PÉCSI TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

**EGÉSZSÉGÜGYI   
WEBES ALKALMAZÁSFEJLESZTÉS**



Témavezető: Kiss-Vincze Tamás Készítette: Csoboth Enikő (P9Y1RZ)

tanársegéd Programtervező Informatikus

PÉCS, 2022

Tartalomjegyzék

1. Bevezetés
2. Kivonat
3. Témaválasztás motivációja (PCOS)
4. Alkalmazott technológiák
5. PostgreSQL

1.1 Az adatbázisszerkezet ismertetése

1. Java
   1. Mi az MVC?
   2. Springboot keretrendszer bemutatása
   3. Maven
   4. JPA
   5. Rest API
2. Javascript
   1. Vue.js keretrendszer
   2. Node.js
3. Webalkalmazás megvalósítása
4. Feladat specifikáció
5. Tervezés, részfeladatok

2.1 Fontosabb fogalmak

2.2 Folyamatok ismertetése

1. Eltervezett felhasználói felület

3.1 Részegységek ismertetése

1. Megvalósítás

4.1 Osztályok bemutatása

4.2 Programrészek

1. Konfiguráció, üzembe helyezés

5.1 Tesztelés Postman-ben

5.2 Cypress tesztek

1. Felhasználói dokumentáció

6.1 Regisztráció

6.2 Belépés

6.3 Ételek felvétele az adatbázisba

6.4 Étrend generálás

1. Összefoglalás
2. Irodalomjegyzék

**Kivonat**

…

**Bevezetés**

Szakdolgozatom célja, hogy bemutassam egy webes felületű étrendtervező alkalmazás tervezését az kezdetektől, valamint magának a fejlesztésnek a folyamatát és mérföldköveit. Kifogok térni a használt adatbázisra és arra, hogy milyen lépései voltak az adatbázistervezésnek. Ismertetni fogom a használt programnyelveket, mind backend mind frontend oldalról. A megvalósítás folyamatai részben szó lesz arról is, hogy milyen részegységekre osztottam fel a fejlesztést, hogyan terveztem meg egyes részeknek működését, majd kivitelezését és milyen problémákba ütköztem az út során. Külön választottam a tervezett és a megvalósított programrészeket, ugyanis sok problémába ütköztem fejlesztés során, ezáltal nem minden eltervezett dolgot sikerült megvalósítani. A témaválasztásom oka elsősorban az volt, hogy ne felejtsek ki semmilyen makrótápanyagot, illetve vitamint a mindennapos étrendemből. Pajzsmirigy alulműködéssel és PCOS-sel diagnosztizáltak, ennek szintentartásaképpen megadott ételeket ehetek csk, illetve többnyire olyan élelmiszereket amik nincsenek feldolgozva. Mindig problémám volt avval, hogyne felejtsek ki egy vitamint se beszedni, illetve, hogy mindig mindenhez próbáljak meg zöldséget vagy gyümölcsöt fogyasztani. Ekkor született meg az az ötletem, hogy mind ezt egy alkalmazás formájában valósítsam meg, így kiküszöbölve azt a problémát, hogy valamelyik tápanyagot esetleg elfelejtsem.

**PCOS**

Elsősorban azoknak szeretnék ezzel az alkalmazással segíteni, akik ilyen vagy ehhez hasonló betegséggel küzdenek. Mivel ennél főszerepet játszik a mindennapokban az, hogy rendszert tudjon kialakítani az ember maga körül, mind a táplálkozásában, mind pedig az életmódjában.

Ennek gondoltam első lépését megvalósítani, azt hogy legyen egy étrend, amit követhet a felhasználó. A többi lépést később terveztem megvalósítani, ide fog tartozni az alvás, a felkelés utáni testhőmérséklet mérésének feljegyzése, az adott ciklusnaphoz tartozó megfigyelések lejegyzése, az adott naphoz tartozó étvágy feljegyzése, valamint a különböző vitaminoknak a szedését is úgy fogom beütemezni, ahol éppen az adott ciklusban tart a felhasználó. Kiderülhet az alábbi sorokból, hogy elsősorban ez az alkalmazás nőknek készült, legalábbis ők tudják ténylegesen kihasználni, viszont ez egy jó alap lehet a többi betegséghez is, továbbfejleszthető ez a koncepció, akár a diabetikus betegeknek is, vagy bármely olyan betegséghez, ahol fontos rendszert tartani vagy épp figyelhető a napi bevett kalória mennyiség és az ételeknek a minősége. Azon felhasználóik, akik nem szenvednek semmilyen ilyesfajta betegségben, csupán csak szeretnének folyamatosságot az életükbe és kicsit egészségesebben akarnak élni és táplálkozni, azoknak is megfelelő segítséget nyújthat ennek az alkalmazásnak a használata.

**Alkalmazott technológiák**

**PostgreSQL**

A PostrgeSQL egy nyílt forráskódú, objektum-relációs adatbázisrendszer, amely az SQL nyelvet használja. Azért döntöttem ennek a használata mellett, mivel már több projektemnél is alkalmaztam, ezáltal ismerős volt a használata és számomra egyszerű. Minden operációs rendszeren fut ez az adatbázisrendszer, így nem okozott volna gondot, h a fejlesztést más rendszerben folytatom. 2001 óta ACID kompatibilis, vagyis A, mint atomicitás, amely megköveteli, hogy több műveletet oszthatatlan műveletként lehessen végrehajtani, azaz vagy az összes művelet sikeresen végrehajtódik, vagy egyik sem. K, mint konzisztencia, ez biztosítja, hogy z adatok a tranzakció előtti érvényes állapotból ismét egy érvényes állapotba kerüljenek.

I, mint izoláció, amely azt biztosítja, hogy az egy időben zajló tranzakciók olyan állapothoz vezetnek, mint amilyet sorban végrehajtott tranzakciók érnének el, vagyis egy végrehajtás alatt álló tranzakció hatásai nem láthatóak a többi tranzakcióból. D, mint durability, vagyis tartósság, amely azt jelenti, hogy a végrehajtott tranzakciók változtatásait egy tartós adattárolón kell tárolni, hogy a szoftver vagy hardver meghibásodása, áramszünet vagy egyéb hiba esetén is megmaradjon.

Java

A projektben java 11 van használva, ennek oka az volt, hogy szerettem volna egy viszonylag új java verziót használni a későbbi esetleges fejlesztések miatt, hogy ne ütközzek olyan hibába ami a verzióból adódik. A java 8 után a következő stabil és támogatott verzió a 11 volt. Ez jelenleg az utolsó ingyenes hosszútávúan támogatott java verzió a java 10 után. Az Oracle továbbra is kínál OpenJDK kiadásokat, amelyeket ingyenesen letölthetünk és használhatunk. Ezek a buildek ingyenesek, valamint márka nélküliek, ellentétben az OracleJDK-val, amely egy márkás, kereskedelmi konstrukció, 5tehát fejlesztés során ingyenesen használható, viszont fizetni kell érte, hogyha élesben, üzleti célokra szeretnénk felhasználni. A jelentős különbség még a kettő között, amiért talán az OracleJDK-t érdemesebb használni, az az, hogy míg az OpenJDK-ban csak azokra a verziókra érkeznek frissítések, amelyeket éppen most adtak ki, addig az OracleJDK-ban hosszabb támogatást és verziófrissítéseket kaphatunk. Az OpenJDK egy Java platform, ahol legfőképp nyílt forráskódú projektek találhatók, innen tudjuk sok esetben letölteni az alkalmazni kívánt Java JDK-t. A verziók száma napjainkban már a 18-nál tart, viszont még nem jelent meg a hosszabb távú támogatása a verziónak, ez is közrejátszott a döntésemben. A mostani Java verziók kiadása körülbelül fél éves szakaszokra bomlik, régebben, még a 9-es verzió előtti időkben 3-5 évente jött ki egy újabb verzió. Szerencsére a Java visszafelé kompatibilis más nyelvekkel ellentétben, mint például a Python 2-es és 3-as verziója. Itt például egy Java 5-tel írt program gond nélkül le fog futni egy Java 11-es virtuális géppel vagy akár magasabb verzióval. Viszont fordítva nem működik, mert lehet, hogy kifejezetten olyan dolgokat használtunk a projektünkben, amik csak Java 18-cal kompatibilisek, akkor az nem fog működni egy régebbi verzióval rendelkező környezetben. Visszatekintve az alapokhoz tudnunk kell először is, hogy mi a Java JRE és JDK, valamint, hogy mi a különbség a kettő közt?

A JRE (Java Runtime Environment) magát a Java virtuális gépet ás a „java” parancssori eszközt tartalmazza, míg a JDK(Java Development Kit) már eleve tartalmazza a JRE-t , illetve javac fordítóprogramot és még pár eszközt. Régebben le lehetett tölteni külön külön a JREét és a JDK-t is, manapság már egyben lehet ezt megtenni a JDK telepítésével. Ahhoz, hogy java használható is legyen a gépünkön nem lesz elég csak letölteni, be kell raknunk a számítógépünk PATH nevű környezeti változójába.

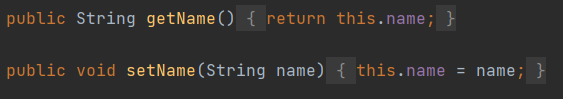
**Folyamatok ismertetése**

Elsőkörben létrehoztam egy alap SpringBoot projektet a Spring initializr segítségével, amihez hozzáadtam a Web, JPA, ThymeLeaf és PostgreSQL dependenciákat. (alábbiakkésőnn is hozzáadhatók a projekthez a pom.xml nevű fájlban). A már korábban ismertetett adatbázis ezután belekonfiguráltam a projektbe, ezt az applicatiohn.properties nevű fájlban tettem meg. Itt meg kellett adni magának az adatbázisnak és táblának az url-jét, a felhasználónevet és a jelszót, nálam ez alapértelmezetten maradt, de későbbieknek, ha élesen kerül felhasználásra az alkalmazás, akkor ezt célszerű megváltoztatni! Létrehoztam ezekután a Model, Repository és Controller rétegekhez tartozó package-eket.majd azután kezdődött el a komolyabb tervezése a folyamatoknak. Több tervet készítettem, melyekből később csak részleteket valósítottam meg. A fejlesztés során mindig előjött egy egy újabb szükséges mező az adatbázisban, így a dolgozatom vége fele is módosítottam a szerkezetén. Először a táblákhoz hoztam létre a különféle modelleket, ezt a legkönnyebben egy pluginnal tudjuk megtenni, melyet az intelliJ-ben könnyen le tudunk tölteni, neve POJOGenerator, ami abban segít, hogy a táblának minden egyes oszlopához készít egy osztálytulajdonságot, sok esetben még a típus is megegyezik, de volt rá példa, hogy javítani kellett benne egy keveset, viszont még így is jóval megkönnyítette a munkát, főleg, hogyha sok oszlopa van az adott adatbázis táblának. Meg vannak adva a learísban azok a változótípusok, amikra fogja átkonvertálni az adott oszlop típusát, például egy varchar típusú oszlopból egy String típusú változót hoz létre.



* 1. ábra – PojoGenerator oszlop átkonvertálás

A plugin használata után, ahhoz hogy adatokkal tudjunk dolgozni létre kell hoznunk gettereket és settereket, amelyeket módosíthatók, illetve lekérdezhetők az adatok.



* 1. ábra- Getter és Setter szerkezetének bemutatása ( Forrás: Saját kód)

Ezekután létrehoztam a a Repository-kat az összes modelhez, amelyben JPA kiterjesztést hazsnáltam, így már sok metódust, például a keresést id-szerint vagy mentést nem kellett külön megírnom, csak azokat a különleges metódusokat, amelyek nem szerepeltek a listában, például az étrend generálásához egy olyan sql parancsot haszáltam, ahol random kiválaszt egy sort az adatbázisból, ehhez csináltam a repositoryban egy random függvényt, melyhez adtam egy Query típusú annotációt, ennek segítségével egy sima sql lekérdezést írhatunk, melyre később egy függvényként hivatkozhatunk. Azért választottam ezt a fajta random oszlop kiválasztó metódust, mert számomra macerásabb lett volna megvalósítani, hogy minden táblánál lekérdezze először a sorok számát, majd abból generáljon egy random számot és aszerint válassza ki a táblából id szerint azt az értéket. Meg is próbáltam csinálni ezt a fajta megoldást, de nem bizonyult gyorsnak a folyamat, majd az internet segítségével ez a megvalósítás mellett döntöttem. A Query annotációnál egy két helyen, ahol kifejezetten native Query-t szeretnénk használni, vagy esetleg nincs JPA Query-s megfelelője a használt metódusnak, ott érdemes berakni még az annotációba, hogy nativeQuery=true, vagyis az egész SQL parancsot konkrét SQL parancsként értelmezi, nálam ez ott merült fel problémaként, amikor a random() függvényt nem ismerte fel a JPa Query-ben és nem találtam rá megoldást sehol, hogyan lehetne megvalósítani, így ezt a natívSQL-es megoldást alkalmaztam.



1.3 ábra- Query annotáció bemutatása (Forrás: saját kód)

Ezekután következtek a Controllerek, ahol a rest API-hoz szükséges endpointokat hoztam létre. Minden model-hez cináltam egy CRUD funkciókból álló controllert, még akkor is ha nem feltétlen terveztem használni az összes funkciót belőle, viszont később akár a továbbfejlesztés során jól jöhet, hogy ezekkel több gond már nem lesz. Minden Controllernél hozzáadtam a @RestController annotációt,

Irodalomjegyzék

<https://www.postgresql.org/about/>

<https://hu.wikipedia.org/wiki/ACID>

<https://en.wikipedia.org/wiki/ACID>

<https://docs.oracle.com/cd/E17276_01/html/programmer_reference/transapp_atomicity.html>

java:

<https://www.baeldung.com/java-11-new-features>

https://www.marcobehler.com/guides/a-guide-to-java-versions-and-features#\_java\_features\_8\_17