Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a tanszék, a hallgató, a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre (Fájl/Információ/Tulajdonságok/Speciális tulajdonságok).

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Varga Ádám Marcell

**Multitenant ötletkezelő alkalmazás Spring alapokon**

Konzulens

Dr. Forstner Bertalan

BUDAPEST, 2024

Tartalomjegyzék

[Összefoglaló 7](#_Toc180852570)

[Abstract 8](#_Toc180852571)

[1 Bevezetés 9](#_Toc180852572)

[2 Motiváció 10](#_Toc180852573)

[3 Felhasznált technológiák 11](#_Toc180852574)

[3.1 Angular 11](#_Toc180852575)

[3.1.1 Angular CLI 11](#_Toc180852576)

[3.1.2 Angular Material 12](#_Toc180852577)

[3.2 TypeScript 13](#_Toc180852578)

[3.3 Spring Boot 13](#_Toc180852579)

[3.3.1 Kotlin 13](#_Toc180852580)

[3.3.2 Spring Data JPA 13](#_Toc180852581)

[3.3.3 MockK 13](#_Toc180852582)

[3.4 MySQL 13](#_Toc180852583)

[3.5 Docker 13](#_Toc180852584)

[3.6 AWS (Amazon Web Services) 13](#_Toc180852585)

[3.7 Figma 13](#_Toc180852586)

[3.8 Git/GitHub 14](#_Toc180852587)

[4 Tervezés 15](#_Toc180852588)

[4.1 Magas szintű követelmények 15](#_Toc180852589)

[4.2 Funkcionális követelmények 15](#_Toc180852590)

[4.3 Architektúra tervezés 16](#_Toc180852591)

[4.3.1 Alkalmazás felépítése 16](#_Toc180852592)

[4.3.2 Multitenancy 17](#_Toc180852593)

[4.4 Modell tervezés 20](#_Toc180852594)

[4.4.1 User 21](#_Toc180852595)

[4.4.2 Role 21](#_Toc180852596)

[4.4.3 IdeaBox 21](#_Toc180852597)

[4.5 UI Tervezés 22](#_Toc180852598)

[5 Implementáció 23](#_Toc180852599)

[6 Összefoglalás 24](#_Toc180852600)

[Irodalomjegyzék 25](#_Toc180852601)

[Függelék 26](#_Toc180852602)

Hallgatói nyilatkozat

Alulírott **Rezeda Kázmér**, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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 hitelesített 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.

Kelt: Budapest, 2024. 10. 26.

...…………………………………………….

Összefoglaló

Az Ötletkezelő alkalmazások, más néven Idea Management alkalmazások, olyan digitális eszközök és platformok, melyeket vállalatok és szervezetek használnak az innováció ösztönzésére és az ötleti kreativitás folyamatának támogatására. Ezek az alkalmazások kiterjednek a koncepciók gyűjtésétől a projektek végrehajtásáig, lehetővé téve a szervezeteknek a rendszeres ötletgenerálást, a közösségi együttműködést és az ötletek hatékony menedzselését.

A fő jellemzők közé tartozik az ötletgyűjtés és rangsorolás, ahol a felhasználók könnyedén megoszthatják ötleteiket a platformon, és mások értékelhetik, rangsorolhatják azokat. A kollaborációt támogató eszközök segítik a csapatoknak az ötletek közös kidolgozásában és megosztásában, miközben a kétirányú kommunikáció elősegíti a konstruktív visszajelzéseket.

Az Ötletkezelő alkalmazások a visszajelzések és fejlesztés fontos szakaszában is segítenek, ahol a vezetőség és a felhasználók visszajelzése alapján az ötleteket továbbfejleszthetik vagy elvetik. Az átláthatóság és analitika révén a vállalatok nyomon követhetik az ötletgenerálási és fejlesztési folyamatokat, és az eszköz analitikai képességei segítik a vezetőséget a stratégiai döntéshozatalban.

Ezen funkciók együttesen segítik a vállalatokat abban, hogy ösztönözzék a kreativitást, fokozzák az innovációt és gyorsítsák fel az ötletek megvalósítását. Az Ötletkezelő alkalmazások így nemcsak a vállalati kultúra fejlesztésében játszanak kulcsszerepet, hanem hozzájárulnak a versenyképesség fenntartásához és a hosszú távú siker eléréséhez a dinamikus üzleti környezetben.

A dolgozatomban egy ilyen alkalmazásnak fogok egy lehetséges implementációt készíteni. A kliens oldali alkalmazás fejlesztéséhez Angular-t, a szerver fejlesztéséhez Spring Bootot használtam.

Abstract

Idea Management applications, also known as Idea Management tools, are digital tools and platforms used by companies and organizations to stimulate innovation and support the creative process of generating ideas. These applications encompass the entire spectrum from idea conception to project execution, allowing organizations to engage in regular idea generation, foster collaborative teamwork, and efficiently manage ideas.

Key features include idea collection and ranking, where users can easily share their ideas on the platform, and others can evaluate and rank them. Tools supporting collaboration assist teams in jointly refining and sharing ideas, while bidirectional communication facilitates constructive feedback.

Idea Management applications also assist in the crucial stage of feedback and development, where ideas can be further developed or discarded based on feedback from both management and users. Through transparency and analytics, organizations can track the processes of idea generation and development, and the analytical capabilities of the tool aid leadership in strategic decision-making.

Collectively, these features help organizations stimulate creativity, enhance innovation, and expedite the realization of ideas. Idea Management applications not only play a key role in developing corporate culture but also contribute to maintaining competitiveness and achieving long-term success in the dynamic business environment.

In my paper, I will present a possible implementation of such an application. For the development of the client-side application, I utilized Angular, and for the server-side development, I used Spring Boot.

# Bevezetés

//TODO

//ötletkezelő bemutatása

//Multitenancy bemutatása

# Motiváció

# Felhasznált technológiák

## Angular

Az Angular egy olyan platform és keretrendszer, amelyet a single-page webalkalmazások (SPA) fejlesztéséhez terveztek. Ez azt jelenti, hogy az alkalmazás egy HTML fájlt tölt be a böngészőben, és futás közben JavaScript segítségével módosítja azt a megjelenítendő nézeteknek megfelelően, nem pedig egy új HTML oldalt tölt be.

Az Angular architektúrája modulokra és komponensekre épül. A komponensek lehetővé teszik, hogy a projektben kisebb vagy akár nagyobb kódrészletek újrahasznosíthatóak legyenek. Ez megkönnyíti az új elemek létrehozását vagy a meglévő elemek egységes átalakítását, mivel bizonyos esetekben csak egy fájlt kell módosítani, és az mindenhol megváltozik. A modulok logikai részeket csoportosítanak össze. Egy Angular alkalmazás több modulból állhat, de mindenképpen tartalmaz egy gyökérmodult.

Az Angular 2 és az azt követő verziókban a fejlesztés két fő nyelven történik: HTML és TypeScript (TS). Ezenkívül a weboldalunk megjelenését CSS fájlokkal módosíthatjuk. Alapvetően egy komponens egy HTML fájlból, egy TypeScript fájlból (és esetenként egy CSS fájlból) áll. Az HTML fájlt általában sablonfájlnak nevezik, ami meghatározza a komponens szerkezetét, és ezt tölti be a Dokumentum Objektum Modelljébe (DOM), amikor a komponens betöltődik. Ehhez tartozik egy TypeScript fájl, ahol a komponens logikáját és metódusait lehet leírni. A megjelenítendő adatokat itt deklarálják, és itt kezelik a bejövő és kimenő adatokat. Ha a komponenshez tartozik CSS fájl, akkor abban lehet a komponensre szabott stílusokat alkalmazni. Alapvetően az Angular-ban ezek a stílusok csak arra a komponensre vonatkoznak, amelyhez tartoznak, de ezt a funkciót ki lehet kapcsolni.

### Angular CLI

Az Angular CLI (Command Line Interface) egy eszköz, amely lehetővé teszi az Angular alkalmazások egyszerűbb létrehozását, fejlesztését és kezelését parancssori utasítások segítségével. Az CLI legfontosabb funkciója az Angular projekt csomagolása. Ezt úgy érjük el, hogy lefordítjuk a TypeScript fájlokat, majd összecsomagoljuk a kapott JavaScript fájlokat, HTML fájlokat, CSS-t (vagy más, esetleg külön fordítást igénylő 9 stíluslapokat, például SCSS-t). Az összecsomagolás előnye, hogy az egész projektet csak néhány fájl importálásával lehet használni, nem szükséges mindegyiket külön importálni.

A fejlesztés támogatásához az CLI számos parancssori utasítást nyújt a fejlesztő számára, közülük a legfontosabbak:

* **ng new**

Új Angular projektet hoz létre. Utána paraméterként írható név, ami a projekt megnevezése lesz. A generálás során több opciót is meg lehet adni parancssori kérdés-válasz formában

* **ng generate**

A meglévő projektbe generál új komponenseket, szolgáltatásokat, direktívákat, moduloka és még sok mást.

* **ng serve**

Az Angular alkalmazás futtatása, alapértelmezetten a localhost:4200-on. Ha fut az Angular projekt, akkor az automatikus változásdetektálás miatt, ha a fájlok változnak, az oldal frissül, ami megkönnyíti a dinamikus fejlesztést.

* **ng build**

Az alkalmazás lefordítását és összecsomagolását végzi el. A parancs végrehajtása után az alkalmazás készen áll, hogy egy szerveren kiszolgálja a felhasználókat.

### Angular Material

Az Angular Material egy UI függvénykönyvtár, amit a Google fejleszt. Előre definiált komponenseket és felületi elemeket tartalmaz, amik könnyen és egyszerűen alkalmazhatóak az alkalmazásokban. A komponensek a Material Design irányelveit követik, elérhetőek több változatban, de a személyre szabásra is nyújt lehetőséget a könyvtár. Használatuk egy csomagkezelővel (például npm) történő telepítés után nagyon egyszerű. A komponensek modulokra vannak osztva, melyeket külön-külön lehet importálni a projektbe, így csak az kerül bele, amire valóban szükség van.

## TypeScript

A TypeScript egy nyílt forráskódú, JavaScript alapú programozási nyelv, amit a Microsoft fejlesztett ki. A nyelv kiemelkedő tulajdonsága, hogy támogatja mind az explicit, mind az implicit típusokat, ami miatt objektumorientált. A TypeScript kódja JavaScriptre fordul le, így a meglévő JavaScript kód is működik TypeScript fájlokban. Ez különösen előnyös a webfejlesztés területén, mivel a legtöbb weboldal kliensoldali kódja JavaScript. Ráadásul a fejlesztőknek nem szükséges más nyelvet tanulniuk a szerveroldali fejlesztéshez sem, mivel a Node.js JavaScriptet futtat. Ennek köszönhetően egyetlen nyelv használatával, kizárólag TypeScript segítségével lehet teljes (full stack) alkalmazást írni.

## Spring Boot

### Kotlin

### Spring Data JPA

### MockK

## MySQL

A MySQL egy relációs adatbázis, amit először a MySQLAB cég fejlesztett, de ma már az Oracle tulajdonában áll. Még ma is az egyik legelterjedtebb és legtöbbet használt adatbázis-kezelő, ami elérhető az interneten. Mivel relációs adatbázis, az adatokat táblákban tárolja, a táblákon belül sorokba rendezve.

## Docker

## AWS (Amazon Web Services)

## Figma

A Figma egy modern, webes felhasználói felülettervező alkalmazás akár drótvázak (wireframe) akár teljes UI design-ok tervezésére. A vektorgrafikus tervezőfelületen több forma, alakzat elkérhető, emellett minden alakzatnak beállíthatunk különböző tulajdonságait. Ezek a tulajdonságok az egész alap funkcióktól, mint például a szín, vonalvastagság, egészen az igen specifikus részletekig terjednek, mint például a különböző effektek (árnyékok, homályosítás és még sok más). A felület lehetőséget biztosít rétegek, rétegcsoportok kialakítására, amitől sokkal átláthatóbb lesz a készülő felület. Egy adott alakzatcsoportból komponensek készíthetőek, amik utána újra felhasználhatóak, illetve változtatásukra minden 11 előfordulásnál megváltoznak. Ezen komponenseknek lehetnek variánsai is, amik színesítik a látványtervet. A felületen lehetőség van prototípusokat létrehozni a design-hoz, amitől az interaktív lesz. Beállítható minden alakzatra, hogy például kattintásra melyik nézet jelenjen meg, milyen adatok jelenjenek meg. Ez a tervezési folyamat során, illetve például egy megrendelői bemutatásnál sok félreértést el tud oszlatni ez a funkció.

## Git/GitHub

Manapság a legelterjedtebb verziókezelő rendszer. Használata során létrejön egy saját adattár minden fejlesztőnél, ahova a kódbázis kerül. Ha ezen a kódbázison változtatás történik, akkor a két fájl közötti változás lokálisan elmentésre kerül. Ezen változtatásokat a fejlesztők hozzáadhatják a kódbázishoz, ezáltal az megváltozik. Ekkor a többi fejlesztő, aki ugyanazt a kódbázist használja már egy elavult kódbázist birtokol, nekik frissíteni kell a saját kódbázisukat, mielőtt a saját változtatásaikat feltöltenék. Ez a folyamat több kódösszeakadást eredményezne, ezért a verziókezelők több ágon (branch) működnek. Egy-egy ágat le lehet származtatni a fő ágról (ez általában egy master ág) így a változások frissítése csak azokat érintik, akik ugyan azon az ágon dolgoznak. Általában egy ág lehet például egy funkció is, de akár egy egész modul. Ha a fejlesztett ág hibátlannak bizonyul, akkor a fejlesztők beleolvaszthatják (merge) a fő ágba a fejlesztéseiket. Ezzel biztosítva van (jó esetben), hogy a fő ágra csak működő kód kerül A projektben, mivel egyedül dolgoztam, ezért csak adatok tárolására használtam a Git egy ingyenes, weben és asztali alkalmazás formájában is elérhető szolgáltatóját, a GitHubot

# Tervezés

Ebben a fejezetben az alkalmazás tervezéséről lesz szó. Először bemutatom az alkalmazás követelményeit, majd a modell és UI tervezését, ezek után pedig az architektúra tervezésének lépéseit.

## Magas szintű követelmények

Az alkalmazás célja, hogy a regisztrált felhasználók a saját környezetükben ötleteket vessenek fel egy adott témában. Ezeket a témákat a szervezet vezetői szabják meg ötletdobozok létrehozásával.

Az ötletdobozok rendelkeznek egy egységes értékelési lappal, amellyel az összes beérkező ötlet pontozva lesz így biztosítva azt, hogy objektíven és egységesen legyenek elbírálva az ötletek. Emiatt az egységes értékelés miatt csak azok az ötletek kerülnek megvalósításra, amelyek az adott kereteken belül a legjobb eredményt nyújtják.

A rendszernek biztonságosnak is kell lennie, mivel a szolgáltatást több szervezet is használja. Figyelni kell arra, hogy a felhasználói jogosultságok jól legyenek beállítva a rendszeren belül is, illetve arra is, hogy a felhasználók ne érjenek el olyan adatokat, amiket egy másik szervezet használ és fordítva.

## Funkcionális követelmények

## Architektúra tervezés

### Alkalmazás felépítése

Az alkalmazás tervezése során három rétegű architektúra mellett döntöttem, mivel ebben a megoldásban jól el tudom szeparálni a fő komponenseket (kliens oldal, szerver oldal, adatréteg)

A megjelenítési (kliens oldal) réteghez Angular 14-et használtam, mivel ebben a keretrendszerben már volt előzőleg tapasztalatom. Az Angular keretrendszer egy jó választás közepes és nagy méretű projektek kliens oldali megvalósításához, mivel rengeted könyvtár létezik hozzá a különböző komponensek gyors megvalósítására, amik nem mellesleg követik a Material design alapelveit. Ezáltal a fejlesztés gyors ütemben tud haladni.

Az üzleti logikai réteg megvalósításához a Spring Boot keretrendszer mellett döntöttem, mivel ez a keretrendszer már régóta az egyik legjobban elterjedt szerver oldali keretrendszer tele olyan beépített megoldásokkal, amik segíthetnek az alkalmazás megvalósításában. Emellett a Spring támogatja a Kotlin nyelvet, amely egy sokkal modernebb, olvashatóbb nyelv a Java-hoz képest, így a fejlesztés benne sokkal egyszerűbb és gyorsabb.

Az kliens oldali réteget és az üzleti réteget REST alapú kommunikációval terveztem megoldani, ami jelenleg a legjobban elterjedt megoldás web alkalmazások kliens és szerver oldali kommunikációjának megvalósítására.

Az adatrétegben már a tervezés elején eldöntöttem, hogy adatbázisként MySQL-t szeretnék használni, mivel ezzel is volt már tapasztalatom, emellett ez is egy piacon már bevált, jól támogatott relációs adatbázis megoldás. A MySQL remek Spring Boot támogatással rendelkezik, így az architektúrám rétegeinek integrációja sem okoz kihívást.

A kezdeti fázisban azt a döntést hoztam, hogy amíg magát az ötletkezelő alkalmazást fejlesztem, addig elég egy kapcsolatos, egy sémás adatréteg megoldás.



ábra 1 - Kezdeti architektúra

### Multitenancy

A multitenant adatréteg megvalósítását az alkalmazás elkészülte után terveztem, mivel a 3 rétegű architektúrámnak köszönhetően ez gyakorlatilag nincs hatással az alkalmazás többi részére.

Ezt az új adatréteg implementálását egy kutatás előzte meg, ahol megismerkedtem a multitenant architektúrák 3 alfajával, azok működésével, előnyeikkel és hátrányaikkal.

**Tenantonkénti Adatbázis**

Ebben az implementálási modellben minden tenantnak saját adatbázisa van, amihez csak ő férhet hozzá. Ez a legmagasabb szintű elkülönítés, amit multitenant architektúrában el lehet érni. Ez a módszer akkor ajánlott, ha az alkalmazás szigorú adatbiztonságot követel meg.

*Előnyei:*

* + Legmagasabb szintű elszigeteltség; ha egy adatbázis leáll, a többi bérlő nincs hatással.
  + Könnyen skálázható horizontálisan további adatbázisok hozzáadásával.

*Hátrányai:*

* + Magasabb költségek, mivel minden adatbázis külön erőforrásokat igényel.
  + Nehezebben kezelhető nagy skálán, mivel több adatbázist kell felügyelni.

**Tenantonkénti séma**

Ebben az implementálási modellben minden tenant ugyan azt az adatbázist használja, de minden tenantnak külön adatbázis séma van fenntartva. Ez egy közepes szintű elkülönítést eredményez, ami jó egyensúlyt nyújt az adatbiztonság és a költségek között olyan alkalmazások számára, amelyek nem igényelnek teljes adatbázis-szintű elkülönítést.

*Előnyei:*

* + Kezelhetőbb, mint az "Tenantonkénti adatbázis" modell, mivel csak egy adatbázist kell felügyelni. Könnyen skálázható horizontálisan további adatbázisok hozzáadásával.
  + Költséghatékonyabb, mivel a bérlők ugyanazt az adatbázist osztják meg.

*Hátrányai:*

* + Az adatbázis erőforrásai osztoznak a bérlők között, így növekedés esetén fokozódhat a verseny az erőforrásokért.
  + Bonyolultabb lehet a mentés és visszaállítás, mivel a sémák egy adatbázison belül függenek egymástól.

**Megosztott adatbázis, megosztott séma**

Ebben az implementációban minden tenant ugyan azt az adatbázist és sémát használja. A tenantokat egy discriminator oszlop választja el (pl tenant\_id). Ez egy alacsony szintű elkülönítést eredményez, mivel az adatokat csak szűrőkkel védik. Ez a megoldás deális SaaS alkalmazásokhoz, ahol nagyszámú, kisebb bérlő van, és kevés az egyedi igény.

*Előnyei:*

* + A leghatékonyabb erőforráshasználat; alacsonyabb költségek és könnyebb skálázhatóság.
  + Egyszerű adatbázis-kezelés, mivel csak egy sémát kell karbantartani.

*Hátrányai:*

* Nehezebb biztosítani az adatok biztonságát, mivel minden bérlő ugyanazokat a táblákat használja.
* Teljesítményproblémák jelentkezhetnek az adatbázis növekedésével, különösen nagy bérlőszám és komplex lekérdezések esetén.

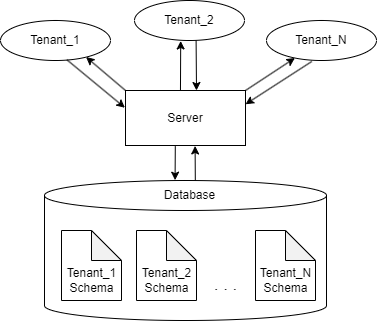
A tervezés során a **Tenantonkénti séma** és a **Megosztott adatbázis, megosztott séma** implementációkban mélyültem el, ezeket teszteltem egy kis projekten, hogy át tudjam majd ültetni a leendő kész alkalmazásba.

Azért esett erre a két módra a választás, mivel az alkalmazás alapvetően nem üzleti titkok és értékes üzleti döntések megvitatására lesz készítve, így nem szükséges a legnagyobb biztonsággal kezelni az adatokat. Másrészről viszont, mivel ez egy off-the-shelf szoftver megoldás, így fontos, hogy sok tenant esetén is megfelelően működjön és költséghatékony legyen.

A **megosztott adatbázis, megosztott séma** implementáció elsőre egy könnyebb megoldásnak tűnt, így ezt próbáltam ki először. A Spring keretrendszer támogatja ezt a multitenancy implementációt, vannak beépített @Filter dekorátorok, amelyekkel meg lehet jelölni Entity szinten a discriminator oszlopot a táblákban. Ennek a megoldásnak az implementációja során azonban abba a problémába ütköztem, hogy felül kell írni az alap Spring által biztosított EntityManagert, ami szimpla lekérdezéseknél nem okoz problémát, de bonyolultabb lekérdezéseknél előfordulhatnak problémák a tranzakciók ütemezésével. Ezekre biztosít a Spring megoldást, de amíg teszteltem nem sikerült hiba mentesre felüldefiniálni az EntityManagert.

Így jutottam a második és a jó megoldásnak bizonyuló **Tenantonkénti séma implementálásához.** Ezt a megoldási módszert is támogatja a Spring Boot. Ennek részletesebb implementálását az Implementáció részben fejtem ki.

Az alább képen látható a végleges architektúra a kicserélt adatréteggel.



ábra 2 - Végleges architektúra

## Modell tervezés

Az alkalmazás funkcionális követelményeinek definiálása után a modelltervezés volt a következő feladat. Ebben a fejezetben erről fogok írni.

A Spring Boot keretrendszer egyik előnye, hogy nagymértékben megkönnyíti az alkalmazás üzleti logikai oldalának és az adatrétegének az összekapcsolását a beépített ORM megoldásával és a Spring Data JPA-val. Ennek egyik hatása az, hogy az osztályok definiálásánál nem feltétlen kell úgy megtervezni az adatkapcsolatokat ahogy azt egy relációs adatbázisnál tennénk, a megfelelő annotációkkal komplex, osztályok közötti tartalmazásokat lehet definiálni, amikből az ORM fog kapcsolótáblákat leképezni. Emiatt a kódunkban élvezhetjük a nem relációs, dokument alapú adatstruktúrákat, de az adatbázisunkba a színfalak mögött mégis létrejönnek a szükséges kapcsolótáblák.

Ezt tudva a modellemet a következőképpen alakítottam ki. Először is összeszedtem, hogy milyen osztályokkal lehet implementálni a fent leírt viselkedést. Ehhez magától értetődő volt a következő osztályok definiálása:

* **User**
* **IdeaBox**
* **Idea**
* **Comment**
* **ScoreSheet**

Ezek azok az osztályok, amik elengedhetetlenek a megoldásban, így ezekkel kezdem a modell tervezés bemutatását.

### A screenshot of a computer Description automatically generatedUser

A User osztály felel azért, hogy a felhasználók adatait kezelje. Az osztály @Entity annotációval van ellátva azért, hogy a Spring JPA könnyen elérje és felhasználja az adatréteggel való kommunikáció során. Emellett az osztály megvalósítja a userDetails osztályt, ami a spring részét képezi és segít a biztonság és fiókkezelés megvalósításában.

A User osztály a képen láthatóan több változóval és listával rendelkezik, ezeket az ORM fogja helyesen relációs adatbázisba illő táblákká alakítani. Az osztályban ezek mellett találhatunk ENUM értéket is, a szerepkör a role változóban van tárolva.

ábra 3 - User osztály

### A screenshot of a computer Description automatically generatedIdeaBox

ábra 4 - IdeaBox osztály

A következő elengedhetetlen osztály az ötletdobozok osztálya. Ez az osztály tárolja és rendszerezi az ötleteke. Egy-egy ötletdoboz több pontozási lapot is tárol, ezek közül az első mindig a Template, aminek segítségével egységesen lehet értékelni a dobozba kerülő ötleteket. A többi eleme a listának a bírálók által beérkezett pontozások. Az egységes pontozást szolgálja az is, hogy minden ötletdoboznak egy előre definiált, kötelező értékelő bizottsága van, ezek a requiredJuries listában vannak tárolva. Ők azok a Juryk, akiknek kötelezően értékelni kell az ötletdobozba érkező ötleteket.

//TODO tobbi

## UI Tervezés

# Implementáció

# Összefoglalás

Irodalomjegyzék

Függelék



ábra 5 Szerepkörökhöz tartozó jogosultságok táblázata