments>
* <https://angular.io/docs>
* <https://blog.angular-university.io/angular-file-upload/>
* <https://www.javatpoint.com/angular-spring-file-upload-example>
* <https://spring.io/guides/gs/uploading-files/>
* <https://www.faqcode4u.com/faq/150443/angular-6-unable-to-convert-base64-to-pdf>
* <https://www.youtube.com/watch?v=5zQPLd04gHE&list=PLUErynknWn5WicSgvQLf7FAFaLROGe-Hb&index=3>
* <https://github.com/jankiz/Web-development-frameworks>
* <https://www.youtube.com/watch?v=x6ObfaURc-s>
* <https://www.w3schools.com/jsrEF/met_storage_clear.asp>

## **Fullstack**

* <https://www.baeldung.com/spring-boot-angular-web>
* <https://www.youtube.com/watch?v=Gx4iBLKLVHk>
* <https://www.bezkoder.com/spring-boot-angular-14-postgresql/>
* <https://www.bezkoder.com/spring-boot-postgresql-example/#google_vignette>
* <https://frontbackend.com/spring-boot/spring-boot-2-angular-11-upload-file-example>
* <https://www.javaguides.net/2020/09/angular-9-spring-boot-basic-auth-login-logout.html>
* <https://www.youtube.com/watch?v=r4VfwBMx-H0>

## **JPA és PostgreSQL database server**

* <https://www.youtube.com/watch?v=y5Ytqjr7SaA>
* <https://www.youtube.com/watch?v=UHySlGmrJYg>
* <https://www.javatpoint.com/jpa-one-to-one-mapping>
* <https://www.postgresql.org/docs/8.1/index.html>
* <https://dzone.com/articles/all-jpa-annotations-mapping-annotations>
* <https://www.baeldung.com/jpa-entities>
* <https://self-learning-java-tutorial.blogspot.com/2019/07/jpa-column-precision-specify-precision.html>
* <https://www.youtube.com/watch?v=wdYgpGxDn9E>
* <https://docs.jboss.org/hibernate/orm/5.3/userguide/html_single/Hibernate_User_Guide.html#associations-many-to-one-example>
* <https://www.baeldung.com/hibernate-entitymanager>
* <https://github.com/bezkoder/spring-boot-security-postgresql>
* <https://spring.io/guides/gs/accessing-data-jpa/>
* <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
* <https://www.geeksforgeeks.org/spring-data-jpa-attributes-of-column-annotation-with-example/?ref=rp>
* <https://codehandbook.org/angular-download-pdf-from-base64-string/>

## **Spring Boot és Spring Security**

* <https://www.geeksforgeeks.org/http-headers-access-control-allow-credentials/>
* <https://jwt.io/introduction>
* <https://www.bezkoder.com/spring-boot-login-example-mysql/>
* <https://www.bezkoder.com/spring-boot-jwt-mysql-spring-security-architecture/>
* <https://www.bezkoder.com/spring-boot-jwt-authentication/>
* <https://www.bezkoder.com/spring-boot-login-example-mysql/>
* <https://www.baeldung.com/spring-security-basic-authentication>
* <https://spring.io/guides/gs/securing-web/>
* <https://spring.io/projects/spring-boot>
* <https://www.baeldung.com/jackson-custom-serialization>
* <https://www.baeldung.com/jackson-annotations>
* <https://www.youtube.com/watch?v=tLBX9fq813c&list=PLGRDMO4rOGcNzi3CpBWsCdQSzbjdWWy-f>
* <https://www.baeldung.com/spring-profiles>
* <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using.build-systems.starters>
* <https://www.javatpoint.com/how-to-generate-random-number-in-java>
* <https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm>
* <https://docs.spring.io/spring-security/site/docs/3.2.8.RELEASE/reference/htmlsingle/#jc-method>
* <https://docs.spring.io/spring-security/site/docs/4.1.3.RELEASE/guides/html5/form-javaconfig.html>
* <https://www.javainuse.com/spring/ang7-login>
* <https://www.javainuse.com/spring/boot_swaggersec>
* <https://www.baeldung.com/spring-rest-openapi-documentation>
* <https://springdoc.org/#spring-webmvc-support>

### **Blob kezelés**

* + <https://github.com/eligrey/FileSaver.js/>
  + <https://javascript.info/blob>
  + <https://www.programcreek.com/java-api-examples/?api=java.sql.Blob>
  + <https://github.com/uglide/azure-content/blob/master/articles/storage/storage-java-how-to-use-blob-storage.md>
  + <https://www.youtube.com/watch?v=HtyYKRuAU2c>
  + <https://www.youtube.com/watch?v=uZXfZZ59cjU>
  + <https://stackoverflow.com/questions/17400497/how-to-convert-blob-to-string-and-string-to-blob-in-java>
  + <https://www.tabnine.com/code/java/methods/java.sql.Blob/%3Cinit%3E>
  + <https://www.gnostice.com/nl_article.asp?id=165&t=How_To_Save_a_PDF_Document_To_A_Database>
  + <https://developer.mozilla.org/en-US/docs/Web/API/Blob>
  + <https://stackoverflow.com/questions/53731136/making-a-post-request-that-returns-a-pdf-using-angulars-httpclient>