Miskolci Egyetem Miskolci Egyetem  
Gépészmérnöki és Informatikai Kar Gépészmérnöki és Informatikai Kar  
Általános Informatikai Intézeti Tanszék Alkalmazott Matematikai Intézeti Tanszék



**Munkaerőközvetítési adatok   
elemzése és hatékony kezelési   
módjainak vizsgálata**

**Szakdolgozat**

**Készítette**:

**Név**: Hornyák Balázs Boldizsár

**Neptunkód**: AD4AYB

**Szak**: Mérnökinformatikus BSc

Korszerű Web technológiák szakirány

|  |  |  |  |
| --- | --- | --- | --- |
| **Miskolci Egyetem**  **Gépészmérnöki és Informatikai Kar** |  | | **Általános Informatikai Intézeti Tanszék**  3515 Miskolc-Egyetemváros |
| Szak**: Mérnökinformatikus** | |  | Szakdolgozat azonosító: **IAL/AD4AYB/BSc/2021** |
| Szakirány: Korszerű WEB-technológiák | |  | **Intézmény azonosító: FI 87515** |
|  | |  |  |

**SZAKDOLGOZAT FELADAT**

**HORNYÁK BALÁZS BOLDIZSÁR**

BSc mérnökinformatikus jelölt részére

|  |  |
| --- | --- |
| A tervezés tárgyköre: | **Webalkalmazás fejlesztés** |
| A feladat címe: | **Munkaerőközvetítési adatok elemzése és hatékony kezelési módjainak vizsgálata** |
| **A feladat részletezése:**   * A megfelelő szakemberek megtalálása nem egyszerű feladat. Ehhez már rendelkezésre állnak különböző adatbázisok és webalkalmazások (például mestertkeresek.hu). * A dolgozat egy olyan webalkalmazás megtervezését, elkészítését és tesztelését mutatja be, amely már összegyűjtött, nagyobb mennyiségű adat hatékony kezelését igyekszik segíteni. * A kezelendő adatok között egyaránt szerepelnek a különféle szakterülethez tartozó mesterek, a hozzájuk tartozó rendelések és az ezekkel kapcsolatos számlázási információk. * A nyilvántartott adatokról az alkalmazás statisztikát is készít, amely naprakészen megtekinthető kell legyen web-böngészőben. * A webalkalmazás elkészítéséhez PostgreSQL adatbázis, Node.js futtatókörnyezet és React JavaScript keretrendszer használata szükséges. | |

|  |  |
| --- | --- |
| Tervezésvezető:  Piller Imre | Tanszék, beosztás:  Alkalmazott Matematikai Intézeti Tanszék, egyetemi tanársegéd |
| Konzulens(ek): | Cég, beosztás: |
| A szakdolgozat kiadásának időpontja: | 2021.09.30. |
| A szakdolgozat beadásának határideje: | 2021.11.12. |
| Miskolc, 2021.09.30. | **Prof. Dr. Kovács László**  tanszékvezető egyetemi tanár |

1. A szakmai gyakorlat helye: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. A szakmai gyakorlat vezetőjének neve: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. A szakdolgozat módosítása: szükséges (a módosítást külön lap tartalmazza)   
 nem szükséges (a megfelelő rész aláhúzandó)

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tervezésvezető aláírása

4. A tervezést ellenőriztem: (1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(3) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(4) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dátum, tervezésvezető aláírása

5. A szakdolgozat beadható

nem adható be

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

konzulens aláírása tervezésvezető aláírása

6. A szakdolgozat ….. szövegoldalt,

….. db rajzot,

….. db CD mellékletet

….. egyéb mellékletet tartalmaz.

7. A szakdolgozat bírálatra: bocsátható

nem bocsátható

A bíráló neve, címe: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tanszékvezető aláírása

8. Osztályzat: a bíráló javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a tanszék javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság döntése: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság elnökének aláírása

**EREDETISÉGI NYILATKOZAT**

Alulírott **Hornyák Balázs Boldizsár**; Neptun-kód: **AD4AYB**

a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős mérnökinformatikus BSc szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy

**Munkaerőközvetítési adatok elemzése és hatékony kezelési módjainak vizsgálata**

című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom

felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

* szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
* tartalmi idézet hivatkozás megjelölése nélkül;
* más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy

plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,.............év ………………..hó ………..nap

…….…………………………………

Hallgató

Tartalomjegyzék

[1. Bevezetés 1](#_Toc88787899)

[2. Koncepció 2](#_Toc88787900)

[2.1. Megoldandó feladatok 2](#_Toc88787901)

[2.2. A probléma megoldására szolgáló módszerek 3](#_Toc88787902)

[2.3. Webalkalmazás használatának előnyei 3](#_Toc88787903)

[3. Követelmény analízis 5](#_Toc88787904)

[3.1. Befizetések kezelése 5](#_Toc88787905)

[3.2. Alap statisztikák megjelenítése 5](#_Toc88787906)

[3.3. Adatok megtekintése és szerkesztése 6](#_Toc88787907)

[3.4. Szakember ellenőrzése 6](#_Toc88787908)

[4. A kliens alkalmazás specifikációja 7](#_Toc88787909)

[4.1. Egyoldalas alkalmazás 7](#_Toc88787910)

[4.2. Funkciók 8](#_Toc88787911)

[4.2.1. Megrendelések 8](#_Toc88787912)

[4.2.2. Alapvető adatok 9](#_Toc88787913)

[4.2.3. Adatok szerkesztése 10](#_Toc88787914)

[5. Technológiák kiválasztása 12](#_Toc88787915)

[5.1. Adatbázis 12](#_Toc88787916)

[5.1.1. PostgreSQL 12](#_Toc88787917)

[5.1.2. Adatbázis bemutatása 13](#_Toc88787918)

[5.2. Frontend technológiák 15](#_Toc88787919)

[5.2.1. Keretrendszerek áttekintése 15](#_Toc88787920)

[5.2.2. React, Angular és a Vue – alap gondolatok 15](#_Toc88787921)

[5.2.3. React, Angular és a Vue funkcióinak összehasonlítása 17](#_Toc88787922)

[5.2.4. React, Angular és a Vue szintaxisának összehasonlítása 18](#_Toc88787923)

[5.2.5. React, Angular és a Vue komplexitásának összehasonlítása 19](#_Toc88787924)

[5.2.6. React, Angular és a Vue – teljesítményének összehasonlítása 19](#_Toc88787925)

[5.2.7. Frontend technológia kiválasztása 21](#_Toc88787926)

[5.3. Backend technológiák 22](#_Toc88787927)

[5.3.1. React és a Node.js (Express) 22](#_Toc88787928)

[5.3.2. Laravel és a Ruby on Rails összehasonlítása 23](#_Toc88787929)

[5.3.3. A backend technológia kiválasztása 25](#_Toc88787930)

[5.4. Webalkalmazás megtervezése 25](#_Toc88787931)

[5.4.1. Backend tervezés 25](#_Toc88787932)

[5.4.2. Frontend tervezés 28](#_Toc88787933)

[5.4.3. Frontend, backend és az adatbázis kapcsolata 30](#_Toc88787934)

[6. Implementáció 32](#_Toc88787935)

[6.1. A fejlesztéshez használt szoftverek 32](#_Toc88787936)

[6.2. Program bemutatása 32](#_Toc88787937)

[6.2.1. Megrendelések 32](#_Toc88787938)

[6.2.2. Alapvető adatok 40](#_Toc88787939)

[6.2.3. Adatok szerkesztése 41](#_Toc88787940)

[7. Tesztelés 43](#_Toc88787941)

[8. Telepítés és futtatás 45](#_Toc88787942)

[9. Összefoglalás 47](#_Toc88787943)

[10. Summary 48](#_Toc88787944)

[Irodalomjegyzék 49](#_Toc88787945)

# Bevezetés

Szakdolgozatomban egy jelentős és összetett probléma megoldását tűztem ki célul, méghozzá egy munkaerőközvetítői weboldal üzemeltetésével járó adminisztratív munkák megkönnyítését, az általam kidolgozott webalkalmazás segítségével.

A mestertkeresek.hu egy már létező honlap, amely szakemberek hirdetéseivel foglalkozik. Hatékony és gyors működéséhez elengedhetetlen a megfelelő adminisztratív háttér biztosítása, ugyanis a beérkező és meglévő adatokat szűrni, kezelni és feldolgozni kell. Ez a folyamat mennyiségénél fogva olyan terhet jelent az ügyintézők számára, amelynek egy része jelentősen csökkenthető az általam létrehozott webes applikáció használatával, amit akár informatikai szaktudással kevésbé rendelkező személy is könnyedén tud használni. A kész webes applikáció felhasználói a mestertkeresek.hu vállalkozás adminisztratív munkáját ellátó munkatársai, tehát egy igen speciális célú webes alkalmazás készítése áll a szakdolgozatom középpontjában. A dolgozat törekszik ennek tervezését, létrehozását és működését teljeskörűen bemutatni. Az applikáció kidolgozása és használata több szempontból is az ügyintézői munkafolyamatok elvégzési idejének jelentős mértékben történő redukálására irányul. Például a kezelendő adatok nagy mennyiségben szerepelnek a különféle szakterülethez tartozó mestereket, a hozzájuk tartozó rendeléseket és az ezekkel kapcsolatos számlázási információk. Az alkalmazásnak ezen adatok ellenőrzését és javítását kell ellátnia, illetve kisebb szintű pénzügyi statisztikát megjelenítenie, mindezt naprakészen, web-böngészőből elérhető formában.

Szakdolgozatomban először a megoldandó problémákat tekintem át, majd ezek megoldásához tervezetet készítek. A kész szoftver bemutatását folyamatokra lebontva, példákon keresztül részletezem.

# Koncepció

Ahhoz, hogy megoldást találjunk, először mélyebben bele kell ásni magunkat a problémába. A már működő, a bevezetésben említett weboldal adatbázisát felhasználva kell az információkat megjeleníteni és szerkeszthetővé tenni az adminisztrációs munkát végző felhasználó számára. De miért is van erre szükség? Nagyon sok cég a mai napig Excel táblázatokban vagy egyszerű *CSV* fájlokban tárolja az adatait. Véleményem szerint ez azért rossz, mert azok gyakran átláthatatlanok és sérülékenyek. A mestertkeresek.hu viszont komplex adatbázisban tárolja adatait, és ezt felhasználva kell a rendszert felépíteni úgy, hogy azok integritása megmaradjon a relációs adatbázisban a szerkesztés után is. Az adatbázis műveletek használata programozói munkát és sok időt igényel ehhez, az egyébként egyszerűnek tűnő feladathoz. Erre fókuszálva kell alkotni egy adminisztrációs felületet, amelyet használva egy, az informatikában nem jártas személy is tud kezelni. Így tehát egy könnyedén áttekinthető felületre van szükség, amelynek használatát pár perc alatt el tudja sajátítani a felhasználó. Ennek következtében a pénzügyi, marketing és ügyintézői munkavégzés kiadható nem fejlesztők kezébe is.

## Megoldandó feladatok

A szoftver feladata, hogy a mestertkeresek.hu adatbázisából kinyerve megkapjuk az ügyintézéshez szükséges információkat és mindezeket módosítani is tudjuk. Az egyik ilyen megoldandó probléma a pénzbefizetések adminisztrálása. A munkaerőközvetítői weboldalon egy hirdetés feladásáért a felhasználónak fizetnie kell. Az ilyen tranzakciók ellenőrzést igényelnek, különösképp egy banki átutalás, hiszen az akár több napot is igénybe vehet, hogy beérkezzen az összeg a felhasználótól, és az a vállalkozás számlájára jusson. Ezeknél nagy segítség egy olyan alkalmazás, amely nyilvántartja ezeket, és segítséget nyújt az adminisztrálásban azzal, hogy át tud írni adatokat, egyszerre több táblában, akár egyetlen gombnyomásra.

Adatbázis táblákból kiolvasva elég nehéz megtudni, hogy éppen hogyan is áll a vállalkozás. A program feladata többek között, hogy kisebb statisztikát készítsen a bevételről és vásárlásokról. Mindezeket, akár diagramon, akár egyszerű kimutatásként ciklusokra felosztottan meg kell tudnia jeleníteni. Ezekkel az információkkal naprakészen lehet látni a cég állását, amellyel egyszerűbben lehet számolni akár azt, hogy szükség van e intenzívebb marketingre, vagy másféle változtatásokra, hogy a kívánt bevételt termelje az oldal.

Regisztráció során sok különféle adat kerül eltárolásra az adatbázisban egy felhasználóról. Az ezekhez való hozzáférést el kell látni a rendszernek. Gyakran szükséges lehet megtekinteni, vagy adott esetben változtatni is egy ügyfél adatain. Előfordulhat a hirdetéseket feladó emberekkel, hogy hibásan írják le a megjelenítésre szánt szöveget anélkül, hogy ezt észrevennék. Az így ejtett magyar nyelvtani hibák átírását, javítását is egyszerűen el kell tudni végezni, ezzel is növelve a szakemberkereső oldal színvonalát.

## A probléma megoldására szolgáló módszerek

Egy adatbázis kezelésére léteznek adatbáziskezelő szoftverek, melyben át lehet tekinteni az egyes táblák adatait, vagy akár szerkeszteni is lehet ezeket. Ilyenek például a *RazorSQL* [1], *TablePlus* [2], *pgAdmin* [3] és még sok más hasonló szoftver. Azonban ezekben elég nehéz lehet egy hatalmas méretű, komplex adatbázis kezelni, még akkor is, ha saját SQL lekérdezéseket írunk. Esetenként nem csak a problémákra való megoldás tűnhet időigényesebbnek és bonyolultabbnak, de maga az adminisztrátori munkavégzés is. A fent említett problémákra egy saját webalkalmazás fejlesztése a legmegfelelőbb megoldás, amellyel egy webes böngészőben szinte bárhonnan el lehet végezni a szükséges munkát, így nincs szükség semmiféle adatbázis kezelő szoftver ismeretére, továbbá az ügyintézési folyamat is gördülékenyebbé válik.

## Webalkalmazás használatának előnyei

Amikor egy adatbázis kezelő szoftvert úgymond *bekonfiguráltunk*, illetve az adott munkához szükséges SQL lekérdezések megírtuk, azután ha ezeken változtatni szeretnénk, vagy megsérülnek, akkor azt hatalmas munkával jár helyreállítani. Nem is beszélve arról, hogy ezek a szoftverek egy adott számítógépre vannak telepítve, amellyel hatalmas veszélynek tesszük ki az adatbázist és ezzel együtt a vállalkozást is a vírus és egyéb támadások szempontjából. Például, ha az alkalmazott számítógépe meghibásodik, és nem tud dolgozni az adatbázissal, akkor ez idő alatt nem fogja tudni ellátni az adminisztrátori feladatát.

Mind ez elkerülhető egy olyan webalkalmazás használatával, amely egy felhőben futó szerverrel oldja meg ezeket a funkciókat, és biztonságosan használható bármely internetre csatlakozó számítógépről.

A grafikus felület kialakítása hatékonyabb lehet azáltal, hogy csak azok az információk jelennek meg, amelyek relevánsak az épp elvégzendő feladathoz. A vizuális megjelenítést is egyszerűbbé teszi, szinte bármilyen diagramot vagy egyéb kimutatást létre lehet hozni. Mindemellett előnye még, hogy könnyebb a karbantartása, és a szükség szerinti fejlesztése a webalkalmazásnak, mivel a kliensek verzióinak frissítése természetes módon adódik.

# Követelmény analízis

## Befizetések kezelése

A mesterkeresek.hu-n havi előfizetéses rendszer működik, amelyek három, hat és tizenkét havi megjelenést biztosít egy mesterember számára, aki ezáltal hirdetheti magát az oldalon. Ezeket az előfizetéseket banki átutaláson keresztül vagy online bankkártyás fizetéssel tehetik meg a felhasználók.

A bankkártyás fizetés Barion rendszeren keresztül történik, amely egy elektronikus fizetéseket lebonyolító szolgáltatás [4]. Ha ezt a fizetési módot választotta a felhasználó, és sikeres volt a tranzakció, akkor azzal automatikusan megtörténik az előfizetés aktiválása. Azonban, ha a banki átutalást választja, akkor azt egy adminisztrátornak ellenőriznie kell. Amíg nem érkezik be az előfizetés helyes összege, addig az oldalon nem jelenik meg a felhasználó hirdetése. Többek között ezt az ellenőrző és aktiváló folyamatot hivatott segíteni a dolgozatban bemutatásra kerülő webes alkalmazás.

## Alap statisztikák megjelenítése

Az alkalmazásban szerepelnie kell a bevétel kisebb statisztikájának megjelenítése. Az eladások darabszámának és a bevétel forint alapú kiírásának, illetve ezeknek ciklusokra bontásának elérhető funkcióknak kell lenniük. Mindezeknek az előző periódushoz viszonyított százalékos megjelenítését is képes legyen kiszámolni. Például, ha előző héten 4 darab eladás volt, ezen a héten pedig 6, akkor az +50%-os többleteladást ismertet. Továbbá az összes regisztrált mester, és a függőben lévő rendelések aktuális darabszámának megjelenítése is fontos, hogy naprakész információkat kapjunk a vállalkozásról. Így sokkal egyszerűbb meghatározni a cég aktuális pénzügyi egyenlegét. Ez akkor lehet hasznos, ha látjuk az előző ciklushoz képest, hogy több vagy kevesebb bevételt termelt az oldal. Például kevesebb keresletnél át kellhet gondolni a marketing stratégiát.

Az alkalmazásban szerepelnie kell az eladások szemléltetésére egy havi lebontású diagramnak. Ez az alapvető statisztikai illusztráció a csomagok eladásának darabszámú megoszlását kell tudnia reprezentálni.

## Adatok megtekintése és szerkesztése

Az adatbázis egyes tábláit – melyet részletesen az 5. fejezetben mutatok be –megjeleníthetővé és szerkeszthetővé kell tenni. Ilyenek például a szakemberek, felhasználók és munkák tábla, melyek sorait elsősorban pár lényegi információval szükséges megmutatni, melyből beazonosítható az adott rekord.

Ezek szerkesztésére és a további információinak megjelenítésére egy adott gombra kattintva egy új felületnek kell megnyílnia. Ennek a felületnek űrlapszerű módon kell megjelennie, ahol a különböző adattípusoknak megfelelő kitöltési módnak szükséges szerepelnie. Például dátum esetében dátumválasztó (date picker), logikai változónál pedig kapcsoló (switch button).

Továbbá indokolt a *CSV* exportálás is a kijelölt rekordokra, amely az adatok egyszerű továbbításában és csoportosításában tud segíteni.

## Szakember ellenőrzése

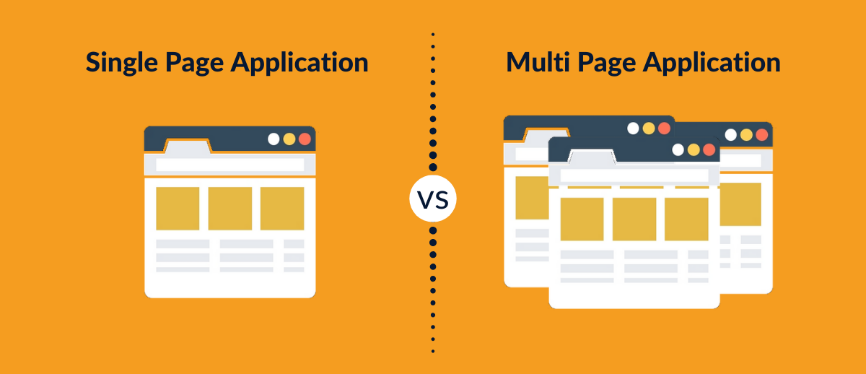
A mesterkeresek.hu egyik nagy előnye, hogy a szakember ellenőrzésre kerül, amikor a hirdetése megjelenik az oldalon. Ennek kivitelezése a regisztrációnál megadott adószám felhasználásával történik. A hiteles nyilvántartó rendszerekből ellenőrizhető, hogy a hirdetést feladó szakember adatai a valóságnak megfelelnek-e, valamint a tényleges tevékenységi köre megegyezik-e az általa megadott információkkal. Ezen eljárás a fogyasztók védelmére irányul, növelve ezáltal a szolgáltatás minőségét.

A program célja többek között a vizsgálatot végző személy munkájának könnyebbé tétele azáltal, hogy – a valós eredmény esetében – egy kattintásra megváltozik a felhasználói webes felületen a szakember hirdetésének státusza („Nem ellenőrzött mester”-ről „Ellenőrzött mester”-re).

# A kliens alkalmazás specifikációja

## Egyoldalas alkalmazás

Az egyoldalas alkalmazások – angolul: Single Page Application (SPA) – dinamikusan töltik be a weboldalak tartalmának egy részét. Ez azt jelenti például, hogy az alkalmazásnak nem kell egyszerre betöltenie az egész oldalt, csak azokat az elemeket, amelyekre éppen szükség van. Az egyik legnagyobb előnye az SPA webalkalmazásoknak, hogy a felhasználó – a projektem esetén egy ügyintéző – ugyanazon az oldalon marad, és nem kell több webböngésző ablak között navigálnia, ezáltal gördülékenyebbé téve a felhasználást. Ennek szemléltetésére az 1. ábra szolgál. Szakdolgozatomban egy ilyen egyoldalas alkalmazás megvalósítása a célom, amelyet bárki tud kezelni, ezzel lerövidítve az adminisztratív munkák idejét.



1. ábra Single Page Application (SPA) [5]

Az SPA [5] további előnyei, hogy

* a webes alkalmazások az okostelefonokon is egyszerűbben kezelhetők,
* lehetővé teszi, a frontend és a backend rész teljes elkülönítését,
* gyorsítótárazással és offline támogatással is rendelkezik,
* a webböngészők beépített fejlesztői eszközével végzett hibakeresés sokkal gyorsabb és egyszerűbb az egyoldalas alkalmazásoknál.

## Funkciók

A következőkben sorra veszem, hogy melyek azok a műveletek, amelyekre képesnek kell lennie az alkalmazásnak, illetve a különböző esetekre hogyan kell reagálnia. Ezek megvalósítását a 6. fejezetben fogom részletesen bemutatni.

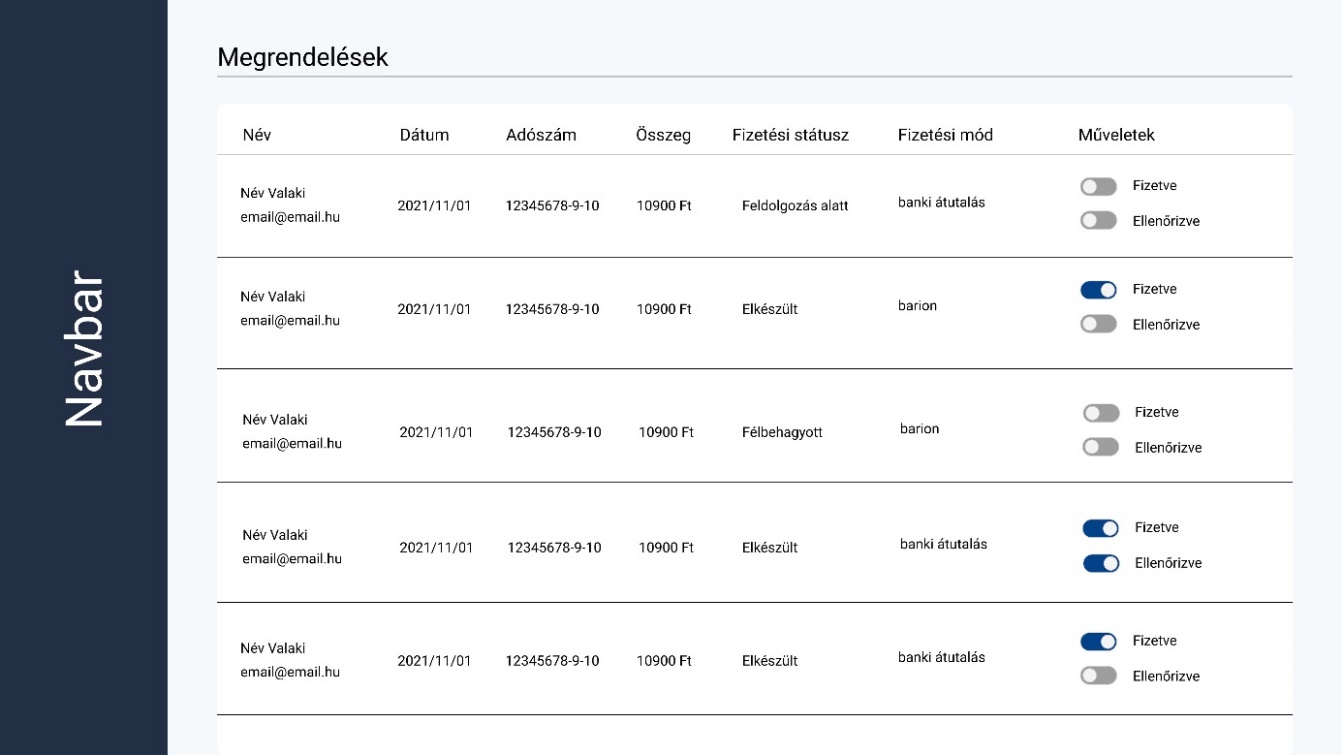
### Megrendelések

A megrendelések esetében azt a kérdést kell megválaszolni, hogy milyen használati esetek fordulhatnak elő, illetve hogyan kell működnie mindennek? Táblázatszerűen meg kell jeleníteni a felhasználó egyes adatait, melyek a következők: név, email cím, adószám. A megrendelés adatait is ezen a felületen kell látni egyben a vásárló adataival, amelyek nem mások, mint a rendelésszám, keltezés, a befizetés összege, státusza és módja (online bankkártyás vagy banki átutalásos). A felsorolt adatoknak a táblázat egy sorát kell képezniük, melyek mellett helyet kell kapnia a fizetési állapotot és ellenőrzöttségi státuszt átállító gombnak. Az utóbbi egy manuális folyamat, melyet az alkalmazást használó ügyintézőnek kell elvégezni az erre alkalmas nyilvántartó rendszerek segítségével.

A fizetési státusz állításának lehetőségét csak banki átutalásnál kell lehetségessé tenni, ugyanis a bankkártyás tranzakciónál egyértelmű információk vannak a fizetés sikerességéről. A fizetési állapot jelzői a következők lehetnek: elkészült, félbehagyott, feldolgozás alatt. Az elkészült státusz vagy egy sikeres Barionos (bankkártyás) fizetés után lehetséges, vagy egy olyan banki átutalás, amelynek összege sikeresen beérkezett és, azt az ügyintéző már jóváhagyta. A feldolgozás alatti állapotjelző banki átutalásnál lehetséges, ezek olyan megrendelések, melyek pénzösszege nem érkezett még be, vagy az adminisztrátor még nem hagyta jóvá. Ezt az ellenőrző folyamatot a fizetési státusz módosítására szolgáló gomb megnyomásával kell tudnia a programnak lezárni, ezzel elkészült státuszba léptetni a megrendelést. Félbehagyott státusz Barionos fizetésnél lehetséges, amikor a felhasználó vagy sikertelen tranzakcióval zárta le a folyamatot, vagy pedig elnavigált az oldalról. Ennek megjelenítése marketing szempontból fontos, ugyanis ha az ügyintéző látja, hogy adott megrendelésre sikertelen volt a fizetés, akkor felkeresheti emailben a felhasználót segítségnyújtás céljából.

A fent leírtakra a 2. ábra ad egy mintát, amelyen az alkalmazás egy képernyőterve (wireframe) látható.

A wireframe [6] (magyarul drótváz) az elkészíteni kívánt megjelenítés előzetes vázlata, amely tartalmaz minden olyan fontos elemet, amelynek a kész alkalmazásban szerepelnie kell. Jobban megérthető segítségével a koncepció, azáltal, hogy vizuálisan is könnyebben elképzelhető.

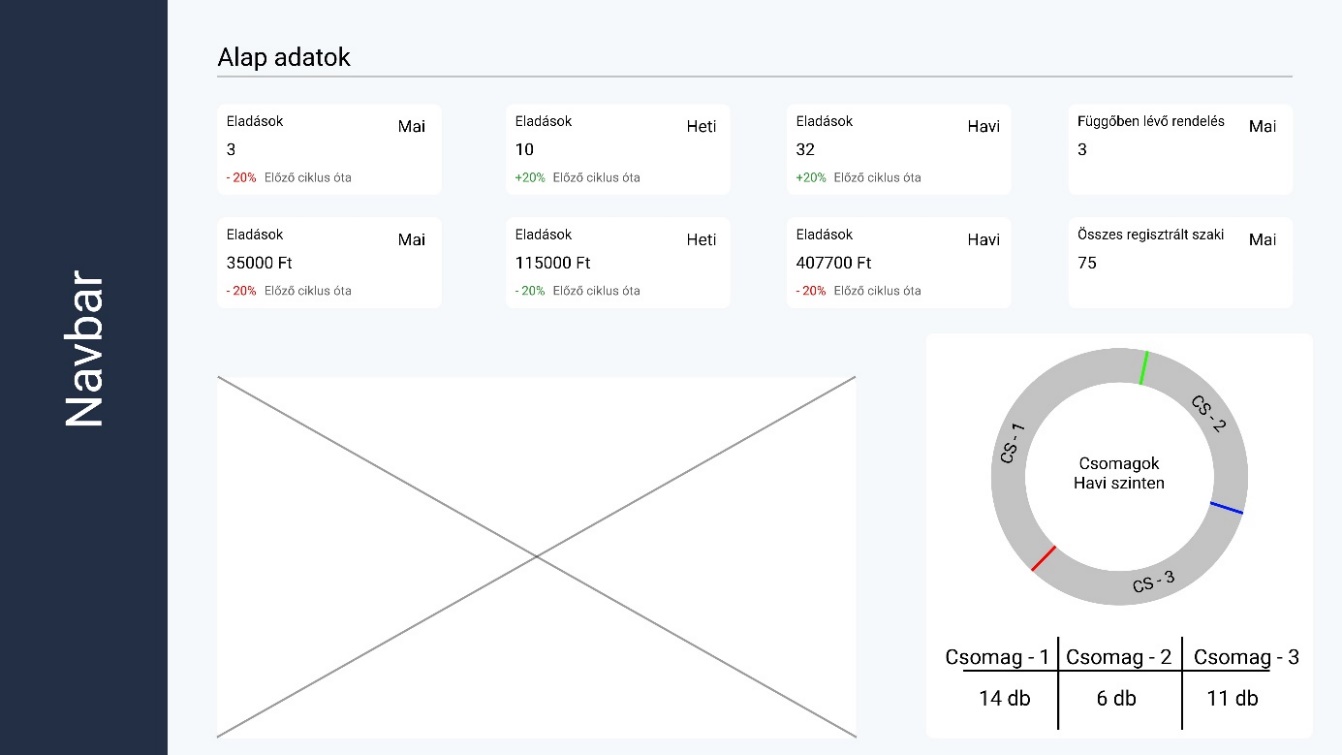


2. ábra Megrendelések képernyőterv

Ennek az egész táblázatos kialakításnak oldalanként lapozhatónak kell lennie, és ki kell tudni jelölni azt, hogy hány elemet szeretnénk megjeleníteni az adott oldalon. Mindezt különösen nagy várakozási idő nélkül kell tudnia teljesíteni az alkalmazásnak a gördülékeny munkavégzés érdekében.

### Alapvető adatok

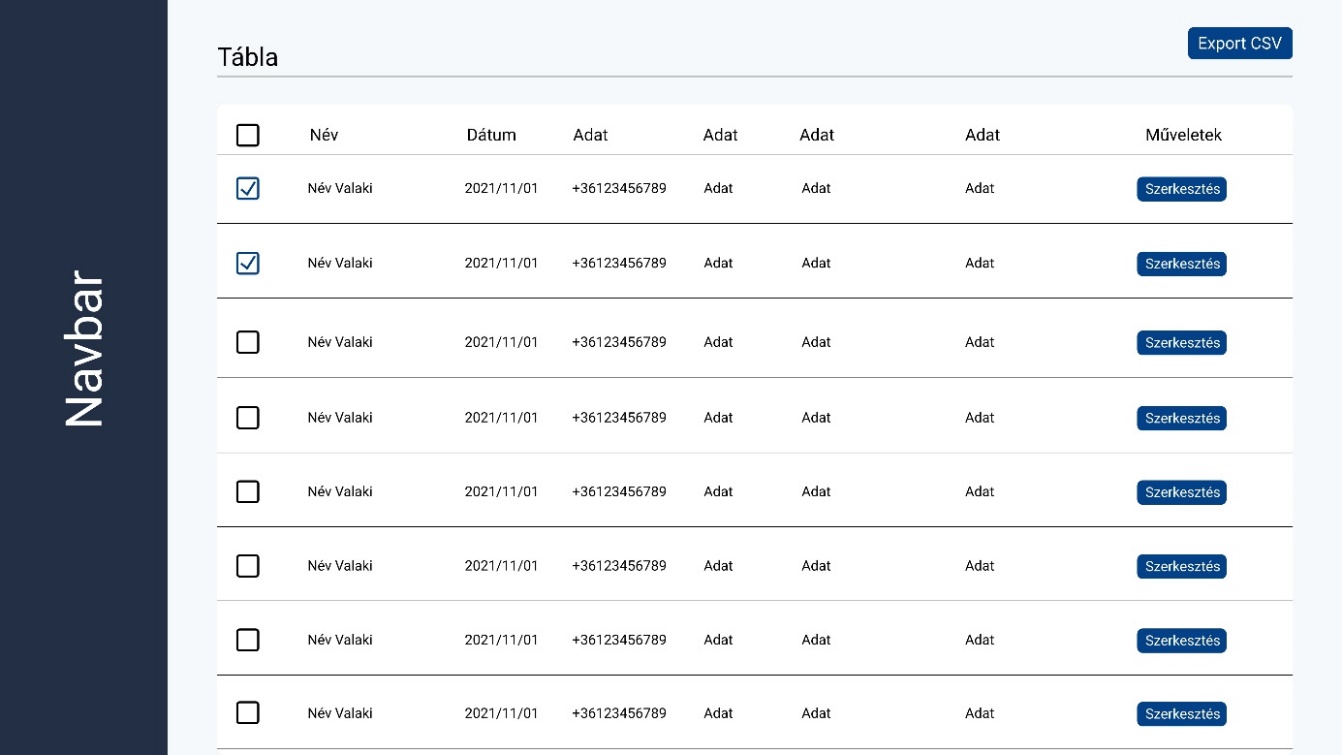
A 3.2. fejezetben leírtak alapján ezen az oldalon a vállalkozás naprakész statisztikáját kell tudni átláthatóan megjeleníteni. A bevétel illetve az eladások számát napi, heti és havi lebontásban szükséges kimutatni. Ezeket az előző ciklushoz mérten a rendszernek ki kell tudni számolni a többletet vagy épp a csökkenést. Ezeket mindig az aktuális naptól számítva kell nézni, hiszen például ha a havi lebontást mindig a hónap első napjától számolnánk, akkor reális értéket csak a hónap utolsó napján kapnánk. Mindez igaz a csomag-eladások megoszlásának kimutatására szolgáló diagram esetében is. Ezek szemléltetésére a 3. ábrán található képernyőtervet készítettem el.



3. ábra Alap adatok képernyőterv

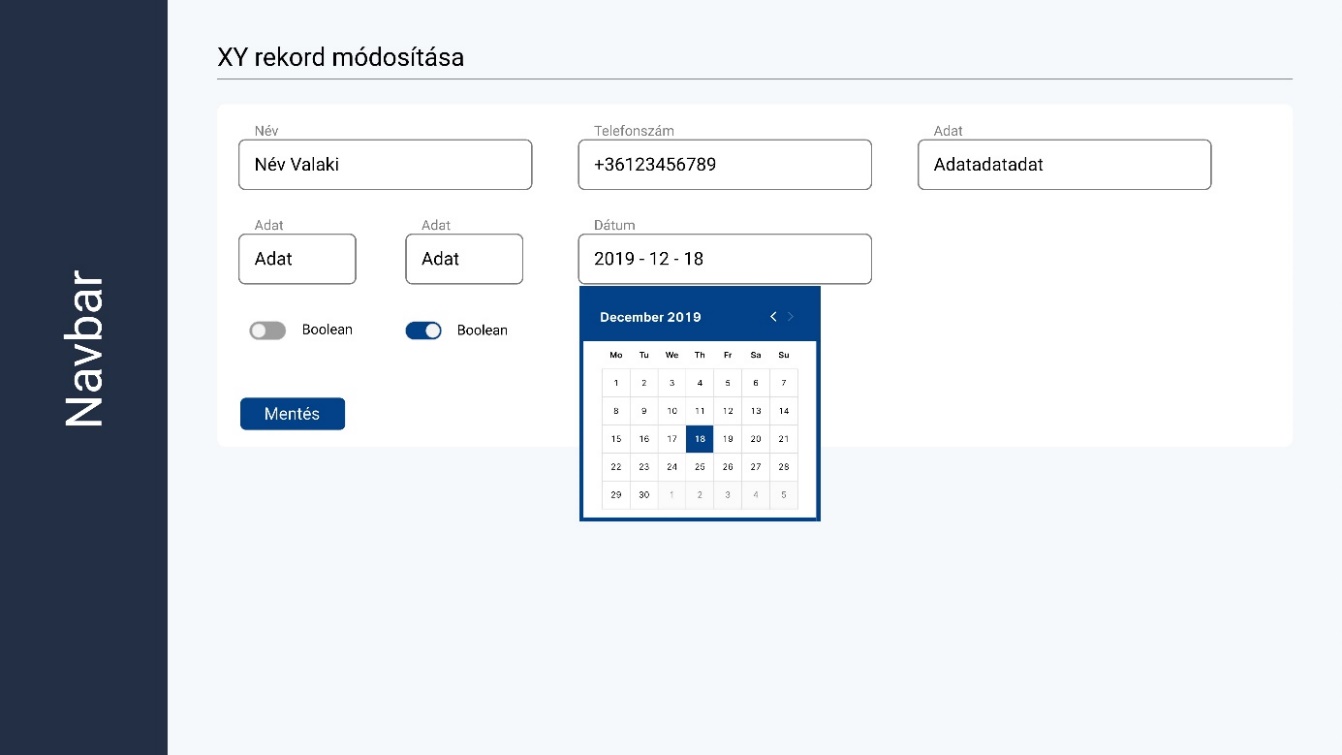
### Adatok szerkesztése

Ebben a funkcióban az adatbázis legfontosabb tábláit kell megjeleníteni és módosíthatóvá tenni. Ezeket egy rendezhető táblázatban és – mint a megrendelések funkciónál – itt is testreszabhatónak kell lennie az adott oldalon megjelenő sorok számának. Egy adott sorban csak pár lényegi információnak kell szerepelnie az átláthatóság érdekében. A 3.3. szakaszban leírtak alapján az alkalmazásban szerepelnie kell CSV exportálás funkciónak is. Ehhez szükség van minden sorban egy jelölőnégyzet elhelyezésére, melynek segítségével összegyűjthetjük azokat a rekordokat, melyeket az exportálás gombra kattintva azonnal letölthetünk. Ezek tervezett megjelenítését a 4. ábrán található wireframe mutatja be.



4. ábra Egy tábla megjelenítésének képernyőterve

Az egyes rekordok szerkesztéséhez az adott soron elhelyezett "Szerkesztés" feliratú gomb segítségével kell tudni eljutni. Itt minden szerkeszthető információt fel kell tüntetni, és az adott adattípushoz megfelelő megjelenítési és változtatási módot kell használni. Ezek illusztrálására az 5. ábra szolgál.



5. ábra Rekord szerkesztése képernyőterv

# Technológiák kiválasztása

A webalkalmazás három fő részre bontható: frontend, backend és az adatbázis. Ezek létrehozására több technológia létezik. A következőkben összehasonlítom ezeket és megpróbálom a legmegfelelőbbet kiválasztani, majd megtervezni ezek alapvető funkcióit és működési mechanizmusát.

## Adatbázis

A programom közös adatbázist használ a mestertkeresek.hu-val, amely teljes mértékben kivitelezhető, tulajdonképpen ez egy normál működési mód. Az adatbázis szempontjából nem lényeges, hogy ki az „ügyfél”. Ez lehet egy webalkalmazásból származó kapcsolat vagy egy vastag kliens, illetve egy szerver alapú folyamat. Az adatbázis számára ezek mindegyike csak egy másik felhasználó, és amint a kliens által kért adatbázis-feladat befejeződött, az adatbázis türelmesen vár egy másik feladatra.

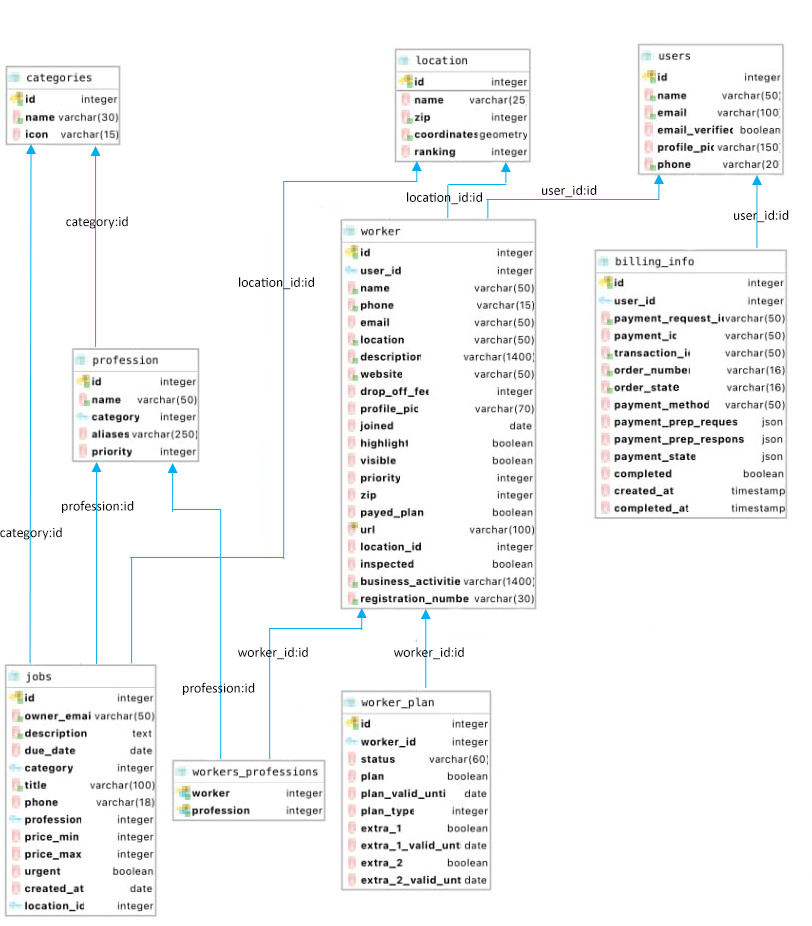
Az általam kidolgozott programban a már meglévő adatbázisból dolgozva kell az adatokat megjeleníthetővé, szerkeszthetővé és áltáthatóvá tennem adminisztrációs feladatok elvégzésének céljából. Ez az adatbázis *PostgreSQL* alapú, amely mára az egyik legnépszerűbb nyílt forráskódú adatbázis lett [7].

### PostgreSQL

A *PostgreSQL* jól működik minden modern webes keretrendszerrel. Támogatja az SQL-t és a JSON relációkat [8], továbbá a fejlett adattípusokat és a teljesítményoptimalizálási szolgáltatásokat, amelyek jellemzően csak drága kereskedelmi adatbázisokban érhetők el, mint például az *Oracle* [9] és a *Microsoft SQL Server* [10]. Tapasztalt fejlesztői közösség támogatja, akik óriási hozzájárulást tettek ahhoz, hogy rendkívül megbízható DBMS-rendszerré váljon.

### Adatbázis bemutatása

A mestertkeresek.hu adatbázisa meglehetősen nagy és szerteágazó, azonban az alkalmazásom ezekből csak pár fontosabb táblát használ. Ezekre fókuszálva próbálom bemutatni az adatbázis-táblákat és kapcsolatokat, amelyet a 6. ábra szemléltet.



6. ábra Adatbázis relációs sémája

A fő tábla a felhasználók (*users* tábla), amelyhez kapcsolódnak a mesteremberek (*worker* nevű tábla), illetve a számlázási információk (*billing\_info* tábla). Ezek a *user\_id* azonosítóval kapcsolódnak össze. A *billing\_info*, *payment\_prep\_response* és *payment\_state* oszlopa olyan JSON típusú adatokat tárol, amelyet a Barion online fizetési oldal ad vissza, így ezek csak az online bankkártyás fizetési módnál jönnek létre. A banki átutalásnál ezek NULL értéket vesznek fel és csak a *payment\_prep\_request* szintén JSON formátumban tárolt adatai érhetőek el. A *completed* boolean típusú mezőben tárolódik el, hogy sikeres volt-e a fizetés. A banki átutalás helyes összegének sikeres beérkezése után ennek a mezőnek az értékét állítja át az ügyintéző TRUE-ra, viszont sikeres *Barionos* fizetés esetén ez már automatikusan TRUE értéket vesz fel.

A *worker* tábla a *location* táblához kapcsolódik, ahol Magyarország települései vannak összegyűjtve nevükkel, irányítószámukkal és koordinátáikkal együtt, így az adott mestert hozzá lehet rendelni egy adott településhez, amely a *location* mezővel való összekapcsolással kerül eltárolásra. Továbbá számos más adat kerül rögzítésre ebben a táblában, azonban ezek mezőneveiből egyértelműen kikövetkeztethetők az azokban tárolt értékek. Két további attribútumot szeretnék kiemelni, melyek az *inspected* és a *visible*. Utóbbi egy boolean típusú mező, amely TRUE értéket vesz fel, ha a mesterember hirdetése megjelenik az oldalon. Ez szintén automatikusan történik, – csak úgy, mint a *billing\_info* tábla *completed* mezőjénél – sikeres online bankkártyás fizetésnél, azonban banki átutalás sikeres teljesítésénél is átállításra kerül a programom segítségével. Az *inspected* szintén logikai adattípusú mező, amely a 3.4. szakaszban leírtak indikátoraként szolgál. A még nem ellenőrzött mesterek „hamis”, míg a már egy ügyintéző által megvizsgáltak „igaz” értékre való átállítását lehet megvalósítani az alkalmazásomban.

A *worker* táblához többek között a *worker\_plan* tábla kapcsolódik, ahol maga a megvásárolt előfizetés részletei láthatóak, mint például a megvásárolt csomag aktivitásának határideje, vásárolt-e a felhasználó további extra funkciókat, úgy mint, a közösségi oldalakon való hirdetés vagy első helyre sorolás a keresési eredményekben. A másik hozzá kapcsolódó, a *profession* tábla, amely egy kapcsolótábla segítségével működik. Itt a mestertkeresek.hu-n összegyűjtött szakmák találhatóak meg, amelyek további kategóriákba vannak besorolva és ezek a *categories* táblában kaptak helyet. A *jobs* táblában találhatóak azok a hirdetések, amelyeket egy felhasználó hozott létre, aki egy elvégzendő munkára várja a szakemberek jelentkezéseit. Az adatbázisban ez a korábban említett *categories* és *profession* táblához egyaránt csatlakozik, hogy könnyen azonosítható legyen az adott munkakör. A webalkalmazásomban ezeket is át lehet tekinteni és adott esetben szerkeszteni rajtuk.

## Frontend technológiák

Először azt érdemes megvizsgálni, hogy van-e szükség valamilyen keretrendszer használatára az adott feladat megoldásához [11]. Ha egy kisebb alkalmazáson dolgozunk elég lehet natív *JavaScript*-tet vagy *jQuery*-t használni. Ezt csak akkor érdemes választani, ha biztosak vagyunk benne, hogy nem lesz komplexebb a projekt. Egy keretrendszer több száz MB-nyival is megnövelheti az applikáció méretét, ami negatívan hathat ki a betöltési időre. Azonban, ha megoldandó feladat összetett, és van rá esély, hogy további funkciókkal is fog bővülni, akkor érdemes elgondolkodni egy keretrendszer vagy könyvtár használatán.

### Keretrendszerek áttekintése

A keretrendszerek sok mindenben eltérnek a hagyományos JavaScript-től, viszont merőben megkönnyítheti a munkát. Vannak dolgok amiket könnyebb kivitelezni velük, vagy csak egyszerűen rövidebb idő alatt megoldhatók bizonyos problémák. A frontend keretrendszerek előnyei például, hogy

komponensekből állnak, amiket újból fel lehet használni,

áttekinthetőbb és strukturáltabb kódot adnak,

időt spórolnak meg számunkra, a készen elérhető funkcióknak köszönhetően,

segítik az SPA felépítésű alkalmazások írását.

Frontend framework esetén a *React*, *Angular* és a *Vue.js* (általában csak *Vue* néven használják) piacvezetők, így a következő részben ezeket hasonlítom össze.

### React, Angular és a Vue – alap gondolatok

Először is fontos megjegyezni, hogy nincs olyan, hogy legjobb keretrendszer vagy könyvtár [12]. Mindhárom könyvtár jó okokból nagyon népszerű. Mindegyiküknek megvannak az erősségei és gyengeségei, azonban általában bármelyik könyvtárat fel lehet használni bármely projekthez.

Az *Angular* a Google által kifejlesztett keretrendszer ami 2010-ben jelent meg. A Google belsőleg is használja az *Angular*-t, ezért elképzelhetetlennek tartom, hogy ez a keretrendszer eltűnik egyik napról a másikra, hiszen fenntartják és folyamatosan fejlesztik. [13]

A *React.js*-t eredetileg a *Facebook* fejlesztette, ebből adódóan a Facebook is azt használja, valamint az Instagram. 2013-ban jelent meg, tehát később, mint az *Angular*. [14]

A *Vue* egy önálló projekt, amely fejlesztése mögött egyetlen vállalat sem áll. Korábban egy személy fejlesztette (Evan You), azonban mára már elkötelezett munkatársaival együtt dolgoznak a *Vue* fenntartásán és továbbfejlesztésén. [15]

A három közül az *Angular* határozottan a „legnagyobb” keretrendszer. Az *Angular* sok mindenre kiterjed. Segít a fejlesztőnek a felhasználói felület (User Interface) vezérlésében, a felhasználói bevitelre való reagálásban, a felhasználói input form-ban történő validálásában, útválasztásban (*routing*), az *AJAX* HTTP kérések küldésében, offline támogatás és PWA képességek biztosításában, tesztelésben, alkalmazás felépítésében, több alkalmazás kezelésében és még számos más folyamatban is. Ezen kívül létezik egy hivatalos CLI (Command-Line Interface), amely segítséget nyújt az *Angular* projektek létrehozásában és kezelésében, naprakészen tartásával, függőségek hozzáadásával és telepítésével.

A *React* jellemzőiben és irányadó vonásaiban nagy mértékben különbözik a korábban röviden ismertetett *Angular*-tól. A *React* csak egyetlen könyvtárat ad, amely tartalmat jelenít meg a DOM (Document Object Model) számára és azt hatékonyan ellenőrzi. Továbbá megadja az összes szükséges eszközt annak meghatározásához, hogy milyen körülmények között, milyen módon kell megjeleníteni az információt a képernyőn. Azonban nem tartalmaz többek között beépített űrlap-ellenőrzési támogatást, útválasztót (*router*) és nem szállítja a saját HTTP-kliensét. Ezen kívül nem tartozik hozzá más különleges funkció így ebben a tekintetben határozottan „kisebb”, mint az *Angular*. Ezeknél a funkcióknál a *React* közösségre kell támaszkodni.

A *Vue* egy olyan keretrendszert, amely a *React* és az *Angular* között helyezkedik el. Nem olyan „nagy”, mint az *Angular*, de határozottan több funkciót tartalmaz, mint a *React*. A *Vue* többek között beépített állapotkezelést is biztosít, valamint beépített útválasztóval (*router*) rendelkezik. Nem tartalmazza azonban az űrlap-hitelesítést vagy a HTTP-kliens funkcióit. Úgy, mint az *Angular* és a *React* esetében, a *Vue* lényegi célja a felhasználói felületek felépítése az újrafelhasználható komponensek kombinálásával.

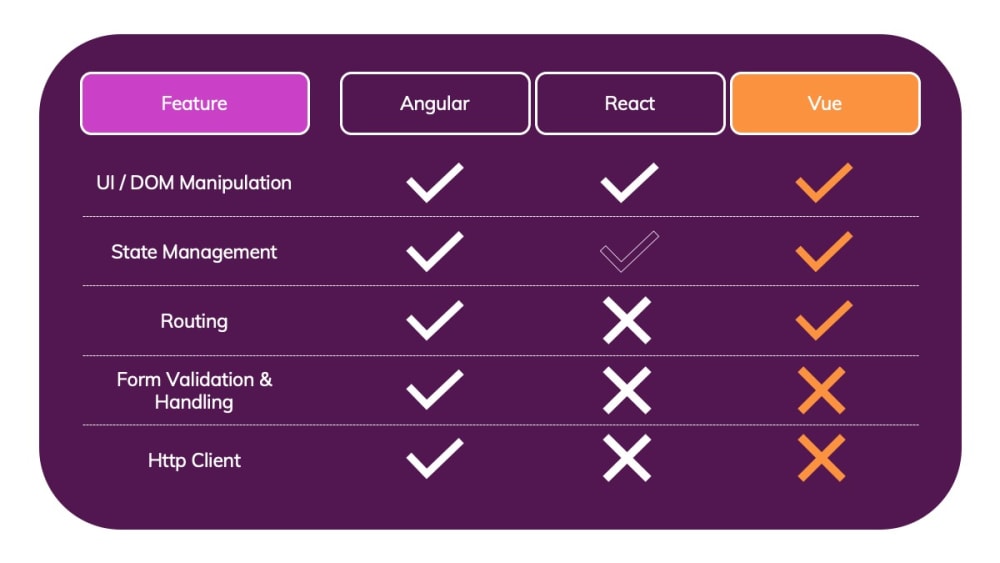
### React, Angular és a Vue funkcióinak összehasonlítása

Az általam elemzett keretrendszerek közül az *Angular* kínálja a legtöbb funkciót, amely a fejlesztők számára is több lehetőséget jelent. Azonban ez nem jelenti egyértelműen azt, hogy jobb lenne a többitől, hiszen számos más tényező – projekt, személyes preferencia – is befolyásolhatja azt.

A *React* egyszerűsége, valamint a komponensekre és a kezelőfelület-megjelenítésre való nagy hangsúlya jelenti az erősségét. Ahol az *Angularnak* sok mindent össze kell kapcsolnia és gondoskodnia kell arról, hogy zökkenőmentesen működjenek ezek együtt, a *React*-nek nem kell mindezt megtennie. A *React* lényegében egy dolgot csinál, de azt rendkívül jól. A többi „eszközt”, – amelyekre végül szükség lehet – a nagyon aktív *React* közösség biztosítja. Ez a közösség olyan megoldásokat fejlesztett ki (azaz, extra csomagokat, amelyeket hozzá lehet adni a projekthez), mint például a *React* *router*, a *Redux* vagy a *Formik*. Tehát lehet azt állítani, hogy a *React* fejlesztő csapata arra összpontosít, hogy a lehető legjobb felhasználói felület-megjelenítő könyvtárat adja, míg a közösség az egyedi projektekre koncentrál, amelyek kiegészítik ezt a könyvtárat.

Az *Angular* csapata természetesen hatalmas tapasztalattal rendelkező fejlesztőkből áll, így nem valószínűsíthető programhiba bekövetkezte a több könyvtáron való párhuzamos fejlesztés miatt. Ezen túlmenően, az egész kezelése egy szervezeten belül történik, így természetesen ez azt is jelenti, hogy nem alakulnak ki verzióváltási problémák vagy inkompatibilitások. A különböző építőelemek mindig zökkenőmentesen működnek együtt azáltal, hogy egy csapat irányítja őket. Ez az *Angular* és a *Vue* esetében is igaz.

A fent felsorolt funkciókat a 7. ábrán látható táblázat keretrendszerekre lebontottan összegezi.



7. ábra Keretrendszerek és funkcióik [12]

### React, Angular és a Vue szintaxisának összehasonlítása

A keretrendszerek rövid bemutatását és funkcióinak összehasonlítását követően a különböző kódolási szintaxisok ismertetésévél folytatom. Az *Angular* projektek a *TypeScript*-et használják, amely nem futtatható a böngészőben, de az *Angular* projektek olyan eszközöket tartalmaznak, amelyek a háttérben lefordítják a *TypeScript* kódot a böngészővel kompatibilis *JavaScript* kódra. *Angularban*, mint fejlesztőnek, nem kell olyan kódot írni, amely közvetlenül hoz létre vagy távolít el elemeket a DOM-ból. Ehelyett komponenseket hoz létre, hogy a keretrendszer végezze el ezt a körülményes feladatot. Továbbá meghatározható a komponensek bemenetei (tulajdonságai) és kimenetei (eseményei), valamint kezelhető néhány komponens-specifikus vagy alkalmazás-szintű állapot.

A *React* általában JavaScript-et használ, és egy speciális JavaScript „funkciót” kezel, amelyet *JSX* -nek hívnak. A *JSX* valójában nem része a JavaScript nyelvnek. A *React* projektek úgy vannak beállítva, hogy ezt a „*HTML* a *JavaScriptben*” szintaxist támogassák a fejlesztés során. Úgy, mint a *TypeScript* az *Angular* projektekben, a *JSX* is böngészőbarát *JavaScript* kóddá kerül lefordításra.

A *Vue* normál *JavaScript*-et használ (bár opcionálisan használhatja a *TypeScript*-et is) és *Single File Components*-nek hívják, amit kezel. Ez egy olyan keret, amely ugyanúgy komponensekből tevődik össze, de az *Angularhoz* hasonlóan szétválasztja a *HTML* és a JavaScript logikát. A *Vue* támogatja a *JSX*-t is, de a projektek túlnyomó többségében a fent leírtak szerint alkalmazza a kódot.

E szintaxisok közül nem lehet egyértelműen meghatározni, hogy melyik a legjobb, hiszen ez teljes mértékben szubjektív, az egyéni preferenciától függ. Objektív mérések alapján egyik kódolási forma sem lassabb vagy gyorsabb a többitől.

### React, Angular és a Vue komplexitásának összehasonlítása

Mennyire könnyű megtanulni és használni ezeket a keretrendszereket? Véleményem szerint a három közül a *Vue* a legkönnyebben megtanulható keretrendszer. Ennek két fő okát emelném ki:

Nincs szükség speciális beállításokra. Csak importálni kell a *Vue* könyvtárat egy *HTML* fájlba és hozzáadni néhány JS-t ehhez a fájlhoz. További könnyebbsége, hogy nincs szükség egyedi projektbeállításra vagy utólagos fordításra.

Csak *JavaScript*-et és *HTML*-t használ, amelyet direktívák (úgy, mint *v-for*) használatával egészít ki. Ezen kívül nincs különösebb szintaxisa és nem szükséges más, ilyesfajta háttérismeret a *Vue* használatához.

Az *Angular* és a *React* komplexebbnek mondható, mint a *Vue* keretrendszer. Mindkettő bonyolultabb projektbeállítást és fejlett fejlesztői eszközöket (például *Webpack*) igényel az elinduláshoz. A projektek létrehozásában segítséget nyújt egyszerű illesztési beállításokkal az *Angular* CLI és a *create-react-app*. Továbbá fontos a *TypeScript* ismerete az *Angular* estében, a *React*-hez viszont elengedhetetlen a *JSX* szintaxis és a *React*-komponensek megértése (*props*, *hooks*).

### React, Angular és a Vue – teljesítményének összehasonlítása

Könnyű túl nagy hangsúlyt fektetni a teljesítményre, vagy rossz viszonyítási pontokat szemügyre venni és így nem a megfelelő következtetéseket levonni. Különböző típusú teljesítménytesztek léteznek a keretrendszerek vizsgálatára, amelyek közül az indítási és a futásidejű teljesítményt emelném ki, röviden ismertetve azokat.

Az indítási teljesítmény arra irányul, hogy milyen gyorsan töltődik be és válik interaktívvá a webalkalmazás. Az indítási teljesítményt elsősorban a létrehozott kódcsomag mérete befolyásolja, vagyis a saját és a keretrendszer kódja együttesen.

Gyakran egy alapvető *„Hello World”* alkalmazást vesznek igénybe, hogy összehasonlítsák a három fő keretrendszer kötegméretét. Az *Angular* lassabb, mint a *Vue* és a *React* alkalmazások, viszont a *Vue* minimálisan jobb mérési eredményekkel rendelkezik, mint a *React*.

A követkeő táblázatban *Lighthouse* [16] eszközzel mért indítási teljesítmény mérési adatait gyűjtöttem össze. A mérések a következő specifikációval ellátott számítógépen voltak futtatva: i7-8750H, 64 GB RAM, Linux 5.14.14, Chrome 95.0.4638.54 (64-bit). A kapott eredmények az 1. táblázatban láthatóak

1. táblázat Frontend keretrendszerek betöltési idejének mérései

|  |  |  |  |
| --- | --- | --- | --- |
| Keretrendszer | Angular | React | Vue |
| Verziószám | v13.0.0 | v17.0.1 | v3.2.21 |
| Betöltési idő | 2783,2ms | 2580,3ms | 2105,0ms |

Nagyobb alkalmazások esetén mindhárom keretnek nagyjából egyforma méretű kódcsomagokat kell készítenie, feltételezve, hogy olyan optimalizálási technikákat használ, mint a *Lazy Loading*, amelyet mindhárom keretrendszer támogat.

A futásidejű teljesítmény a már elindult alkalmazás működésének gyorsaságát jelenti. A futásidejű teljesítményt elsősorban a választott keret belső elemei befolyásolják, valamint az, hogy miként közelíti meg a DOM manipulációját és frissítését.

Végső soron a felhasználói élmény számít. Ha egy oldal gyorsnak érződik a végfelhasználó számára, akkor nem számít, ha technikailag kicsit lassabb a motorháztető alatt. A *React* nem csak arra koncentrál, hogy minden lehetséges *JavaScript* trükköt felhasználjon az összes szükséges felhasználói felület frissítésének hatékony észlelésére és végrehajtására, hanem olyan technikákat is felhasznál, amelyek során bizonyos feladatokat prioritássá tesz. Annak érdekében, hogy a végfelhasználó a már betöltött oldalon gyorsabb működést érzékelhessen. Ennek okán előfordulhat például, hogy elsőbbséggel kezeli a felhasználói bevitelt, és késlelteti a képernyőn megjelenő szövegek frissítését.

2. táblázat Frontend keretrendszerek teljesítmény mérései

|  |  |  |  |
| --- | --- | --- | --- |
| Keretrendszer | Angular | React | Vue |
| Verziószám | v13.0.0 | v17.0.1 | v3.2.21 |
| 1000 sor létrehozása | 122,2ms | 127,1ms | 106,5ms |
| Mind az 1000 sor frissítése | 105,1ms | 105,5ms | 92,7ms |
| Minden 10. sor frissítése 1000 sor erejéig (16x CPU lassítás) | 181,8ms | 231,0ms | 201,1ms |
| Egy kiválasztott sor kiemelése (16x CPU lassítás) | 72,7ms | 110,9ms | 31,0ms |
| 2 sor cseréje 1000 soros táblázatban (4x CPU lassítás) | 331,2ms | 332,8ms | 51,5ms |
| Egy sor eltávolítása | 20,9ms | 20,9ms | 22,4ms |
| 10 000 sor létrehozása | 1105,4ms | 1105,4ms | 999,3ms |
| 1000 hozzáfűzése egy 1000 soros táblázathoz (2x CPU lassítás) | 244,4ms | 244,4ms | 206,6ms |
| 1000 soros táblázat törlése (8x CPU lassítás) | 155,4ms | 155,4ms | 72,0ms |
| Átlag | 259,9ms | 270,3ms | 198,1ms |

A 2. táblázat értékeit elemezve megállapítható, hogy mindhárom keretrendszer nagyszerű futásidejű teljesítményt nyújt, viszont a *Vue* kimagasló eredményt produkál két sor cseréje és egy kiválasztott sor kiemelésénél.

### Frontend technológia kiválasztása

Az általam tervezett webalkalmazás könnyebben és gyorsabban elkészíthető valamelyik keretrendszer használatával, hiszen egy összetett problémát szeretnék vele megoldani, amely majd előreláthatólag bővíthető lesz további különböző funkciókkal. Így számomra egyértelmű döntés volt, hogy egy keretrendszer segítségével hozom létre az alkalmazásomat.

A kutatás alapján jól látszik, hogy a *React* nagyobb szabadságot biztosít az egyedi könyvtárak használatában, illetve a saját komponens alapú architektúra összeállításában és rendszerezésében. Véleményem szerint az a projekt létrehozásakor nagy előnyt nyújthat számomra. Továbbá előnye még, hogy rövidebb függőségi listával rendelkezik, mint az *Angular*, amelynek terjedelmes függőségeit sokszor nehéz nyomon követni. Ez úgyszintén jelentősen megkönnyíti és lerövidíti – számomra – a frontend fejlesztését. Ezeken kívül a *React* fejlesztői közössége hatalmas, így szinte minden felmerülő akadályra találhatunk valamilyen segítségnyújtást az interneten. Mindent összevetve, az eddigi tapasztalataim, kutatásaim és a személyes véleményem alapján a projekt létrehozására a *React* frontend keretrendszert választottam.

## Backend technológiák

Az egyik lefontosabb lépés a webalkalmazás készítésének folyamatában a megfelelő backend technológia kiválasztása [17]. A frontend írását a *React* keretrendszerben végzem, így az ehhez „illő” backend kiválasztása szintén kulcsfontosságú. A technológiai ipar ugrásszerű fejlődésének köszönhetően számtalan eszköz és backend keretrendszer áll rendelkezésünkre, amelyek zavart okozhatnak a megfelelő döntés meghozatalában. A nem optimális választás előre nem látható következményekkel járhat a jövőben, ezért elengedhetetlen az alapos kutatás és mérlegelés. A backend mondhatni az alkalmazás gerinceként működik, és olyan programozási nyelvek segítségével fejleszthetünk ilyeneket, mint például a *PHP*, a *Ruby*, a *Python*, majd ezekben olyan keretrendszerek közül válogathatunk, mint a *Django*, a *Spring* *Boot* vagy a *NodeJS*.

### React és a Node.js (Express)

Ki használja a *Node*-ot *React* háttérként? A Walmart [18] az első nagyvállalatok között volt, amely profitált a *React.js* és a *Node.js* együttes használatából. 2015-ben a vállalat mintegy 25 alkalmazását – amelyek e-kereskedelmi webhelyüket működtették – *Java*-ról a *React* és a *Node* technológiai kombinációjára költöztette.

Az *Express* egy gyors és minimalista keretrendszer, amely a *NodeJS*-re épül. [18] Ez lehetővé teszi a REST API-k egyszerű létrehozását azáltal, hogy alkalmas a kiszolgáló különböző kéréseihez futó kódok meghatározására. A *NodeJS* egy nyílt forráskódú, *Javascript* futásidejű környezet, amely a Google Chrome V8 JavaScript motorjára épül. Más szavakkal, a *NodeJS* a Google V8 *Javascript* motorját használja a kód végrehajtására. A *Node.js* és a React.js együttes használatának több előnye is van egy single page applikáció megírásánál. A technika ezen kombinációja lehetővé teszi, hogy:

SEO-barát (a Google és más keresőmotorok hatékonyan feltérképezhetik a webhely minden oldalát, a keresett témák alapján a legrelevánsabb és legértékesebb weboldalakat szolgálhatják ki felhasználóik számára) single page applikációkat hozzon létre, amelyek kihasználják az alkalmazás nézetek szerveroldali megjelenítését.

Ugyanazt a V8 *JavaScript*-motort használja kliens és szerveroldali megjelenítéshez.

Használja az *npm* (*Node Package Manager*) kódcsomagokat a webalkalmazás-fejlesztési ciklus felgyorsítása érdekében.

Használja a *Node.js* modulokat a *React* alkalmazás egyetlen fájlba csomagolásához.

Az *Express.js* nyílt forráskódú harmadik fél moduljaként kerül kiterjesztésre a *Node.js* fájlhoz. A *React* könyvtár célja az építészeti szabadság, az *Express.js*-szel együtt történő használata lehetővé teszi a magasan testreszabott alkalmazások készítését. Az *Express.js* keretrendszer könnyű és moduláris. Alapja csak olyan alapvető funkciókat tartalmaz, mint például az útválasztási (*routing*) rendszer, a munkamenetek és a cookie-k támogatása, a MIME segítők, a *RESTful* felület stb., amelyek szükség esetén opcionális modulok vagy köztes eszközök hozzáadásával bővíthetők. Ennek a keretrendszernek egy másik előnye a nagy sebesség és az egyidejű kapcsolatok ezreinek másodpercenkénti kezelése. A teljesítménye emellett javítható gyorsítótárazás vagy több folyamatra skálázás révén. Mivel mindez magas fokú effektivitást eredményez, az *Express.js* aligha válhat szűk keresztmetszetévé egy nagy terhelésű *React* alkalmazás számára.

### Laravel és a Ruby on Rails összehasonlítása

Minden bizonnyal megfelelő választás az *Express* a feladat megvalósítására backend oldalról, azonban érdemes megnézni a további lehetőségeket is. Két elég gyakori *React*-hez való backend párosítást szeretnék megvizsgálni, amelyek a *Laravel* és a *Ruby* *on* *Rails* keretrendszerek.

A *Laravel*-t, mint *PHP* alapú backend keretrendszert azért tartom fontosnak megtekinteni, ugyanis, a PHP-t az összes olyan jelenkori webhely több mint 80%-a használja, amelyek szerveroldali programozási nyelve ismert (2017-es adat) [19]. A *Ruby* *on* *Rails* minden bizonnyal kevésbé közismert, mint backend keretrendszer, azonban használatával meglehetősen gyorsan lehet akár összetett üzleti logikával rendelkező API-okat létrehozni. [20] Ennek okán ezt a két keretrendszert is szeretném a kutatásomba venni és összehasonlítani őket.

* A *Laravel* egy modern webes framework [21], amellyel egyszerűen és gyorsan lehet tervezni webes alkalmazásokat, majd ezek testreszabását könnyedén el lehet elvégezni – csak úgy, mint az *Express* esetében –. Ez egy nyílt forráskódú és gyakran használt keretrendszer *PHP*-ban. Számos olyan funkcióval rendelkezik, amik vonzhatják a fejlesztőt egy webes alkalmazás létrehozásához *Laravelben*.
* A *Ruby* *on* *Rails* egy szerver oldali webes alkalmazáskeret, amelyet *Ruby* programozási nyelvben hoztak létre. [21] Előnye, hogy számos kész funkcióval segíti a fejlesztőt, így ő egyszerűen az üzleti logikára tud koncentrálni, ahelyett, hogy elmélyülne a keretrendszer bonyolultságaiban. A két keretrendszer különbségeit a 3. táblázat foglalja össze.

3. táblázat A Laravel és a Rails backend keretrendszerek különbségei

|  |  |  |
| --- | --- | --- |
| **Az összehasonlítás alapja** | **Laravel** | **Rails** |
| **Nyelv** | PHP alapú | Ruby alapú |
| **Használat** | Vállalati szintű alkalmazások | Adatbázis-alapú webalkalmazások |
| **Megjelenés** | 2011 június | 2004 |
| **Sablon motor** | Blade | ERB |
| **ORM** | Eloquent | ActiveRecord |
| **Jellemzők** | * Gyors ORM * saját CLI * hitelesítési lehetőségek | * Tiszta szintaxis * harmadik féltől származó beépülő modulok * jó MVC-támogatás |

Jól látható, hogy mindkét keretrendszernek megvannak a sajátos tulajdonságai, amelyek egyedivé teszik őket. A *Rails* olyan keretrendszer, jó dokumentációval, útmutatókkal és kiterjedt könyvtárakkal rendelkezik. Megjelenése korábbi, mint a *Laravel*-é, így közössége erősebb és népszerűbb a fejlesztők körében. A *Laravel* számos olyan funkcióval rendelkezik, amellyel alkalmas bármilyen webes alkalmazás létrehozására. ORM-eszköze, az *Eloquent*, a CLI-eszköze és az *Artisan* lehetővé teszi a fejlesztők számára, hogy különböző feladatokat hajtsanak végre a gyorsítótár törlésével, ezáltal könnyedén migrálják az adatbázist. Dokumentációja alapos, és a nyelv minden aspektusára kiterjed.

### A backend technológia kiválasztása

A backend kiválasztása az egyik legfontosabb teendő [22], mivel ez felel az oldal teljesítményéért. Hogyan válasszuk ki a megfelelő háttértechnológiát a webalkalmazáshoz? A választást tovább nehezíti, hogy nem lehet egyértelműen meghatározni, hogy melyik nyelv, keretrendszer vagy futtatókörnyezet jobb a másiknál, ugyanis mindegyiknek megvannak a saját előnyei és korlátai. Egyes projektekben a különböző technológiák egymással felcserélhetően használhatók, más esetekben ez lehetetlen lenne, ezért fontos a megalapozott döntés. A kutatásom alapján a *Node.js* és *Express* párosítása bizonyult a legmegfelelőbbnek a backend megalkotására. Emellett a *React* és a *Node.js* is *JavaScript*-et használ, amely megkönnyíti számomra a backend és frontend párhuzamos fejlesztését azáltal, hogy nem kell egyszerre két programozási nyelvben is gondolkodni.

## Webalkalmazás megtervezése

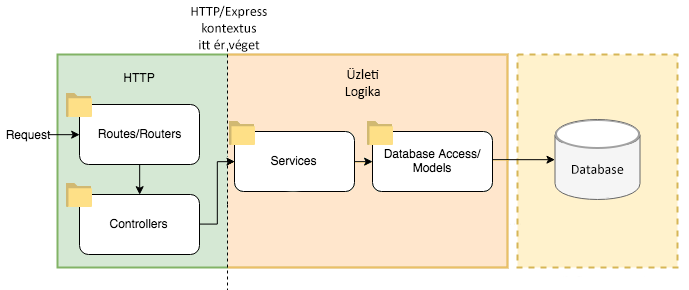
A már kiválasztott technológiák sajátosságaira tekintettel tervezem meg a webalkalmazásom felépítését, gondolva itt a *React* komponens-szerkezetére, illetve a *Node.js* és *Express.js* által nyújtott backend tervezési mechanizmusokra. Továbbá bemutatom, hogy hogyan és hol helyezkedik el az általam tervezett webalkalmazás az egész mestertkeresek.hu vállalkozás struktúrájában.

### Backend tervezés

A frontend API (Application Programming Interface) segítségével kommunikál a backenddel. Ha frontenden szükség van valamilyen adat megjelenítésére vagy szerkesztésére az adatbázisból, akkor a frontend az API-n keresztül egy kérést küld a backendnek, amely feldolgozza azt, és a megfelelő választ küldi vissza. Ezek a kérések HTTP alapúak, amely egy kérés-válasz alapú protokoll a szerver és a kliens között. Ebben a részben a kérések feldolgozásával foglalkozom, és onnantól kezdve mutatom be [23], hogy a kérés eléri a backend alkalmazást, áthalad az egyes rétegeken és választ küld a frontendnek. A már korábban kiválasztott *Node.js* és *Express.js* backend technológia segítségével hozom létre a tényleges feldolgozást végző réteget (backend). Először az architektúrát tervezem meg (8. ábra), majd annak mintájára a projekt struktúrát (9. ábra) és azok elemeit.

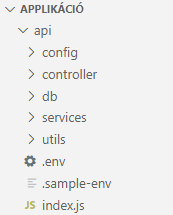
Az architektúra rétegei két fő részre bonthatóak:

* HTTP réteg: Itt az API útvonalak (*routes*) és azok vezérlői (*controller*) szerepelnek, amelyek elérik a HTTP kéréseket és kinyerik a szükséges adatokat a kérésből, majd a megfelelő szolgáltatásokhoz (*services*) irányítja azokat.
* Üzleti logikai réteg: Tartalmazza az üzleti és műszaki követelményekből származó üzleti logikát, valamint az adatbázishoz való hozzáférést. Az adatelérési réteg logikája gyakran a "technikai" üzleti logika, mivel gyakran a követelmények határozzák meg a lekérdezéseket.



8. ábra Backend architektúra diagram

A *Routes/Routers* (magyarul útválasztó) kezeli az API-t, és semmiféle üzleti logikát nem tartalmazhat. Csak a vezérlők funkcióit szabad összeláncolnia. A projektemben egy *index.js* fájlba helyezem őket (9. ábra).



9. ábra Backend projekt struktúrája

A *controller* rétegben kerülnek meghívásra a *service* réteg metódusai, így külön lehet „tartani” az üzleti logikát. Önmagukban a *controller*-ek sem tartalmaznak más logikát, csak az útválasztótól kapott HTTP kéréseket kezelik, és választ adnak vissza, vagy folytatják a hívásláncot *service* hívásokkal. Itt kerül továbbá eldöntésre, hogy mi történjék a visszaküldött adatokkal, és a válasz részeként kezelik a HTTP állapotkódokat is. Ezzel itt véget ér az *Express/HTTP* kontextus kezelése. Ennek nagy előnye, hogy az alkalmazás többi része nem függ a kérés objektumtól (*request*), így, ha az *Express* keretrendszer cseréjére kerülne a sor, ez sokkal egyszerűbben megvalósítható.

A *services* réteg tartalmazza a logika nagy részét, amely magába foglalja az üzleti követelményeket, meghívja az adatelérési réteg funkcióit, és olyan értékeket szolgáltat vissza a *controllernek*, amelyek alkalmasak a továbbküldésre. ezáltal elválasztva az adatfeldolgozást az adatbázissal való kommunikációtól. Az adatelérési rétegbe (9. ábrán a *db* mappa) kerül a DB-vel (Database) való kapcsolat létesítése és az adatok eléréséhez szükséges logika. Az adatbázissal való kommunikáció lehet ORM-en (Object-Relational Mapping) keresztül modellek létrehozása segítségével, vagy SQL lekérdezésekkel. Én az utóbbit választom, ugyanis feladatomban sokszor kell JSON formátumú adatokban módosítást vagy lekérdezést készíteni, amely a magas szintű ORM utasításokkal nehezebben (vannak olyan ORM-ek, amik ezt nem is támogatják) megvalósítható, illetve a lekérdezési idő is nagyban megnövekszik bármelyik ORM használatánál. A 4. táblázatban az egyik legnépszerűbb ORM, az *ORM2* és a nyers SQL lekérdezés mérési eredményei láthatóak.

4. táblázat Adatbázis lekérdezés válaszidők mérései

|  |  |  |
| --- | --- | --- |
|  | ORM2 | Közvetlen SQL |
| Idő | 555ms | 137ms |

A 9. ábrán látható még további két mappa, amelyek a *config* és a *utils*. Utóbbiban azok a segédfunkciót betöltő gyakori logikai függvények kerülnek, amelyek nem feltétlenül az üzleti logikára vagy általában egy API-ra vonatkoznak. Például saját metódusok egy probléma megoldására, amelyet több helyen is fel lehet használni. A *config* mappába kerülnek azok az egységek, amelyek a backend alkalmazás konfigurálásához szükségesek, például az adatbázis kapcsolat megvalósítása és a környezeti változók beállítása.

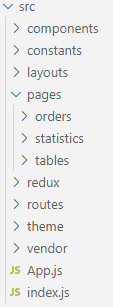
### Frontend tervezés

Érdemes előre megtervezni, hogy hogyan helyezzük el a fájlokat a mappákban, annak ellenére, hogy a *React* ezt nem határozza meg. Ehhez először célravezető elgondolkodni azon, hogy milyen komponens-könyvtárakra van szükségünk az alkalmazás megvalósításához, például *Material* *UI*, *React-Bootstrap*, *React* *Router*, *Redux*, *Axios* stb.

A React alkalmazásomban használt néhány főbb komponens könyvtár az alábbi.

* *Material UI*: Az egyik legnépszerűbb könyvtár [24], amellyel könnyen és gyorsan elkészíthető a felhasználói felület. Elsődleges feladata, hogy a fejlesztők számára számos és nagymértékben testreszabható összetevőket adjon, ezzel lehetővé téve a saját tervezési rendszer (design) felépítését.
* *Redux*: Segít a React komponensek több komponenshez való összecsatolásában, és a konzisztens kódok írásában. Továbbá használható a kód szerkesztésére éppen futó alkalmazásban.
* *Axios*: Ez egy olyan könyvtár [25], amely a kérések (request) küldésében segít. Adatokat szolgáltat az API-nak, illetve annak válaszait (response) kezeli. Így az alkalmazásban könnyedén végezhetünk műveleteket az adatokon.
* *Moment*: Egyszerűbbé teszi a dátumok megjelenítési formázását.
* *Sweetalert2*: Könnyedén testreszabható, igényes megjelenítést biztosít a felugró ablakoknak (alert).
* *Styled* *Components*: Lehetővé teszi a komponens szintű stílusok használatát az alkalmazásokban.

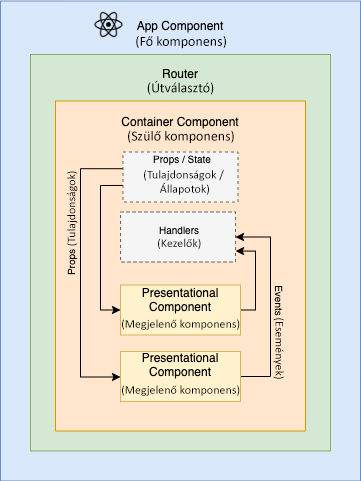
A *React* applikációm felépítését úgy készítem el (10. ábra), hogy az könnyen áttekinthető legyen, illetve tervezem a későbbi új funkciók implementálásának lehetőségeit is.



10. ábra Frontend projekt struktúra

A *components* mappa alá kerülnek azok a komponensek, melyek olyan funkciókat tartalmaznak, amik nem kimondottam egy oldalhoz vagy megjelenítendő részhez tartoznak. Ilyenek például a navigációs sáv, beállítások kezelőfelülete, vagy akár egy, az oldal töltése közben megjelenő animáció. A *constants* mappába kerülnek kigyűjtésre azok a konstans értékkel rendelkező változók, melyekre számos alkalommal van szüksége az alkalmazásnak. A *redux* mappa részét képezi az *actions*, *reducers* és a *store*, amelyet az alkalmazásomban a téma választás kivitelezésére használok, melyek formái a *theme* mappában vannak megvalósítva.

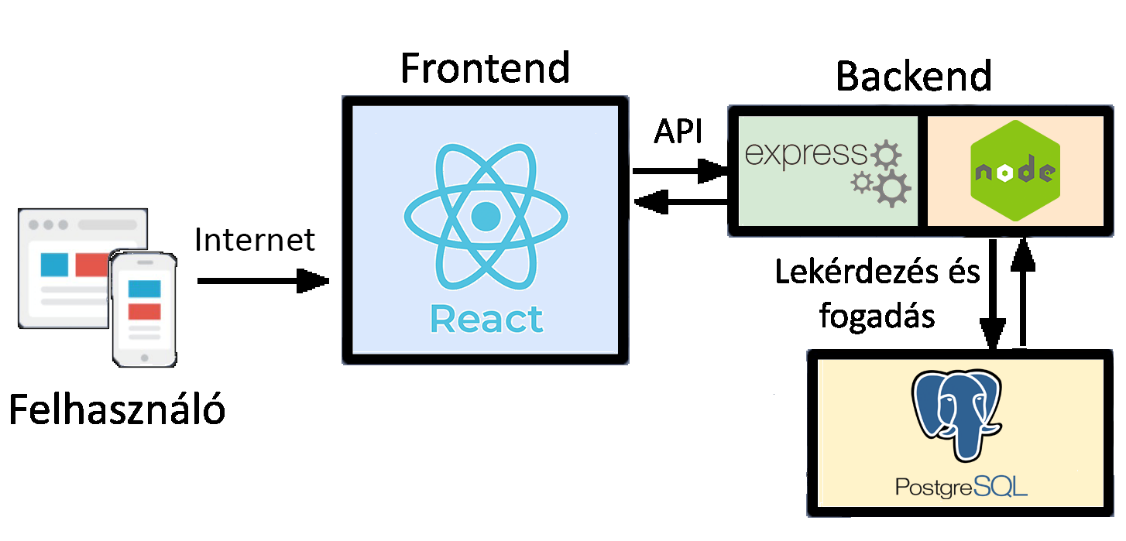
A *pages* felépítését a megjelenő oldalak szerint csoportosítottam, melyek az *orders* (Megrendelések), *statistics* (Alap Adatok) és a *tables* (Táblák), melyek képernyőterveit a 4.2. szakasz ábráin lehet látni (2-5. ábra). Ezen oldalak komponenseit és eléréseit a *routes* köti össze. Egy ilyen felépítési réteget szemléltet a 11. ábra, amely a komponensek közötti kapcsolatot is reprezentálja.



11. ábra Frontend komponens rétegek

### Frontend, backend és az adatbázis kapcsolata

Ebben az alfejezetben bemutatom az eddig megtervezett alkalmazás részek kapcsolatát, melyet a 12. ábra kifejezően illusztrál.



12. ábra PERN Stack

Ezt a fajta full-stack webalkalmazást [26], amely *PostgreSQL*, *Express*, *React* és *Node* elemekből áll, *PERN* *Stack*-nak nevezik.

* P – *PostgreSQL*: ORDBMS (Object Relational Database Management System), SQL alapú adatbázis-kezelő rendszer.
* E – *Express*: Kiszolgálóoldali keretrendszer, amely a *Node* *JS*-re épül, hogy megkönnyítse a szerveroldali feladatokat.
* R – *React*: *JavaScript*-könyvtár, amely segít összetett felhasználói felületek felépítésében és kommunikál a szerveroldali technológiákkal.
* N – *NodeJS*: Szerveroldali technológia, amely lehetővé teszi a *JavaScript* futtatását egy szerveren.

# Implementáció

## A fejlesztéshez használt szoftverek

A frontend és backend fejlesztéshez valamint az adatbázis táblák megtekintésére is a *Visual* *Studio Code-ot (VS Code),* a Microsoft ingyenes, nyílt forráskódú szövegszerkesztőjét használtam. Ehhez telepítettem a *Prettier* kódformázót és a *PostgreSQL* *Management Tool* bővítményeket. Továbbá az *npm* (*Node* *Package* *Manager*) parancssoros szoftvercsomag-kezelő segítségével futtattam a *React* és *Node.js* szervereket.

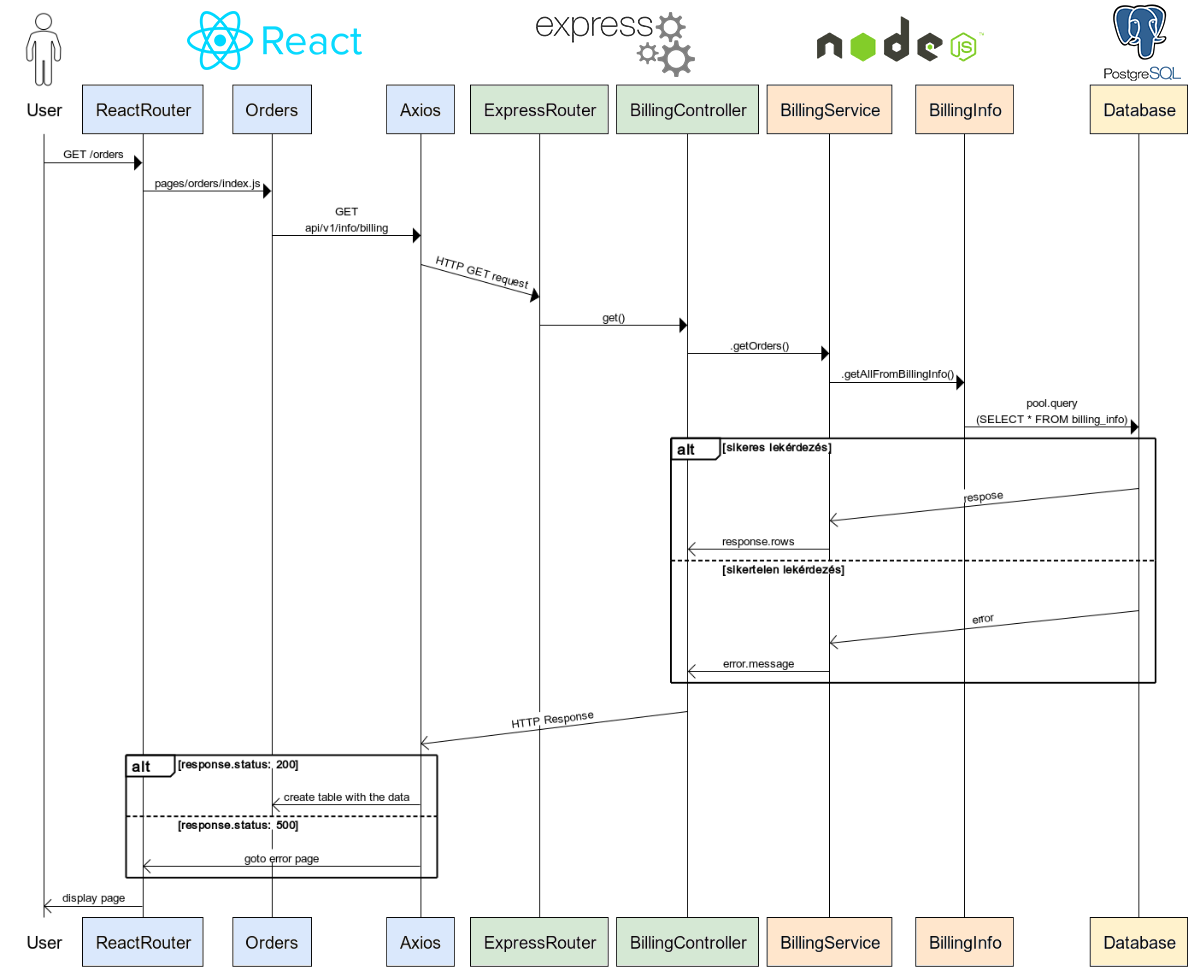
A fejlesztés során a *Postman* nevű ingyenes alkalmazást használtam az API tesztelésére, amely számos végpont-interakciós módszert kínál (GET, POST, PUT, PATCH, DELETE, stb.). A kliens oldali hibák megtalálására a Google Chrome webböngésző beépített fejlesztői eszközét, valamint a *React* *Developer* *Tools* bővítményt használtam. Az adatbázis elkészítését és futtatását a *pgAdmin* nevű alkalmazásban kezeltem, amely egy adminisztrációs és fejlesztési platform a *PostgreSQL*-hez.

## Program bemutatása

Az alkalmazás működését, és az egyes funkciók programozási kódjait a teljes adatfeldolgozási folyamaton végig vezetve, példákon keresztül mutatom be, a frontendtől a backenden át az adatbázisig.

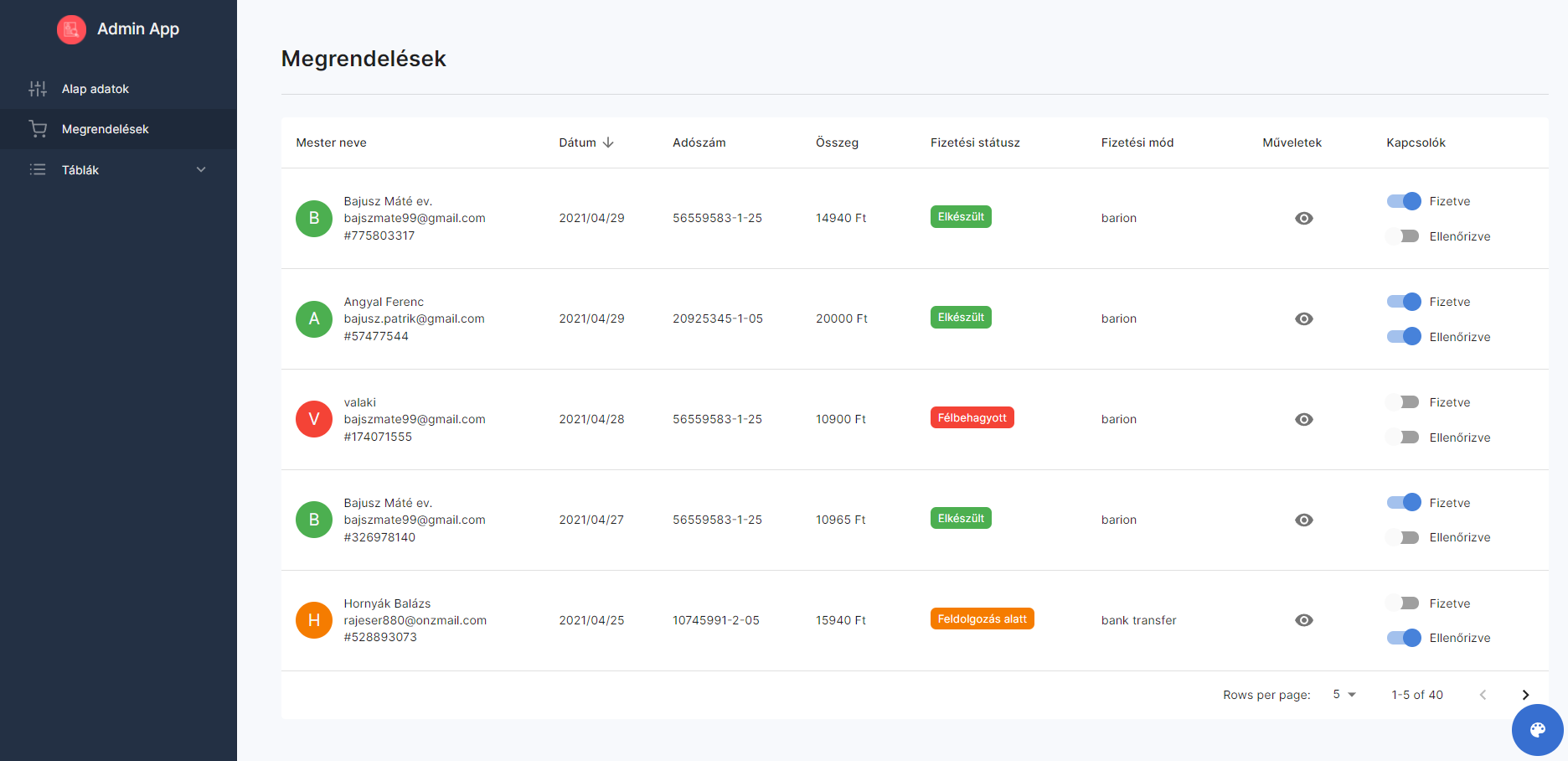
### Megrendelések

Az első példán az alkalmazás *megrendelések* funkciójának (4.2.1. pont) adatmegjelenítését mutatom be. A 13. ábrán látható szekvencia diagramon lehet végigkövetni a folyamatot. Azonban nem minden elágazási és egyedi részlet van megjelenítve az ábrán, ezek részleteire a későbbiekben térek ki.



13. ábra Megrendelések funkció szekvencia diagram

A folyamat onnan kezdődik, hogy egy felhasználó megnyitja (lokálisan futtatva: *http://localhost:3000/orders*) a Megrendelések oldalt. Ez, a szinte minden mai weblapon megtalálható navigációs sávból (navbar) érhető el. Azonban, ha nem egy létező és aktív oldalt próbálunk megnyitni, akkor az alkalmazás átirányít egy hiba oldalra, ahonnan vissza lehet lépni a főoldalra. A megrendelések funkció könnyebb megértése érdekében egy képernyőképet mutat az oldalról a 14. ábra.



14. ábra Megrendelések oldal képernyőkép

Az alkalmazás működésének tényleges logikája a *Routes* komponensben kezdődik, ahol az alkalmazás kétfajta megjelenési elrendezés (layout) közül választ. Az egyik a „normál” felhasználás során történő megjelenés, a másik a már korábban említett hiba megjelenési oldal. Ezeket azért csoportosítottam külön, hogy ha a későbbi új funkciók másfajta megjelenítési módot igényelnek, akkor ez gyorsabban implementálható lesz.

A *Routes* komponens objektum típusú változóiban vannak eltárolva azok, hogy az egyes frontend-végpontokon (endpoint) milyen komponensek jelenjenek meg.

const orderRoutes = {

  id: "Megrendelések",

  path: "/orders",

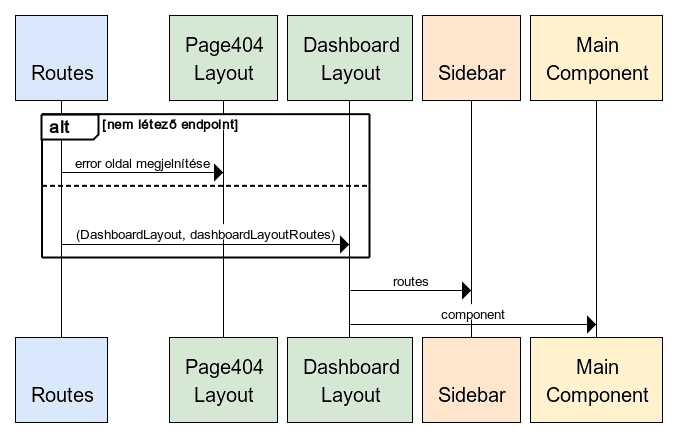
  icon: <ShoppingCart />,

  component: Orders,

  children: null

};

Ezeket az értékeket használja fel a *Sidebar* komponens, amely a navigációs sáv megjelenítéséért és logikájáért felel. Ezek az elemek, egyfajta gombként jelennek meg és adott esetben lenyíló menüt (dropdown) hoznak elő. Linkek segítségével irányít át egy másik frontend-végpontra, mindezt rendkívül rövid idő alatt, úgy, hogy a böngészőben nem történik ablakváltás. A töltési idő közben egy animáció jelenik meg, amelyre a *Loader* komponens szolgál. Ezeket a *Sidebar* és *Loader* komponenseket a *components* mappában (10. ábra) helyeztem el, hiszen ezek nem kimondottam egy oldal megjelenítéséhez köthetőek, hanem az összes „normál” megjelenési elrendezésen (layout) helyet kapnak. Ezt a folyamatot, ahogy a *Routes-*ból a megjelenítendő fő komponensig jutunk, a 15. ábra mutatja be.



15. ábra "Routes-tól a Fő komponensig" szekvencia diagram

A megrendelések oldal fő komponense a *„pages/orders/*” mappában található *index.js* file. Itt található a 14. ábrán látható táblázat *EnhancedTable* komponens meghívása. Ebben egy *Material* táblázatot hoztam létre, melynek fejlécét és testét külön komponensekre szedtem, ezek az *EnhancedTableHead* és az *EnhancedTableBody*. Ezen kívül, itt kerül meghívásra a backend API, *Axios* segítségével, majd a visszakapott adatok (response) átadódnak a táblázat törzsének (*EnhancedTableBody* komponens). Továbbá a táblázat alján szerepel még, a táblázat sorainak számát mutató beállítási lehetőség, illetve ezen táblázat oldalakra felosztása közötti navigációs nyilak (14. ábra).

A megrendelések funkció további bemutatását az alkalmazás működési folyamatával megegyező módon folytatom. Az *EnhancedTable React* komponensében történik a szükséges adatokhoz való HTTP GET kérések elküldése. Az alább bemutatott programkódban látni, hogy a táblázat megjelenítéséhez két API kérés szükséges, melyek a *„/api/v1/info/billings”* és a *„/api/v1/workers”* végpontokon érhetőek el.

useEffect(() => {

    const fetchData = async () => {

      setIsLoaded(false);

      try {

        const [billing, workers] = await axios.all([

          axios.get(API\_URL + "/info/billings"),

          axios.get(API\_URL + "/workers"),

        ]);

        setData(billing.data);

        setWorkers(workers.data);

      } catch (error) {

        setError(error);

      }

      setIsLoaded(true);

    };

    fetchData();

  }, []);

  if (error) {

Ettől a folyamattól kezdve a *React* alkalmazás addig vár, amíg választ nem kap az API-tól. Az API GET hívása HTTP protokollon keresztül történik, és ennek kezelését a backend *Express* keretrendszere végzi. A backend legfelső *Routes* rétegében kerül meghívásra az adott kéréshez tartozó *Controller*. Példámban most csak egy, a *„api/v1/info/billings”* végponthoz tartozó API kérést folyamatát mutatom be, hiszen a másik *„api/v1/workers”* hívása is közel hasonlóan történik. A 13. ábrán jól látható módon történik ez a művelet, így ennek a részfolyamatnak bemutatásához nem tartottam szükségesnek újabb szekvenciadiagram létrehozását.

Az alábbi kódrészletben a *billingController* az API *„api/v1/info/billings”* végpontjához meghívott metódusa látható.

async function get(request, response) {

  try {

    let rows = await billingService.getOrders();

    response.status(201).json({ message: "successful query", rows });

  } catch (err) {

    response.status(500).json({

      message: "could not find billing info",

      error: err.toString(),

    });

  }

}

Ebben a *JavaScript* programkódban található a *billingService.getOrders()* metódusának meghívása, majd annak sikeres lefutása után a válasz (response) visszaküldése. Ennek hibára futása esetén, a megfelelő hibaüzenet és az 500-as HTTP státuszkód kerül visszaküldésre.

A *billingService getOrders()* metódusa hívja meg az adatbázissal kommunikáló egységet, majd annak válasza után visszaadja az eredményt a *billingController*-nek. Az adatbázis elérése a „config/repository.js” fájlban előre konfiguráltan történik a *„pg”*, *Node* csomag segítségével, amely egy *Pool*-t hoz létre, és ezt használják a *service*-okban meghívott metódusok. Az alábbi kódrészletben a megrendelések funkcióhoz szükséges számlázási információk lekérése látható.

async function getAllFromBillingInfo() {

  return await pool.query(`SELECT \* FROM billing\_info`);

}

Ennek eredményével tér vissza a *service*-ba, majd onnan a *controller*-be. A *controller* sikeres lekérdezés esetén egy JSON formátumú eredményt küld vissza (response), melynek megvalósítása a fentebb bemutatott *billingController get* metódus kódjában látható.

Ez a backend folyamat, – vagyis a *billing\_info* tábla adatai eljussanak a frontendre – a *React Axios* kérésétől számolva nagyságrendileg 30-150 ms volt az én futtatási környezetemben. Lényegében, megközelítőleg ennyi idő volt az, amíg a frontend várakozott a backend válaszára. A megrendelések funkció bemutatását folytatva, visszatérek a frontend alkalmazásra, ahova sikeresen visszaérkezett az API-tól minden szükséges adat a táblázat megjelenítéséhez.

A React *EnhancedTable* komponense meghívja az *EnhancedTableBody* komponensét a backendtől visszakapott adatokkal. Utóbbiban egy *Material* Táblázat teste és azon belüli sorok kerülnek létrehozásra, melynek rövidített programkódját mutatom be. A „…” helyén további kódrészek találhatóak.

<TableBody>

      {stableSort(data.rows, getComparator(order, orderBy))

        .slice(page \* rowsPerPage, page \* rowsPerPage + rowsPerPage)

        .map((row) => { …

            <TableRow>

              <TableCell> …

A táblázat első oszlopában található monogrammal ellátott *„avatar”* háttérszíne, valamint a *Fizetési státusz* oszlop adatainak háttérszíne az adott megrendelés feldolgozottsági szintjét indikálja (14. ábra). Ennek logikáját az alábbi kódrészlet mutatja be.

  const avatarColor = (completed, payment\_method) => {

    if (completed === true) {

      return "#4caf50"; //zöld

    }

    if (payment\_method === "bank transfer" && completed === false) {

      return "#f57c00"; //narancs

    }

    if (

      (payment\_method === "barion" || payment\_method === null) &&

      completed === false

    ) {

      return "#f44336"; //piros

    }

  };

Akkor zöldek ezek a mezők, ha a számlázási információkat tartalmazó táblázat (*billing\_info*) *completed* mezőjének értéke *„TRUE”* (*Elkészült* státusz)*.* Ez sikeres bankkártyás fizetés vagy egy, már leellenőrzöttbanki átutalás után lehetséges. Narancssárgán jelenik meg a még függőben lévő banki átutalások (*Feldolgozás alatt* státusz) és pirosan a félbehagyott bankkártyás fizetések *(Félbehagyott* státusz*)*.

Az adatok soronként megjelenítése a *„sor.objektumKulcs”* módon történik további *Material* komponensek segítségével. Azonban a megrendelések funkció nem csak az adatok megjelenítésével foglalkozik, hanem azok szerkesztésével is, ezek a banki átutalások (3.1. szakasz) és a mesteremberek ellenőrzöttségi státuszok állítását jelentik (3.4. szakasz).

A banki átutalások a *billing\_info* tábla *completed* mezőértékének átállítását és az adott felhasználóhoz tartozó *worker\_plan* beállítását jelenti. Már frontend oldalon is történik egy ellenőrző folyamat, amely azt hivatott elősegíteni, hogy véletlenül se kerüljön módosítás az adatbázisban, olyan helyen, ahol az az adatbázis logikájában nem megengedett. Ilyen például, az online bankkártyás fizetések módosítása. Ha egy *Barion*-os fizetést próbálnánk módosítani, akkor egy figyelmeztető ablak (alert) jelenik meg, amelyet a *Sweetalert2* külső könyvtár segítségével oldottam meg. Ennek megvalósítását mutatja be az alábbi kódrészlet.

<Switch

  checked={row.completed}

  onChange={() => {

    if (row.payment\_method == "bank transfer") {

      handleIsPaid(event);

    } else {

      Swal.fire({

        icon: "info",

        title: "Kártyás fizetést nem tudsz módosítani",

        showConfirmButton: false,

        timer: 1800,

      });

    }

  }}

  value={row.id}

/>

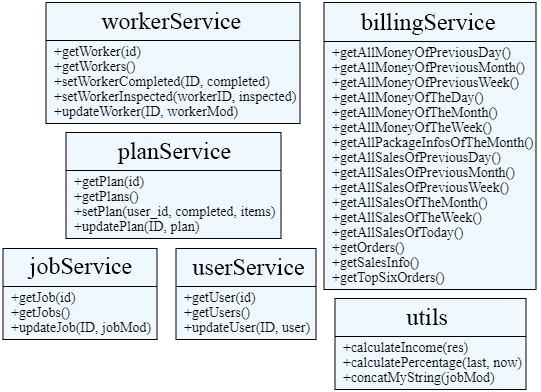
A fenti program kódban látott *handleIsPaid* függvényben történik az API hívás. A backend *Express* folyamatai hasonlóan néznek ki, mint a már korábban bemutatott *„api/v1/info/billings”* GET kérés. Azonban ebben a példában egy HTTP PUT kérést küld az *Axios*, melynek folyamatát a backend service metódusainak bemutatásával folytatom.

Ahhoz, hogy a megrendelést *completed* státuszba lépjen a számlázási információkat tartalmazó táblázat erre vonatkozó mezőjén kívül (*billing\_info:completed*)a felhasználó által megvásárol csomag, és kiegészítő szolgáltatások is eltárolásra kerülnek a *worker\_plan* táblában. Az módosítani kívánt számlázási információt azonosítóját, illetve a *completed* mező új értékét kapja meg a meghívott *service* függvény, amelyet tovább ad az adatbázissal kommunikáló rétegnek. A már korábban bemutatotthoz hasonlóan, az adatbázis eléréséről és annak sikeres frissítéséről a *controller* visszaküld egy választ (response) a frontendnek.

A szakember ellenőrzöttségi státuszára (3.4. szakasz) szolgáló kapcsoló az imént bemutatott folyamathoz hasonlóan történik, azonban itt nem történik a frontenden semmiféle ellenőrzés, hiszen akár egy, még be nem fizetett számlához tartozó szakembert is le lehet ellenőrizni, és ezzel akár gyorsítani az ügyintézési folyamatot. A programban, – az adatbázis kapcsolatoknak köszönhetően – ha egy már ellenőrzött felhasználónak lejárt az előfizetése és újabb csomagot vásárol, akkor automatikusan megmarad az „Ellenőrzött Mester” státusza.

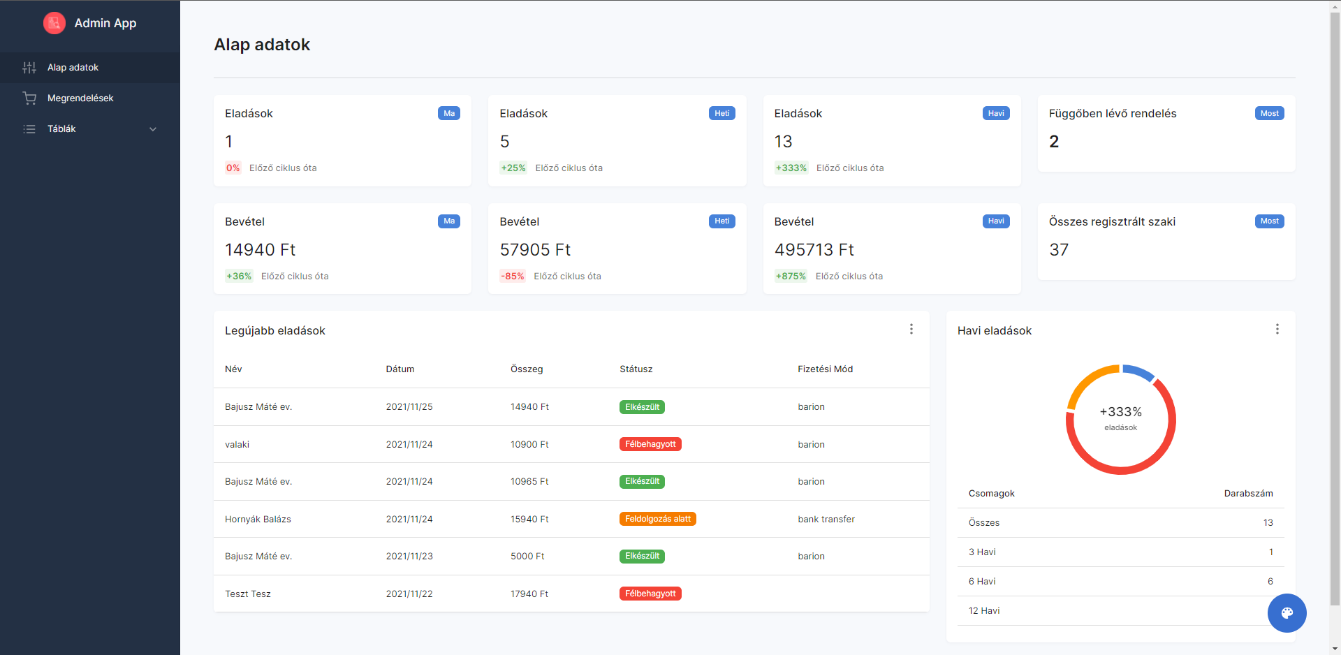
### Alapvető adatok

Az előző szakaszban a megrendelések funkció működési folyamatát mutattam be, azonban ezen kívül még számos más funkcióra képes az alkalmazás. Ennek bemutatására a backend *Node.js* rétegének osztály diagramja ad közelebbi rálátást.



16. ábra Backend services osztály diagram

Ebben a példában az alapvető adatok funkciót (4.2.2. pont) mutatom be, amely a vállalkozás kisebb statisztikai adatait jeleníti meg. Az eladások darabszáma és a bevétel forint alapú megjelenítése mellett szerepel az aktuális hónapban eladott csomagok megoszlási diagramja, valamint az utolsó hat eladás részletei. Ezen információk megjelenítéséhez szükséges API kérések, a 16. ábrán látható billingService függvényei közül a *getOrders()*, *getSalesInfo()* és a *getTopSixOrders()* szolgáltatja az adatokat. Ezeknek az API hívásoknak a folyamata hasonlóan működik, mint a korábbi példában, igy ezeket a továbbiakban nem részletezem. A funkció szemléltetésére a 17. ábrán látható képernyőkép szolgál.

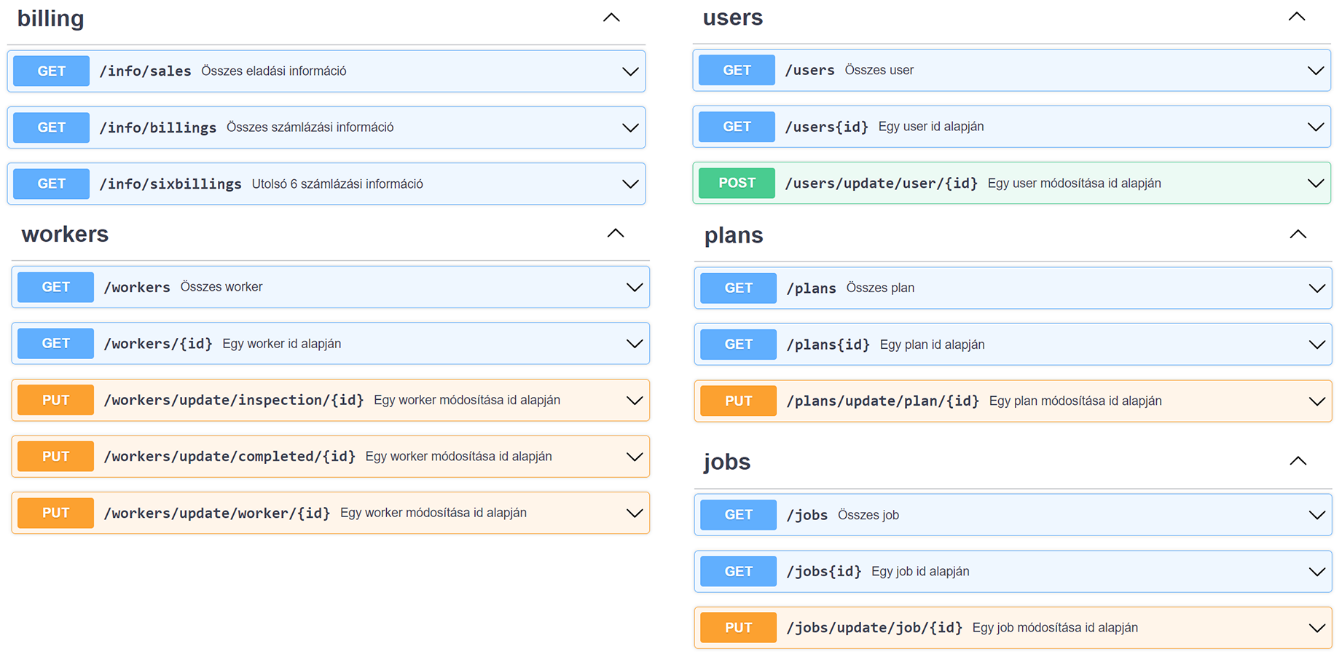


17. ábra Alap adatok oldal

### Adatok szerkesztése

Az alkalmazás képes az adatbázis táblákban tárolt adatok megtekintésére és űrlapszerű szerkesztésére. Ez alól kivételt képeznek az olyan információkat tároló táblák, melyek mondhatni konstans értékekkel rendelkeznek. Ilyenek például, a *location*, amelyben Magyarország települései vannak összegyűjtve, vagy a *categories* és a *profession*, amelyben pedig az előre meghatározott munkakörök vannak eltárolva.

Az adatok megtekintésére szolgáló oldalak a navigációs sávban található „*Táblák*” menüpont alatt érhetőek el. Az adatok táblázatokban jelennek meg, amelyek első oszlopaiban jelölő négyzetek találhatóak. Ezek az adott sor kijelölésére szolgálnak, amellyel a CSV formátumba való exportálás funkciót teszi érhetővé. Az adatok szerkesztéséhez az adott sorban elhelyezett gombra kattintva egy új felület jelenik meg, amelyen a szerkeszteni kívánt rekord jelenik meg a hozzá tartozó értékekkel. Ehhez az API-nak az adott rekord azonosítójának átadása szükséges, amelyet az adatok megjelenítésére szolgáló komponensből ered. Az API összes végpontját a 18. ábra mutatja be.

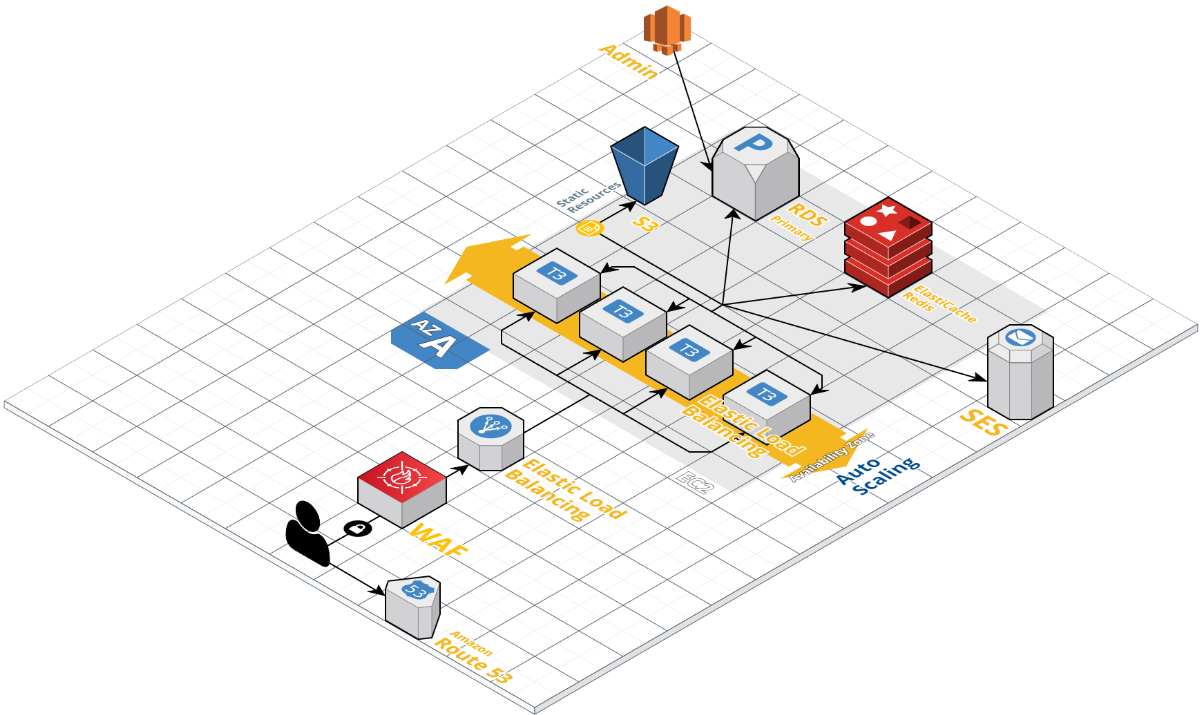


18. ábra API végpontok

# Tesztelés

A fejlesztés során a Postman nevű programmal végeztem az API tesztelését. Ebben a programban egyszerűen lehet HTTP kéréseket küldeni, valamint olvasni azok válaszait. Ezzel a tesztelési móddal könnyedén ki lehet szűrni az esetleges frontend hibákat, azáltal, hogy a Postman-ben küldjük el a kívánt kérést. Ezzel leválasztva a frontend oldalt, és csak a backenddel kommunikációt lehet ellenőrizni.

Továbbá az alkalmazásomat a mesterkeresek.hu vállalkozás jelenleg is nap mint nap használja. A szolgáltatás tesztelése (System Level Testing) az Amazon felhőszolgáltatásában (AWS - Amazon Web Services) történt, ahol maga a mestertkeresek.hu is fut. A teljes rendszer megvalósítását a 19. ábrán lehet látni, ahol a szakdolgozatomban bemutatott alkalmazás „Admin” néven található.



19. ábra A mestertkeresek.hu AWS modellje

Az egység tesztelés a program fejlesztési állapotában fontos, annak érdekében, hogy meggyőződhessünk a megírt programkód helyes-e és valóban az elvártak szerint működik. Ezek eredményességét a tesztlefedettség mutatja százalékos arányban az adott kódra amire az egységet teszteljük. A backend *util*s függvényeinek tesztelésére Unit teszteket hoztam létre, melyek kódja az alábbi.

const utils = require("../utils/index");

describe("tests", () => {

  test("calculate Percentage  2 + 4 should return 100", () => {

    expect(utils.calculatePercentage(2, 4)).toBe(100);

  });

  test("calculate Income should return 1000", () => {

    const data = {

      rows: [

        {

          items: [

            {

              item\_total: 500,

              quantity: 2,

            },

          ],

        },

      ],

    };

    expect(utils.calculateIncome(data)).toBe(1000);

  });

  test("concat obj to sting should return Hello = 'World'", () => {

    const data = {

      Hello: "World",

    };

    expect(utils.concatMyString(data)).toBe(`Hello = 'World'`);

  });

});

A tesztek futtatásához a Jest [27] JavaScript keretrendszerét használtam.

# Telepítés és futtatás

Az alkalmazás futtatásához a Node.js és a PostgreSQL telepítése szükséges, amely a *https://nodejs.org/en/download/* , *illetve a https://www.postgresql.org/download/* oldalakról tölthető le. A Node.js telepítője az ajánlott beállítások mellet telepíti a npm-et (*Node Package Manager*) is.

Az adatbázis létrehozásához a *psql Windows Command Line (SQL Shell)* segítségével vagy egy PostgeSQL-t támogató adatbázis kezelőre van szükség. A *psql Command Line*-ban az alábbi paranccsal tudunk adatbázis létrehozni.

CREATE DATABASE adatbazis\_nev

Ha létrejött az adatbázis, akkor az SQL fájl importálását az alábbi parancs kiadásával lehet megtenni.

psql testdb < filename.sql

Az SQL fájlban lévő adatok dátumokat is tartalmaznak, amelyek az *alap adatok* funkciónál a dátumok elévülése után befolyásolhatják a megjelent adatokat, ugyanis e dátumok alapján vannak kimutatva a mai, heti, havi eladások és bevétel. Ez a program működését nem gátolja, csak az megjelenített kimutatásoknál 0 értékek szerepelhetnek.

A backend első indítása előtt a */api* mappába navigálva egy terminál segítségével az alábbi parancsot szükséges kiadni.

npm install

Az indításhoz */api* mappában az alábbi parancs kiadása szükséges.

nodemon ./index.js

A frontend alkalmazásrés első indításához a */react-client* mappába navigálva egy terminál segítségével az alábbi parancsot szükséges kiadni.

npm install

Az indításhoz  */react-client* mappában az alábbi parancs kiadása szükséges.

npm start

A frontend alapbeállítások mellett a *http://localhost:3000/* címen érhető el.

# Összefoglalás

Szakdolgozatomban a mestertkeresek.hu – szakemberek hirdetéseivel foglalkozó weboldal – adminisztrációs feladatokat ellátó webalkalmazását terveztem és valósítottam meg. Először röviden ismertettem a megoldandó problémákat és a rendszerkövetelményeket, azután példákon keresztül bemutattam az általam készített alkalmazás működését.

A frontend elkészítéséhez a *React.js* és a *Material* könyvtárakat használtam, valamint a backend megvalósítása *Node.js* és az *Express.js* technológiákkal történt. A *PostgreSQL* alapú adatbázis elkészítéséhez a mestertkeresek.hu adatbázisát vettem alapul. A webalkalmazás fejlesztése során a tanulmányaim alatt szerzett tapasztalatokat nagymértékben tudtam hasznosítani, azonban számos új ismeretekre is szert tettem a web technológiák terén.

Úgy vélem egy teljesértékű szoftvert sikerült létrehoznom, amelyet mi sem bizonyít jobban, mint az, hogy a vállalkozás jelenleg is sikeresen használja a szoftvert az ügyintézési folyamatok megkönnyítésére. A program jövőképében szerepelnek olyan funkciók, mint kör-e-mailek küldése, további statisztikai kimutatások és diagrammok megjelenítése, valamint számos más folyamatnak az automatizálása.

# Summary

In my thesis, I established and designed a web-based application of mestertkeresek.hu web page which executes administrative tasks. Firstly, I specified the requirements for the systems and the problems in order to be solved, afterwards I demonstrated the working of the application through examples.

I used the React.js and Material libraries to develope the frontend, and the backend was implemented in Node.js and Express.js technology. I used the database of the mestertkeresek.hu to produce a database based on PostgreSQL. Developing a web-based application was a challenging task while I could practice my experiences acquired during my studies. In addition, I improved and broden my knowledge in web technology. In my opinion, I have developed a well-functioning software, which is used by the mentioned enterprise in order to facilitate the administration process.

In the vision of the future developments, there are functions including sending circular, presentation of statistical summaries, diagrams and automation of many other processes.

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | „RazorSQL,” [Online]. Available: https://razorsql.com/features.html. [Hozzáférés dátuma: 24 11 2021]. |
| [2] | „MarinaDB,” [Online]. Available: https://mariadb.com/kb/en/tableplus/. [Hozzáférés dátuma: 24 11 2021]. |
| [3] | „pgAdmin,” [Online]. Available: https://www.pgadmin.org/faq/#1. |
| [4] | „Barion,” [Online]. Available: https://www.barion.com/hu/fizetesi-kapu/lehetseges-felhasznalasi-szituaciok/. [Hozzáférés dátuma: 24 11 2021]. |
| [5] | „Third Rock Techkno,” [Online]. Available: https://www.thirdrocktechkno.com/blog/single-page-apps-vs-multi-page-apps-what-to-choose-for-web-development/. [Hozzáférés dátuma: 14 11 2021]. |
| [6] | Perfectnova. [Online]. Available: https://perfectnova.hu/tudastar/drotvaz-wireframe. [Hozzáférés dátuma: 26 11 2021]. |
| [7] | T. Rascia, „Log Rocket,” 23 12 2020. [Online]. Available: https://blog.logrocket.com/nodejs-expressjs-postgresql-crud-rest-api-example/. [Hozzáférés dátuma: 20 11 2021]. |
| [8] | R. Peterson, „Guru99,” 7 10 2021. [Online]. Available: https://www.guru99.com/introduction-postgresql.html. |
| [9] | „Oracle,” [Online]. Available: https://www.oracle.com/database/. [Hozzáférés dátuma: 24 11 2021]. |
| [10] | „Microsoft,” [Online]. Available: https://www.microsoft.com/en-us/sql-server/sql-server-2019. [Hozzáférés dátuma: 24 11 2021]. |
| [11] | Gremmédia, „https://gremmedia.hu/,” 05 01 2021. [Online]. Available: https://gremmedia.hu/angular-react-vagy-vue-melyiket-lehet-erdemes-hasznalnunk. |
| [12] | M. Schwarzmüller, „https://academind.com/,” 19 03 2020. [Online]. Available: https://academind.com/tutorials/angular-vs-react-vs-vue-my-thoughts/. |
| [13] | D. Gavigan, „Medium,” 3 04 2018. [Online]. Available: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7. [Hozzáférés dátuma: 24 11 2021]. |
| [14] | „RisingStack,” 11 09 2021. [Online]. Available: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/. [Hozzáférés dátuma: 24 11 2021]. |
| [15] | B. Azeez. [Online]. Available: https://www.ateamsoftsolutions.com/the-story-of-evan-you-the-father-of-vue-js/. [Hozzáférés dátuma: 24 11 2021]. |
| [16] | [Online]. Available: https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html. |
| [17] | S. W. Sapnesh Naik, „https://blog.crowdbotics.com/,” 13 10 2020. [Online]. Available: https://blog.crowdbotics.com/most-compatible-frontend-backend-framework-pairings/. |
| [18] | P. H. Kseniia Kyslova, „https://proxify.io/,” 21 01 2021. [Online]. Available: https://proxify.io/articles/node-and-react. |
| [19] | H. James, 15 12 2017. [Online]. Available: https://haydenjames.io/80-percent-web-powered-by-php/. [Hozzáférés dátuma: 24 11 2021]. |
| [20] | „Codia,” 25 02 2020. [Online]. Available: https://www.codica.com/blog/pros-and-cons-of-ruby-on-rails-for-web-development/. [Hozzáférés dátuma: 24 11 2021]. |
| [21] | P. Pedamkar, „Educba,” [Online]. Available: https://www.educba.com/laravel-vs-ruby-on-rails/. |
| [22] | A. Yushkevych, „Monovm,” 15 06 2021. [Online]. Available: https://monovm.com/blog/backend-development-how-to-choose-the-right-framework/. |
| [23] | „Coreycleary,” [Online]. Available: https://www.coreycleary.me/project-structure-for-an-express-rest-api-when-there-is-no-standard-way. [Hozzáférés dátuma: 20 11 2021]. |
| [24] | „Technostacks,” 21 08 2021. [Online]. Available: https://technostacks.com/blog/react-component-libraries. |
| [25] | „KnowledgeHut,” 17 03 2021. [Online]. Available: https://www.knowledgehut.com/blog/web-development/axios-in-react. |
| [26] | GeeksforGeeks, 04 05 2021. [Online]. Available: https://www.geeksforgeeks.org/difference-between-pern-and-mern-stack/. |
| [27] | Jest. [Online]. Available: https://jestjs.io/. [Hozzáférés dátuma: 25 11 2021]. |