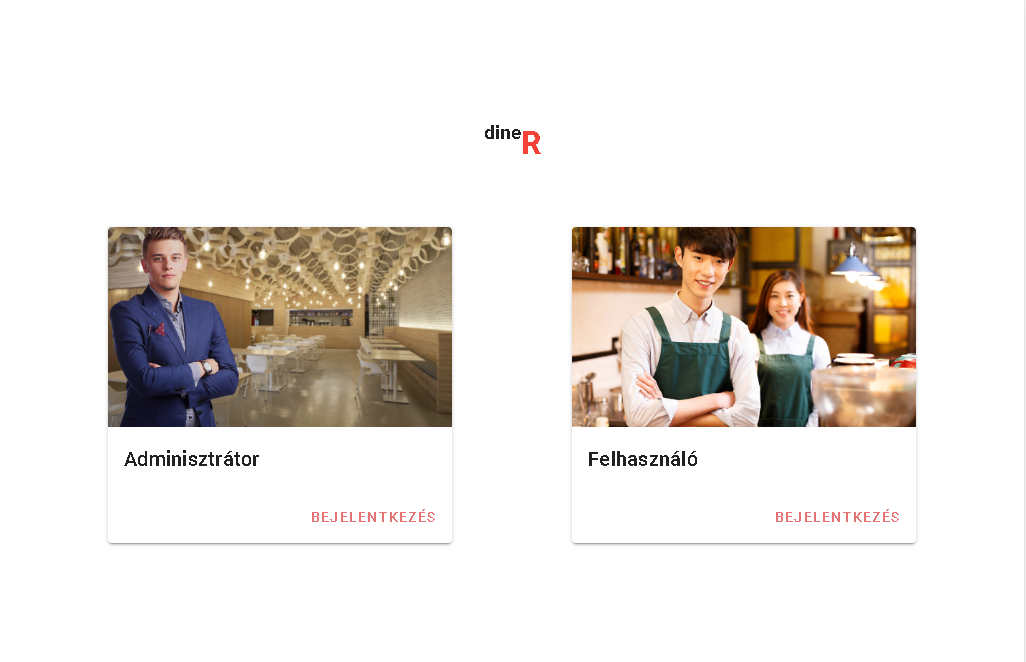
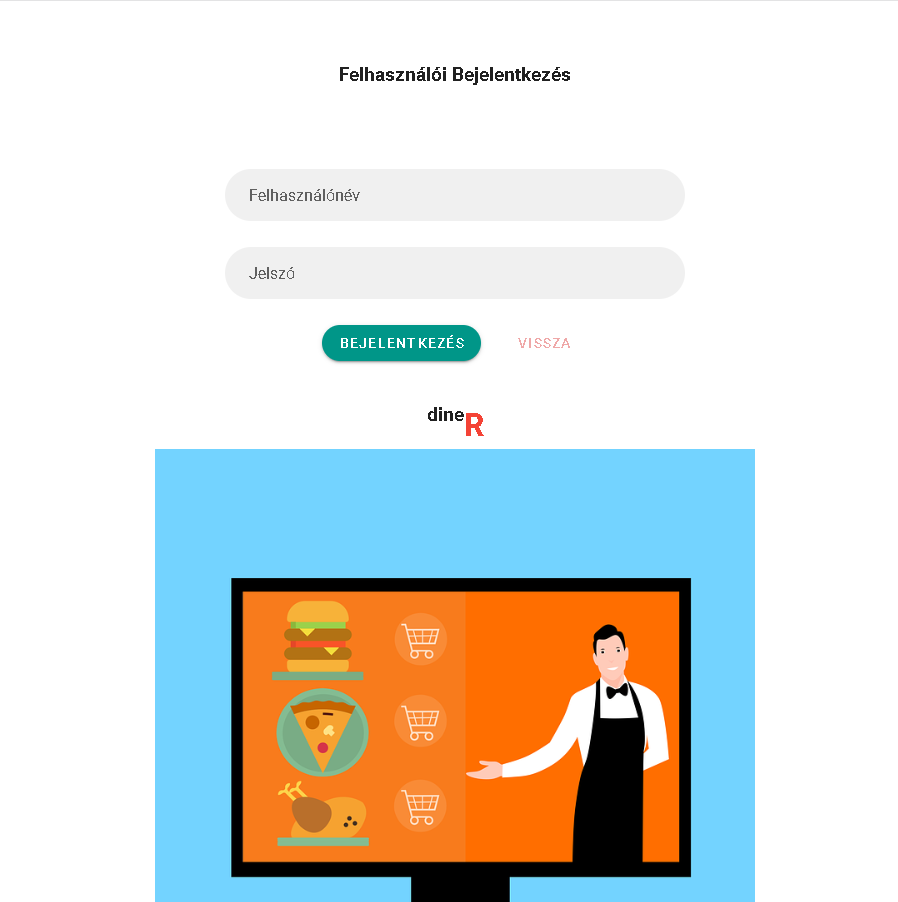
# Dokumentáció: **dineR**

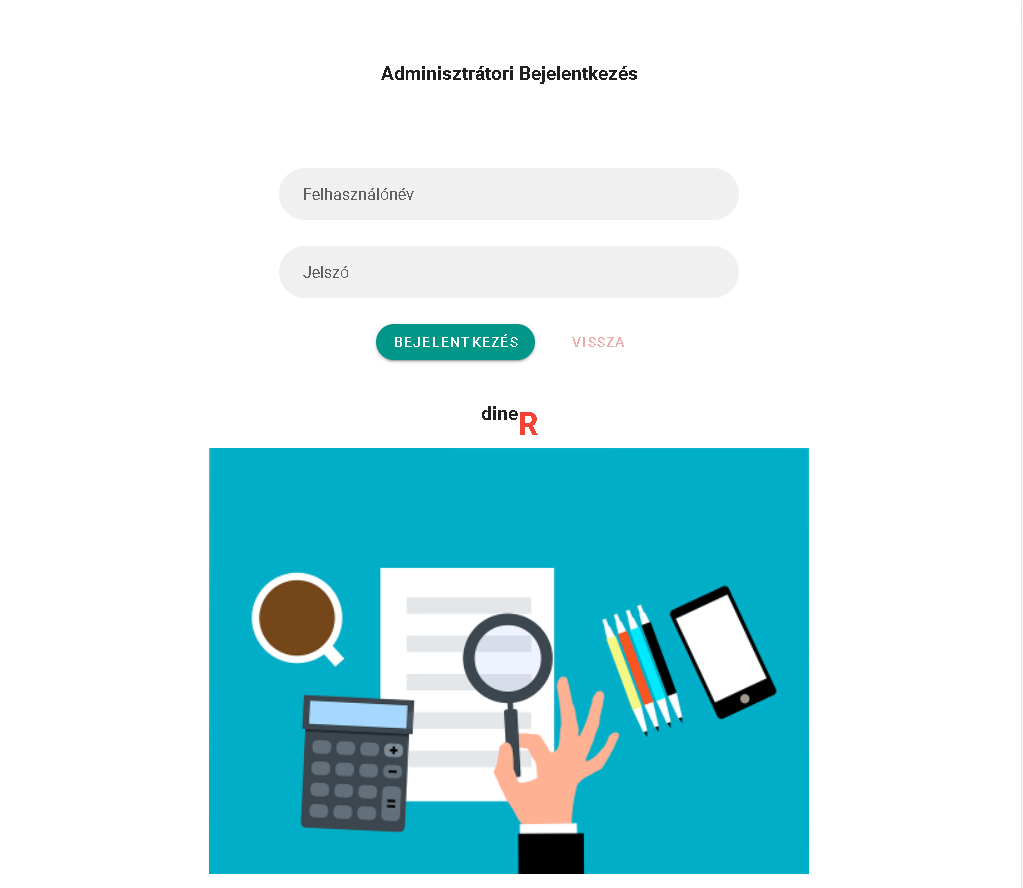
## Bemutatkozás

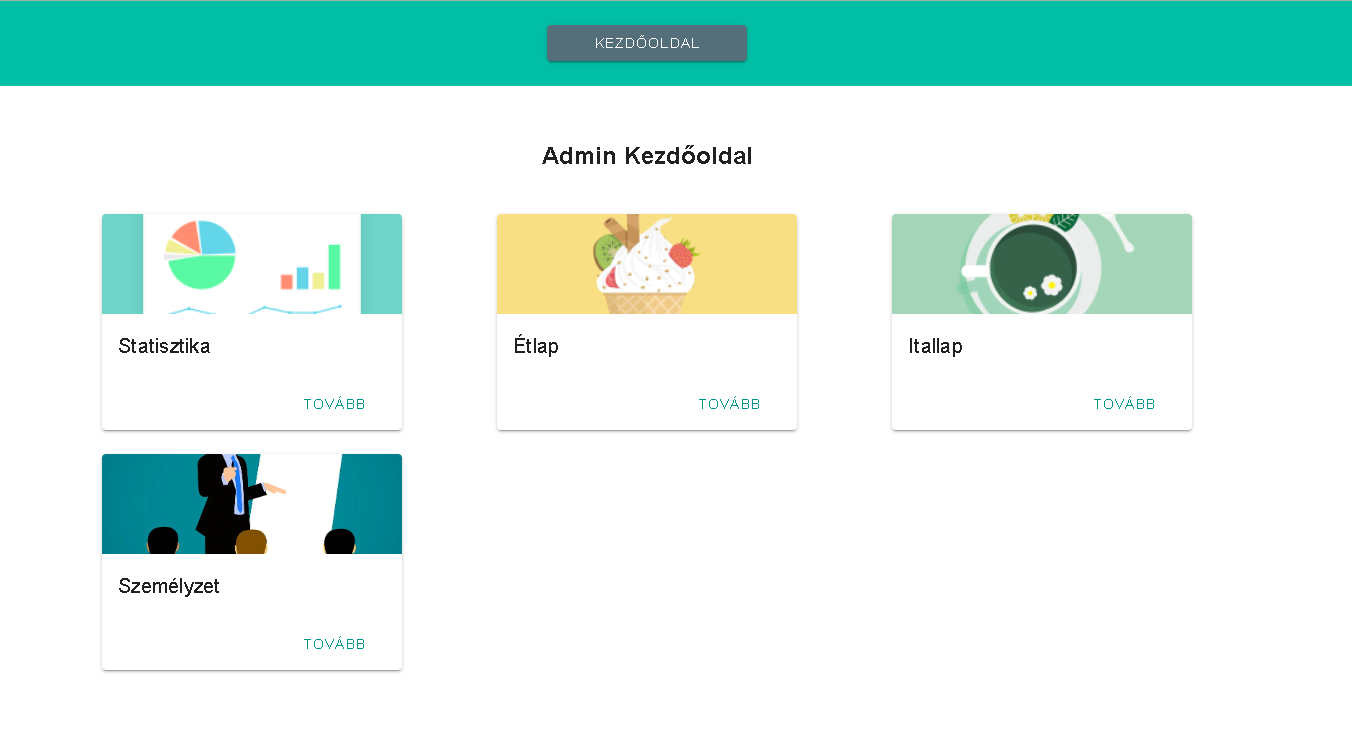
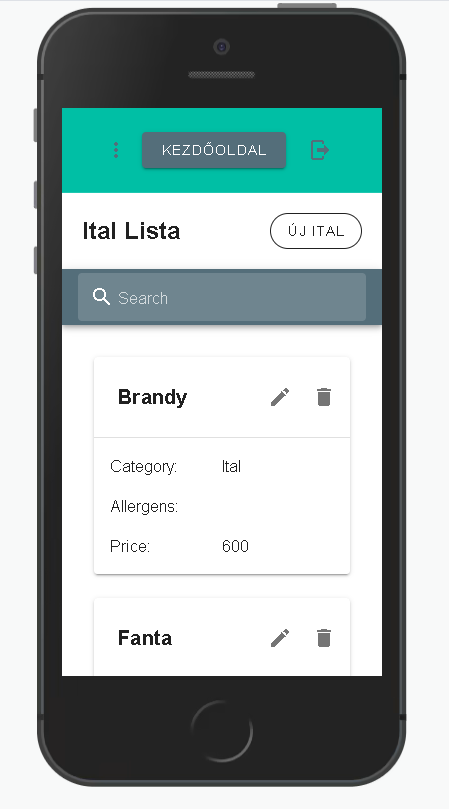
A dineR applikáció az olyan vendéglátóiparban tevékenykedő kisebb éttermek csoportját célozza meg, ahol még nem találtak informatikai megoldást a mindennapi feladatok szimulációjára. Különösképp a felszolgálókra és az ő feladatkörükre fókuszáltam az applikáció tervezése és írása közben. A cél, hogy a hagyományos rendelésfelvételt felváltsa egy valamelyest innovatív megoldás, amely rugalmasan tud alkalmazkodni fejlett életviteleinkhez.

## Admin nézetek

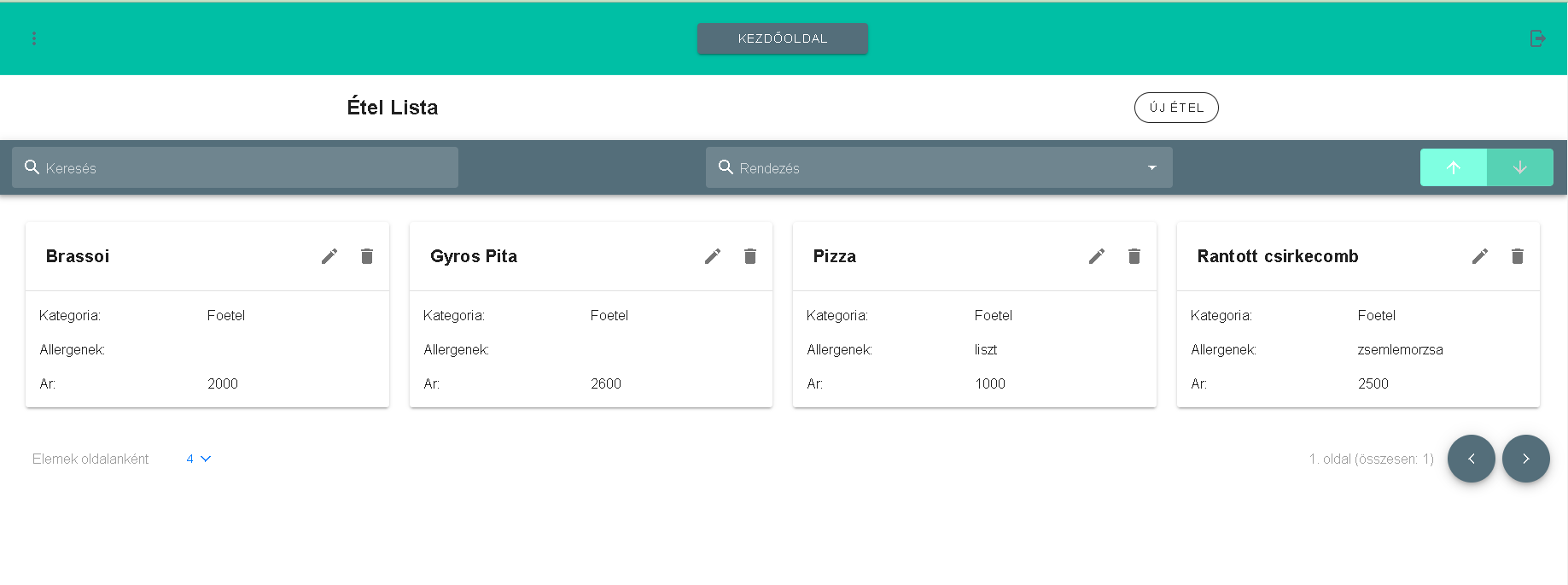
Noha az applikáció központi témája a felszolgálás és rendelésfelvétel, helyet kapnak az alapvető adminisztrációs és menedzsment feladatok is. Ez magába foglalja az ételek, italok és munkaerő erőforrás karbantartását az informatikában úgynevezett CRUD operációkon keresztül. Ennek jelentése nem más, mint az adatok létrehozása, olvasása, szerkesztése valamint törlése. Ezen feladatok az én megvalósításomban az adminisztrátori részhez tartoznak. A két feladat mind szerkezetileg mind megvalósításban eléggé elütnek egymástól, így érdemesnek tűnt az applikációt két részre osztani. Az egyik rész, amelyik a menedzseri CRUD operációkat tartalmazza, külön jogosultságot és bejelentkezést igényel, ezért az ilyen jellegű felhasználó adminként fogja tudni használni és 'konfigurálni' az applikációt. A konfigurálás alatt jelen esetben nem mást értünk, mint azon ételek, italok, személyek létrehozását, szerkesztését, törlését, amelyek és akik által az ügyfeleket tudjuk kiszolgálni. A másik csoport fogja elvégezni a felszolgálással kapcsolatos tevékenységeket. Ilyen értelemben gondolhatunk rendelésfelvételre, felszolgálásra, asztalok és székek tisztántartására, információnyújtásra, számla kiállítására, és még sok más feladatra.

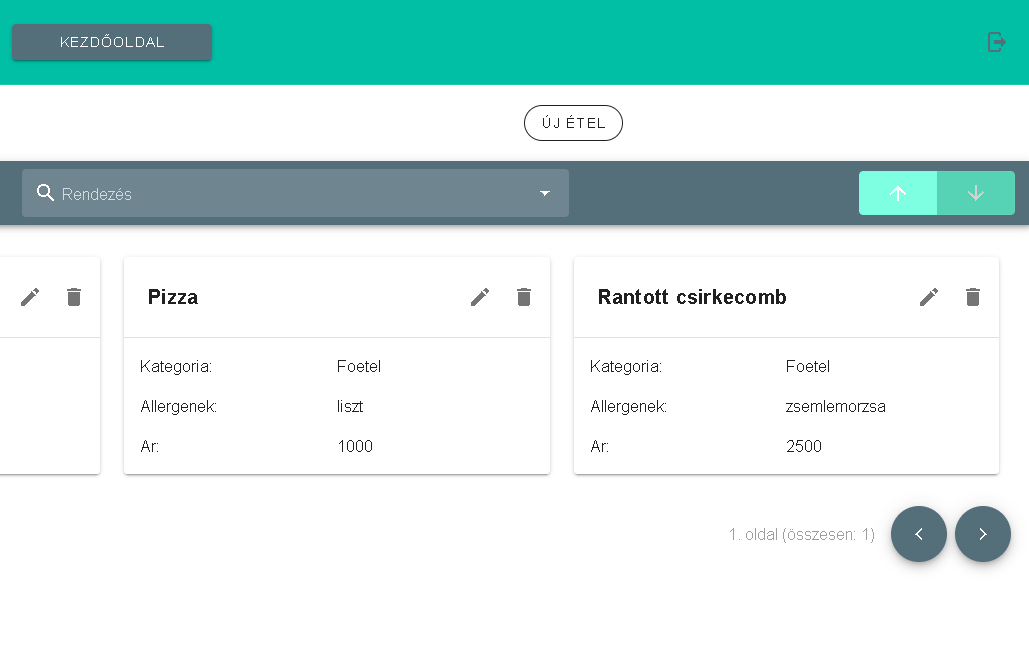
Felhasználói azonosításra két bejelentkezési felületet tettem lehetővé. Kezőoldalon tud választani a felhasználó, hogy melyik felületre szeretne bejelentkezni. Természetesen ezekre a felületekre csak akkor tud bejelentkezni, ha a számára létrehozott felhasználó rendelkezik a megfelelő jogosultsággal. Az autentikálás felhasználó és jelszó megadásával történik. Az applikáció fel fogja ismerni a bejelentkezési kísérlet alapján, hogy a felhasználó miként szeretne bejelentkezni. Az applikáció biztosítja a megfelelő útvonalat a felhasználónak, ugyanakkor nem teszi kizárttá, hogy a felhasználó más útvonalat kövessen. Ezt igyekeztem útvonal védelemmel megelőzni, így ösztönözve a felhasználót, hogy hagyja magát vezetni az applikáció által.



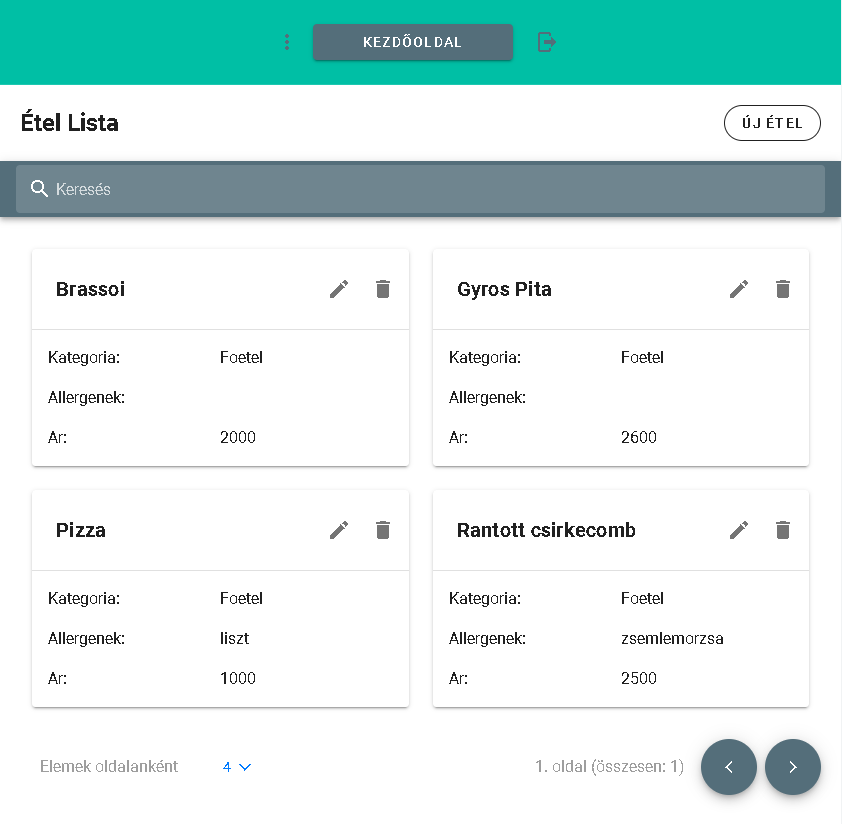
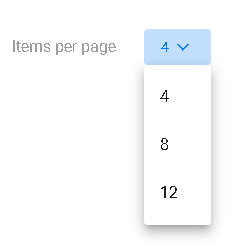
Az admin felhasználókat bejelentkezés után egy navigációs főképernyő fogadja. Innen további 4 oldal elérhető, melyek többnyire a CRUD tevékenységek végrehajtására lettek létrehozva. Pontosan 3 oldalon fogja tudni az adminisztrátor elvégezni a 4 fő adatkezelést. Külön kiemelhetjük a 3 CRUD oldalból a Dolgozók adattábla szerkesztésére szolgáló oldalt, ahol az adminisztrátor további adminisztrátorokat vagy felhasználókat hozhat létre. Felhasználóként fogok a továbbiakban hivatkozni a felszolgálókra. A 3 CRUD aloldal hasonló dizájnt és szerkezetet követ, kisebb változásokkal. Az adminisztrátori kezdőoldalról közvetlenül eljuthatunk az aloldalak listanézetébe.

A hagyományos tábla nézetet mellőztem, mivel napjainkban egyre nagyobb szerepet kap a mobiltelefon, és az ezáltal kialakult 'mobile first' dizájn, amit én is próbálok követni. Ennek célja, hogy elsősorban mobilra fejlesztünk / tervezünk, azt vesszük alapként, majd odafigyelünk arra is, hogy nagyobb képernyőkőn egyaránt stilusosan nézzen ki a programunk.



A mobile first dizájnbarát GRID elrendezésre esett a választásom az adatok listázásában. Egy jól elkészitett GRID minden platformon megállja a helyét. A mi esetünkben ez kártyákból épül fel, amelyeket mobilon stack dizájnban tudunk prezentálni, valamint  kibőviteni 3x3 vagy 4x4 vagy akár nagyobb nézetre, amelyek már megfelelőbbek a nagyobb képernyőkre. A kártyák sorrendjét tudjuk befolyásolni, valamint keresni is tudunk a kártyák között, ami nagyban megkönnyiti a szerkesztés lehetőségét nagyobb adathalmazokban.

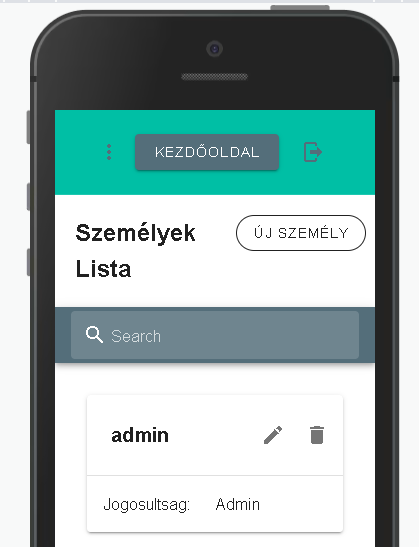
A rendezés alapját tudjuk változtatni, erre az ételek adathalmazban van a legtöbb, mig a személyek adathalmaz esetén van a legkevesebb szabadságunk. Továbbá lehetőségünk nyilik arra is, hogy az alapértelmezettnél több adatot jelenitsünk meg a képernyőn, valamint ehhez kapcsolódóan a lapozást is implementáltam.



A kártyák tartalmazzák a szerkesztés és törlés gyorsgombját. Törlés esetén azonnali reakciót kapunk mind a frontenden mind a backenden. Az ételek adattáblában lehetőségünk nyilik az alábbi mezők szerkesztésére:

* név
* kategória
* allergének
* ár

Ugyanezek a lehetőségek előttünk állnak italok szerkesztésénél is, viszont a tervezés úgy igényelte, hogy az italok adathalmaz ne kapjon külön táblát backenden. Ott ugyanis ugyanolyan entitásként kezeljük, mint az ételeket, a szeparáció csak a frontendben történik meg. A személyek adathalmaz lista nézetét korlátozva érjük el, több információ érhető el, ha az egyes kártyáknál a szerkesztés gombra kattintunk. Igy ugyanis minden mezője látszani fog, és természetesen szerkeszthető is lesz.

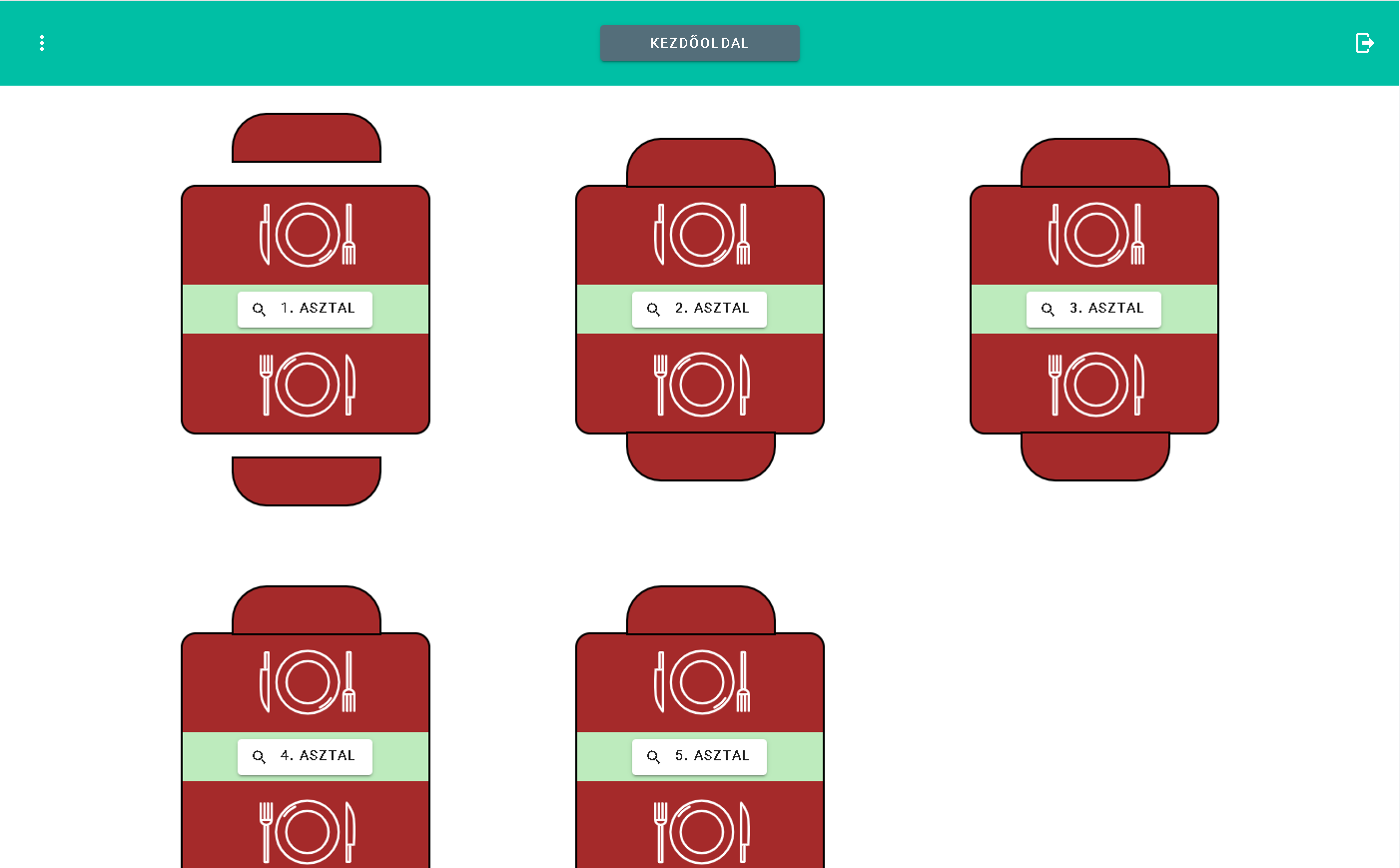
Létrehozásra a listanézetek felső

részében elhelyezett gombbal tudunk. Főleg a létrehozásnál aktuális, de a szerkesztésnél is implementálásra került frontend validálás.

Ez az a folyamat, amikor leellenőrizzük, hogy a beviteli adatok megfelelnek-e bizonyos kritériumoknak, mielőtt további feldolgozásra küldjük a backend felé.

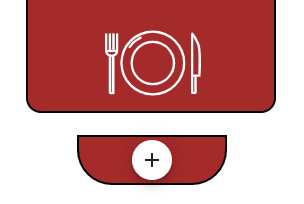
Validálás szempontjából nagyon fontos, hogy egyedi felhasználóneveket használjunk, ugyanis ez problémát okozhat az applikáció futásakor. Kötelezővé tettem néhány mezőt, valamint minimum hosszúságot is beállitottam. Az adattáblák mellett statisztikai adatok megtekintését teszi lehetővé a negyedik főmenü, melynek két alkalmazását teszem lehetővé. Egyrészt az adminisztrátoroknak, akik havi nézetet kapnak a legnépszerűbb ételekről, havi forgalomról, akár napi lebontásban, másrészt a felhasználóknak akik a napi forgalmon kívül vendégek és rendelések pillanatnyi állapotát tudják majd a nézeten elérni.

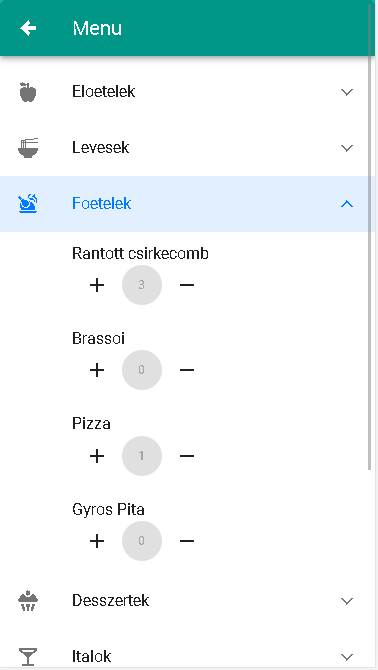
## Felhasználói nézetek



A felhasználói nézetek fő témája az asztalok, székek, rendelések állapota. A dineR egy folyamatot próbál meg szimulálni egy könnyen kezelhető applikáción keresztül. A szimuláció célja, hogy a vendégek érkezésének és távozásának körforgását hatékonyan és életszerűen vissza tudja adni a nézeteken keresztül. Ezt a legegyszerűbben konkrét példán lehet szemléltetni a következő illusztrációkkal.:

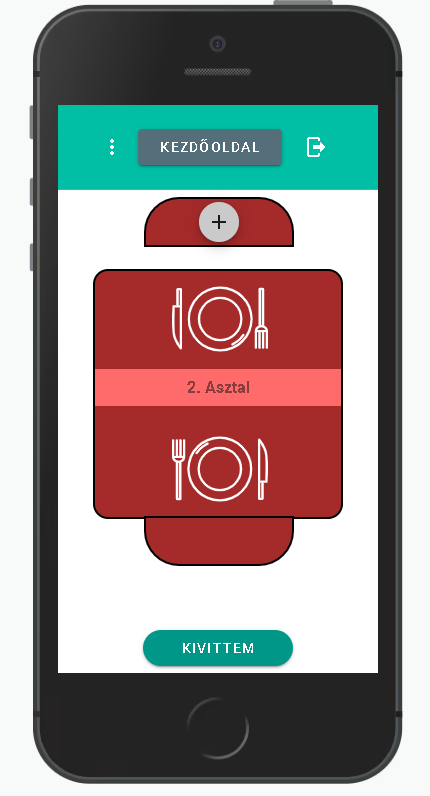
A **vendég érkezése** (a mi esetünkben ez annyit jelent, hogy egy szék állapotát megváltoztatjuk, betolt állapotról kihúzott állapotra)



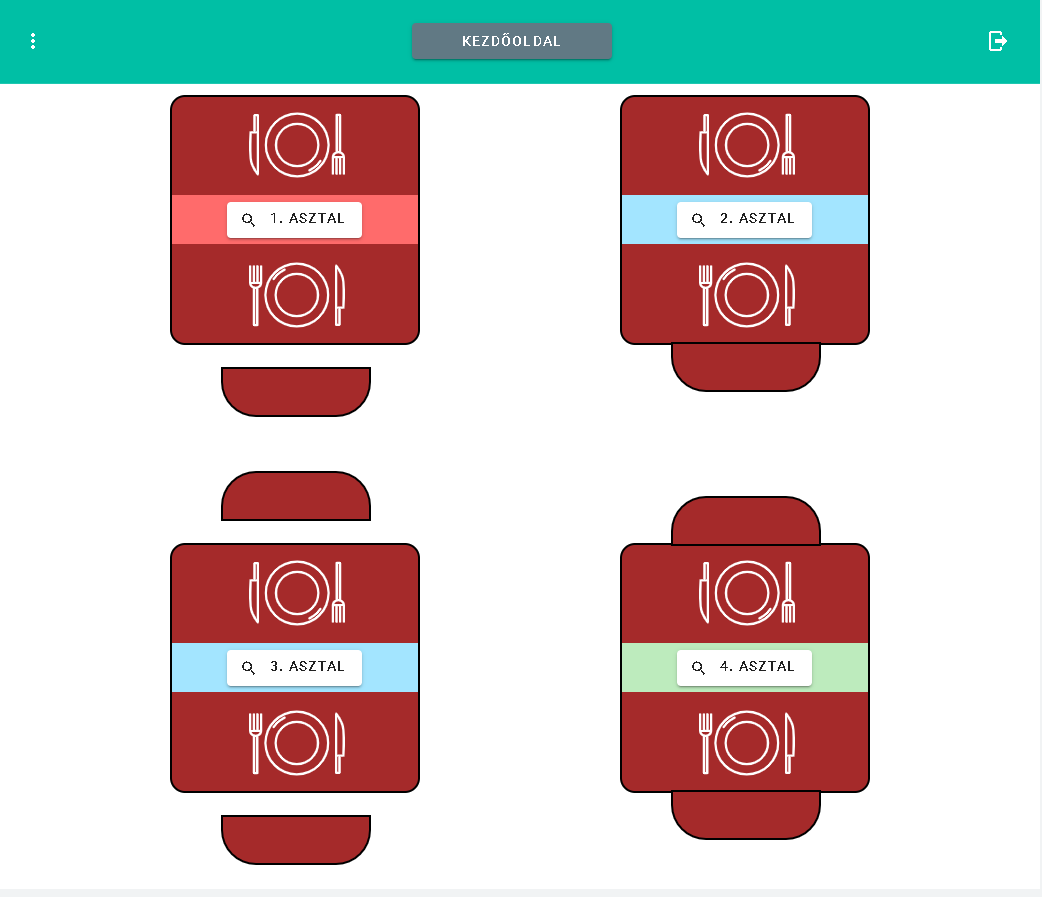
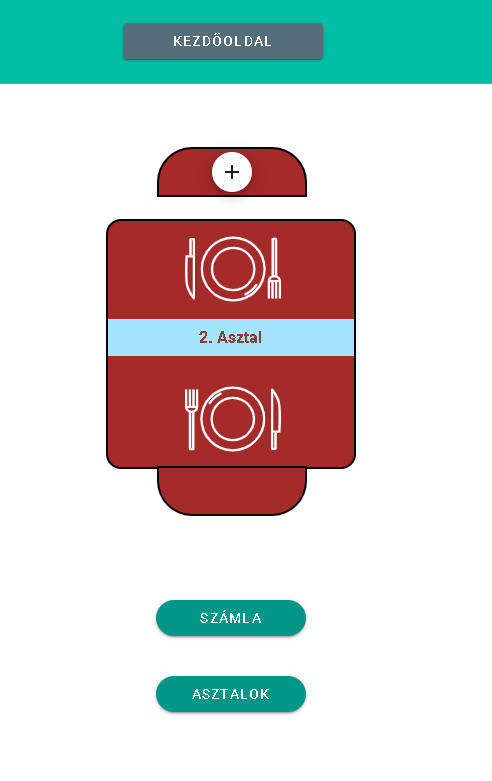
**Rendelés összeállítása** a menülapon keresztül (megjeleníthetjük tehát a menüt, és kiválaszthatjuk a kívánt ételeket, italokat, mintha mi magunk lennénk a vendégek)

Láthatjuk, hogy az összeválogatás közben **folyamatos visszajelzést** kapunk arról, hogy mi lesz a várható számlaösszeg.

**Rendelés leadás** (a menü megjelenítésével és rendelé s összeállításával ellentétben, a rendelés leadás már hivatalos megrendelésnek számít, ebből készül később a statisztika az adminisztrátori és felhasználói statisztika nézetekben.

Az applikációban használjuk ehhez a **Rendelés leadás** gombot. Figyeljünk meg két dolgot, egy, hogy az asztal terítője zöldről vörösre változik, ezzel szemléltetvén, hogy a vendégünk várakozik az ételére. Kettő pedig, hogy megjelenik egy gomb Kivittem névvel. Ez egy dinamikus gomb, szövege és viselkedése változik.)

**Rendelés kivitel** (az étel elkészült, így vihetjük a várakozó vendégünknek, majd ha ezt megtettük, használhatjuk az imént említett **Kivittem** gombot, ezzel jelezve, hogy egyelőre nincs teendőnk, hagyhatjuk a vendéget étkezni. Itt ismét felhívnám a figyelmet, hogy a terítő színe ismét megváltozott, ezúttal beszédes kék színre, a dinamikus gomb pedig a következő szöveget vette fel: Számla)

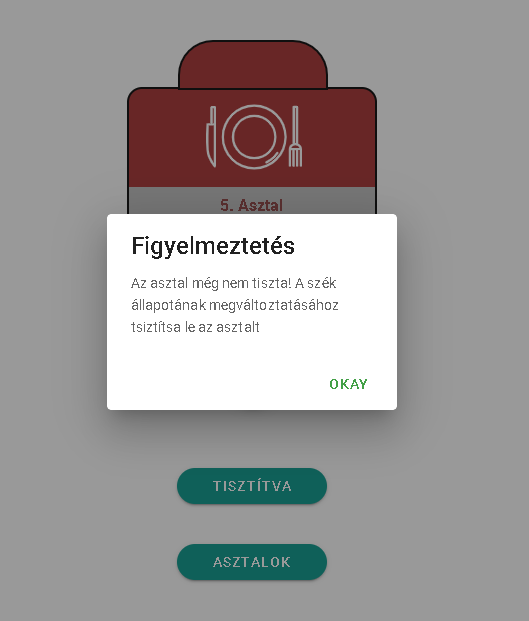


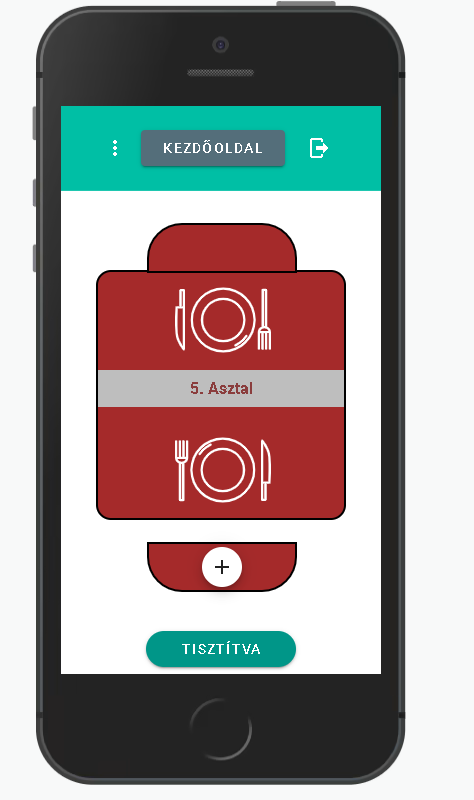
A vendég készen áll fizetni, megtehetjük, hogy a **Számla** gombot használatba vesszük, hogy megkapjuk a számla információt. Fontos megjegyezni, hogy amíg a szék állapota nyitva van, bármikor dönthet úgy a vendég, hogy további ételt vagy italt rendel, ilyenkor a második ponthoz visszakanyarodva újra használhatjuk a menü funkcionalitást, leadhatunk egy rendelést, a terítő vörösre fog változni, a dinamikus gombunk pedig **Kivittem**-re, jelezvén újra a várakozást a vendég részéről.

A **számla megnyitása** után szimulálhatjuk számla kiegyenlítést a Számla Rendezve gombbal.



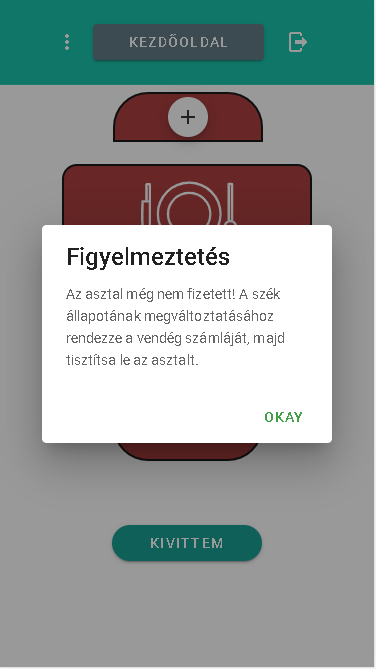
Ezt használva a terítő szürke színűvé fog változni, amivel az asztal takarításra szorulását próbáltam igyekezni.



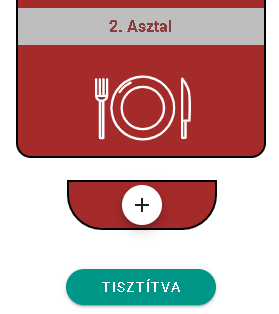
Ennél a pontnál érdemes lehet megismerkednünk a **lezárt szék** helyi kifejezéssel. A lényege a következő:

A szimuláció gondolatmenete, hogy a betolt szék tiszta asztalt von maga után, készen állva a következő vendégre. Ebből kiindulva vannak olyan helyzetek, mikor nem tolhatjuk be, vagy húzhatjuk ki a széket.

Ilyen eset a szimulációnkban amikor egy asztal még nincs letakarítva, ezt a szürke színű asztalterítő jelzi, ami lezárt székeket eredményez, nem tudjuk ezeket a székeket mozgatni mindaddig, amíg az asztal nincs letakarítva.

Hasonlóképpen nem tudunk olyan széket mozgatni a szimulációban, ahol van leadott rendelés. A székeket úgy programoztam a szimulációban, hogy egy olyan asztalnál, ahol már rendeltek, legalább az egyik széknek nyitva kell maradni mindaddig, amíg nem fizetett és nincs letakarítva.

Ahhoz, hogy befejezzük a kört, használjuk a Tisztítva dinamikus vezérlő gombot, amely gondoskodni fog róla, hogy az asztal státusza ismét megfelelő legyen vendégek fogadására. Ezt a zöld terítőnk fogja jelezni, valamint a két betolt szék, amik így idő közben elveszítették a zárt szék státuszukat.



## Adatbázis szerkezet:

TÁBLA: asztalok

MEZŐK:

id [INT]

alsoSzekFoglalt [BOOL]

alsoSzekZarolva [BOOL]

felsoSzekFoglalt [BOOL]

felsoSzekZarolva [BOOL]

rendelesAllapot [STRING]

aktivRendeles [INT]

MAGYARÁZAT:

Az asztalok és székek folyamatos állapot követése. Boolean értékekkel tájékozódunk arról, hogy egy-egy szék jelenleg foglalt-e, vagy zárolva van-e. Az asztalok rendelkeznek rendelés állapottal. Az applikációban 4 állapotot használok ehhez: **idle, ordered, delivered, paid**

TÁBLA: rendeles

MEZŐK:

id [INT]

asztalAzonosito [INT FOREIGN KEY asztalok.id]

letrehozva[DATE]

fizetve[BOOL]

visszavonva[BOOL]

MAGYARÁZAT:

A rendelések rögzítése statisztikai és állapotkövetési célokból szükséges. Egy-egy rendelés egyedi azonosítóval rendelkezik, amihez tartozhat egy vagy több étel vagy italrendelés (rendelésrészlet)

TÁBLA: rendelesreszlet

MEZŐK:

id [INT]

rendelesAzonosito [INT FOREIGN KEY rendeles.id]

asztalAzonosito [INT FOREIGN KEY asztalok.id]

etelAzonosito [INT FOREIGN KEY etlap.id]

etelDarabszam [INT]

MAGYARÁZAT:

Mivel a rendelések több szakaszban is érkezhetnek egy asztalnál, a rendelésrészlet lehetőséget biztosít, hogy ezeket hatékonyan eltároljuk az adatbázisban.

TÁBLA: etlap

MEZŐK:

id [INT]

nev [STRING]

kategoria [STRING]

allergenek [STRING]

ar [INT]

MAGYARÁZAT:

Az étlap táblán keresztül tesszük lehetővé az adminisztrátornak az étlap kezelést, valamint innen tápláljuk be az adatokat a dineR applikációba.

TÁBLA: felhasznalok

MEZŐK:

id [INT]

felhasznalonev [STRING]

jelszo [STRING]

vezeteknev [STRING]

keresztnev [STRING]

admin [BOOL]

MAGYARÁZAT:

A felhasználók táblában hozzuk létre az adminisztrátorokat és felhasználókat, valamint ezen tábla segítségével tudnak majd bejelentkezni az applikációba.

MEGJEGYZÉS:

Ezen leírás csak felületes szemléltetése az adatbázisnak. Minden részlet megtalálható az adatbázis exportban.

Az applikáció backenden Node JS & MySQL , frontenden Vue JS segítségével készült.

Front-end dependenciák (exportált package.json file):

{

"name": "restaurantapp",

"version": "0.1.0",

"private": true,

"scripts": {

"serve": "vue-cli-service serve",

"build": "vue-cli-service build",

"lint": "vue-cli-service lint"

},

"dependencies": {

"axios": "^0.26.1",

"core-js": "^3.6.5",

"vee-validate": "^4.5.11",

"vue": "^2.6.11",

"vue-router": "^3.2.0",

"vuetify": "^2.6.0",

"vuex": "^3.4.0"

},

"devDependencies": {

"@vue/cli-plugin-babel": "~4.5.12",

"@vue/cli-plugin-eslint": "~4.5.12",

"@vue/cli-plugin-router": "~4.5.12",

"@vue/cli-plugin-vuex": "~4.5.12",

"@vue/cli-service": "~4.5.12",

"babel-eslint": "^10.1.0",

"eslint": "^6.7.2",

"eslint-plugin-vue": "^6.2.2",

"sass": "~1.32.0",

"sass-loader": "^10.0.0",

"vue-cli-plugin-vuetify": "~2.4.8",

"vue-template-compiler": "^2.6.11",

"vuetify-loader": "^1.7.0"

},

"eslintConfig": {

"root": true,

"env": {

"node": true

},

"extends": [

"plugin:vue/essential",

"eslint:recommended"

],

"parserOptions": {

"parser": "babel-eslint"

},

"rules": {}

},

"browserslist": [

"> 1%",

"last 2 versions",

"not dead"

]

}

Backend dependenciák (exportált package.json file):

{

"name": "restaurantapi",

"version": "1.0.0",

"description": "API for restaurant application",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "SzokeSandor",

"license": "ISC",

"dependencies": {

"body-parser": "^1.20.0",

"cors": "^2.8.5",

"express": "^4.17.3",

"mysql": "^2.18.1"

}

}