Ceglédi SZC Közgazdasági és Informatikai Technikum

Szoftverfejlesztő és -tesztelő

5 0613 1203

Fejlesztői dokumentáció a „Döner Cegléd” alkalmazáshoz

Jakab Botond, Krausz Márton Alajos, Tóth Dániel János

Cegléd

2025

# Tartalom

[Tartalom 2](#_Toc194044460)

[**Asztali alkalmazás dokumentáció** 4](#_Toc194044461)

[Feladat/cél 4](#_Toc194044462)

[Használt programok, fejlesztői környezetek és futtatási környezet 7](#_Toc194044463)

[Futtatási környezet és hardware 8](#_Toc194044464)

[A program használata 10](#_Toc194044465)

[**Backend** 20](#_Toc194044466)

[Feladat/Cél 20](#_Toc194044467)

[Használt fejlesztési környezet, programok és csomagok 20](#_Toc194044468)

[File struktúra 21](#_Toc194044469)

[Generált dokumentáció 21](#_Toc194044470)

[A dokumentáció elérése 21](#_Toc194044471)

[**Frontend (weboldal)** 25](#_Toc194044472)

[Feladat/Cél 25](#_Toc194044473)

[Használt fejlesztési környezet, keretrendszerek és csomagok 25](#_Toc194044474)

[Kommunikáció a backendel 26](#_Toc194044475)

[Példa adatlekérésre 27](#_Toc194044476)

[Példa adat küldésre 29](#_Toc194044477)

[**Dokumentáció a backend tesztekről** 31](#_Toc194044478)

[Feladat/Cél 31](#_Toc194044479)

[Környezetleírás 31](#_Toc194044480)

[A fájl felépítése 31](#_Toc194044481)

[Használat 31](#_Toc194044482)

[Fontosabb tesztek bemutatása 32](#_Toc194044483)

[Middlewares tesztek 32](#_Toc194044484)

[Error Handler tesztek 32](#_Toc194044485)

[Repository tesztek 32](#_Toc194044486)

[User Repository tesztek 32](#_Toc194044487)

[Dish Repository tesztek 33](#_Toc194044488)

[Controller tesztek 34](#_Toc194044489)

[User Controller tesztek 34](#_Toc194044490)

[Dish Controller 35](#_Toc194044491)

[Források 36](#_Toc194044492)

# **Adatbázis adatmodellek**

még meg lesz csinálva

# **Asztali alkalmazás**

## Feladat/cél

Az alkalmazás célja az éttermi működés hatékonyabbá tétele, az adatok gyors elérése és módosítása, valamint a felhasználók és rendelések egyszerű kezelése. A felhasználóbarát felület és a valós idejű kommunikáció biztosítja a gördülékeny munkafolyamatokat az étterem minden területén.

Rendelések megjelenítése és kiadása

* Az alkalmazás valós időben jeleníti meg az étteremhez beérkező rendeléseket.
* Lehetőség van a rendelések részletes megtekintésére, beleértve a termékek listáját, a mennyiségeket, az árakat és az esetleges megjegyzéseket.
* A rendelés kiadása után az állapot frissíthető az API-n keresztül.

Regisztrált felhasználók kezelése

* Az alkalmazás lehetőséget biztosít a regisztrált felhasználók kezelésére.
* A felhasználók deaktiválhatók, ha már nem aktívak az étteremben.
* A felhasználói adatok, például a név, e-mail cím, szerepkör vagy jogosultságok szerkeszthetők.

Új termékek felvitele

* Új termékek hozzáadása egyszerű és gyors folyamat.
* A következő adatok rögzíthetők az új termékhez:
  + Termék neve
  + Ár
  + Lehetséges módosítási opciók (pl. extra feltétek, szószok)
  + Termék kategóriája (pl. Wrap, Kebab, Drink, SideDish)
  + Termék képe feltölthető PNG formátumban
* Az új termékek mentése az API-n keresztül történik.

Már meglévő termékek teljes körű szerkesztése

* Az alkalmazás támogatja a termékek teljes körű szerkesztését.
* A meglévő termékek adatai bármikor módosíthatók, beleértve:
  + Nevet
  + Árat
  + Módosítási lehetőségeket
  + Kategóriát
  + Termékképet
* Lehetőség van a termékek archiválására vagy inaktiválására, ha már nem elérhetők.

Felület minta

A képen szöveg, menü, Számítógép-monitor, fedett pályás látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

## 

## Kommunikáció a backenddel és a használt osztályok

## Használt programok, fejlesztői környezetek és futtatási környezet

Használt fejlesztői környezetek





A fejlesztéshez használt fejlesztői környezet

Letöltés:

<https://visualstudio.microsoft.com/downloads/>



### Futtatási környezet és hardware

Windows 11-es PC 32GB memóriával

Fejlesztői környezet: Visual Studio 2022 Community Edition

Processzor: AMD Ryzen 5600X

Memória: 2X16GB 3200Mhz DDR4 ram

Tárhely: 1TB NVME SSD

GPU: Nvidia RTX 3060Ti

Megjelenítés: 1920\*1080

Egér: Szükséges

Billentyűzet: Szükséges

# **Backend**

## Feladat/Cél

A backend célja egy REST API készítése volt, amit az express csomag használatával valósítottunk meg. Az API a háttérben fut és végpontok segítségével végez el különféle műveleteket az Asztali alkalmazás és a weboldal számára. Ezek a műveletek adatbázissal kapcsolatosak. A backend az adatbázisunkkal 4 fő műveletet tud végrehajtani, adat lekérés, adat feltöltés, adat módosítás vagy adat törlés. Az adatbázis megvalósításához a MySQL-t választottuk, az adatbázis modelleket, kapcsolatot és egyebeket a sequelize csomag használatával backenden valósítottuk meg.

## Használt fejlesztési környezet, programok és csomagok

Fejlesztési környezet: Visual Studio Code

Használt programok:

* futtatáshoz: XAMPP
* teszteléshez: Postman

Használt (npm) csomagok:

* Express
* Sequelize
* jsonwebtoken (JWT)
* bcrypt
* nodemon
* dotenv
* cors
* body-parser
* cookie-parser
* nodemailer
* swagger-autogen

## File struktúra

A file struktúrát tekintve különböző szintekhez kirendezve dolgozunk. A programunk a server.js file indításával működtethető, ez elindítja az app.js-t, amiben a routes mappa fájljai hívódnak meg. A különböző útvonalak hívásai mellé esetleges middleware file tartozik, majd egy funkció, ami az adott controllerben van definiálva. A controllerekből a service fájlok hívodnak meg, amik végül a repository fileokat hívják meg. Mindegyik szint a saját feladatát ellátva oldja meg az egységes működést.

## Generált dokumentáció

A dokumentációt a swagger nevű csomag segítségével valósítottuk meg, ami az összes végpontra részletes leírást biztosít. A dokumentáció minden végpontra tartalmazza a bekért paramétereket (amennyiben vannak) és a kimeneti értékeket (amennyiben vannak), mindkettőt azonos formátumban, mint amit a program fejlesztéséhez használtunk.

## A dokumentáció elérése

A dokumentációt a backend futtatása után az alábbi URL-en érhetjük el, bármelyik böngészővel: localhost:3000/api-docs

A képen képernyőkép, Multimédiás szoftver, szöveg látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Amennyiben sikeresen elindítottuk a backendet és helyesen beírtuk az oldal címét az alábbi felület fogad minket:

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A sorok lenyitásával részletes leírást kaphatunk az adott végpontról:

A képen szöveg, képernyőkép, szoftver, képernyő látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

# **Frontend (weboldal)**

## Feladat/Cél

A weboldal célja egy felhasználóbarátként kialakított rendelőfelület, ahol a felhasználó regisztráció és bejelentkezés után különféle ételeket tud kiválasztani és módosítani kínálatunkból, amelyeket a virtuális kosarába rakva később megrendelhet. Vásárlásaival pontokat gyűjthet, amelyeket későbbi vásárlásoknál beválthat. Módosíthatja különböző adatait, illetve megtekintheti és újra rendelheti előző vásárlásait. A megvalósításhoz Vue js-t használtunk, mint keretrendszer és a mellé társuló Vueitfy-t designoláshoz.

## Használt fejlesztési környezet, keretrendszerek és csomagok

Fejlesztési környezet: Visual Studio Code

Használt keretrendszerek:

* Vue.js
* Vuetify

Használt (npm) csomagok:

* Axios
* TanStack

## Kommunikáció a backendel

A backendel való kommunikáció az Axios segítségével történik, a hívások számának csökkentése érdekében pedig a TanStack Query-t használjuk az adatok cache-elésére. A redundáns sorok elkerülése érdekében az Axios kliens létrehozáskor beállítjuk az opciók értékeit, így azok minden híváskor automatikusan érvényesülnek és nem szükséges őket ismételten konfigurálni.

A képen szöveg, képernyőkép, szoftver, Operációs rendszer látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

## Példa adatlekérésre

Az alábbi kódrészlettel a menün lévő ételek információit kérjük le egy Axios hívás segítségével, majd cacheljük a TanStack Query használatával, végül megjelenítjük az oldalon.

Először létrehozunk egy típust, a lekért étel adataival.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Az étel képének mérete miatt a hívás kap kiegészítő fejléceket, hogy az oldal probléma nélkül megkapja a nagyobb képeket is és meg tudja jeleníteni őket.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A függvényt ezután meghívjuk a kívánt oldalon.



A kapott adatot a Vue beépített „v-for” ciklusának segítségével fogjuk megjeleníteni.

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Eredmény:

A képen képernyőkép, Konyha, szöveg, étel látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

## Példa adat küldésre

Az alábbi kód részlettel oldjuk meg az új felhasználók regisztrációját, egy Axios post hívást követően megadjuk a végpontot, majd átküldjük a regisztrációs adatot.

Először létrehozunk egy típust a regisztráció adataival.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Megírjuk az Axios függvényt, ami elküldi az adatokat.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Meghívjuk a függvényt és létrehozunk egy állapotváltozót, amiben tároljuk majd elküldjük az adatokat.



A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A kellő adatoknak bemeneti mezőket helyezünk el, amikor változáskor frissítik az állapotváltozónk értékét.

A regisztráció gombhoz hozzáadunk egy függvényt, ami a regisztrációs adatot várja paraméterben.



Használjuk az Axiosos mutációnkat, ami elküldi az adatot a backend részére.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Ha minden sikeres volt, átkerülünk egy másik oldalra és értesítést kapunk a sikeres regisztrációról. Amennyiben valami hiba történt, ha kapunk válaszul hiba üzenetet a backendtől, akkor azt jelenítjük meg. Amennyiben viszont más hiba történt, aminek okát nem tudjuk, egy alap hibaüzenetet fogunk megjeleníteni a felhasználónak.

# **Backend tesztek**

## Feladat/Cél

Ez a dokumentáció részlet a main.test.js teszt fájlt mutatja be, amely a projekt backend működésének menetét teszteli, evvel biztosítva a funkciók hibamentes működését, és megfelelően kezelje a hibákat és hibaüzeneteket.

## Környezetleírás

A teszt futtatása közben használt környezet:

* Windows 10 Operációs Rendszer
* Visual Studio Code 1.98.2-as verzió
* Supertest a tesztelés lefuttatásához

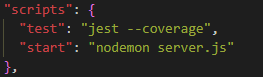
## A fájl felépítése

A main.test.js fájl struktúrája következő féleképpen néz ki:

* Backend tesztek:
  + Controller tesztek: (Például Dish Controller teszt)
  + Middleware tesztek: (Például errorHandler teszt)
  + Repository tesztek: (Például Dish Repository teszt)

## Használat

A tesztek lefuttatása elött szükségünk van néhány előkészületre:

* A helyes Node csomag letöltése, ebben az esetben supertest és jest, ezt megtehetjük a lefuttatni kívánt gyökér mappában, amely ebben a projektben “backend” néven szól.  
  A Terminált használva a megadott úton a “npm i supertest” kódot használva telepítjük a supertest-et:  
  
* A jest beállítása alapértelmezett tesztelő keretrendszernek, amelyet a “package.json” fájlban tehetünk meg a “scripts” mezőnél:  
    
  A ”--coverage” kóddal megadjuk, hogy minden teszt lefutása után készüljön egy kódlefedettségi jelentés.
* A teszt lefuttatása az említett lépések után egyszerűen megtehető a terminálban vagy konzolban a “npm test” kóddal:  
  

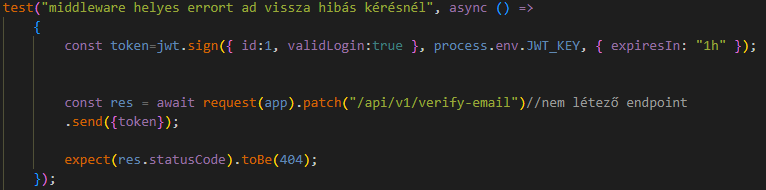
## Fontosabb tesztek bemutatása

### Middlewares tesztek

A Middlewares tesztek fő célja a hibák helyesen tovább küldése, hogy a felhasználó vagy fejlesztő felismerje a hiba eredetét.

### Error Handler tesztek

Az Error Handler, ahogy a neve is sugallja, a hibákat kezeli és küldi azokat tovább a kód biztonsága és működése érdekében, példa erre:

Ebben a példában megpróbálunk egy Controller tesztet lefuttatni, azonban egy nem létező végpontra, ami miatt hibába ütközik a folyamat és 404-es státuszkódot, hibakódot várunk vissza.

Ezt a tesztet azért kell külön “köztes réteggel” (middleware-el) letesztelni, mível a controllerben nem tudjuk azt az esetet letesztelni, ha rossz a végpont, mert nem éri el, ezért az errort tovább küldjük és visszaadja a middleware.

### Repository tesztek

A repository, “tároló” egységek tesztelésének a célja az adattárolás működésének és hibakezelésének biztosítása, emellett biztosítja, hogy az előforduló hibák megelőzhetőek legyenek és stabil maradjon a kód.

### User Repository tesztek

A User Repository tesztek főbb célja a felhasználóval kapcsolatos funkciók kezelését biztosítja, példa erre:

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Ebben a példában leteszteljük a Felhasználó létrehozását és annak lekérését.  
Előre beégetett adatokat használunk, így könyebben ráláthatunk a lehetséges hibákra.

### Dish Repository tesztek

A Dish Repository tesztek főbb célja a fogásokkal, ételekkel kapcsolatos funkciók kezelését biztosítja, példa erre:

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Kettő példa látható, az első példában leteszteljük a fogás létrehozását és annak lekérését.  
Előre beégetett adatokat használunk, így könnyebben ráláthatunk a lehetséges hibákra.

A második példában az összes leteszteljük az összes fogás lekérését és elvárjuk, hogy a visszakapott hossz 1 legyen, mivel az előző tesztben készítettünk egyet.

### Controller tesztek

A vezérlőrétegen végzett tesztek biztosítják a kód működését és a kérések, válaszok helyességét, emellett a bemeneti adatokat is vizsgáljuk és több esetre figyelünk tesztelés során.

### User Controller tesztek

A User Controller tesztekkel biztosítjuk, hogy a felhasználóhoz kapcsolódó kérések és válaszok helyes értékkel térnek vissza és hiba nélkül mennek végbe, példa erre:

Ebben a példában kettő tesztet láthatunk, amelyek a felhasználó létrehozását kezelik, az egyiknél helyes választ várunk, míg a másiknál helytelent. Az első tesztben az ”/api/v1/register” végponton a kérés testében beküldjük a létrehozáshoz, regisztráláshoz elvárt szükséges adatokat, ami után elvárjuk, hogy a válasz státuszkódja a helyzetnek megfelelően helyesen térjen vissza.  
A második tesztben ugyanazt a kérést teszteljük, azonban ebben az esetben egy szándékosan hibás kérést küldünk be (hiányzó adat) és várjuk az adott helyzethez megfelelő választ.

### Dish Controller

A Dish Controller tesztekkel biztosítjuk, hogy a fogásokhoz kapcsolódó kérések és válaszok helyes értékkel térnek vissza és hiba nélkül mennek végbe, példa erre:

A képen szöveg, képernyőkép, Betűtípus, képernyő látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Ebben a példában egy fogás, étel létrehozását láthatjuk, ahol helyes értéket várunk vissza.  
Az ”/api/v1/dish” végponton a kérés testében beküldjük a létrehozáshoz elvárt szükséges adatokat.

# Források

McDonalds rendelés felület (inspiráció)

Postman

Xampp

Visual studio code

Microsoft Visual studio

CocaCola hu (termék fotók)

Leonardo.ai (generált termék fotók)