Hódmezővásárhelyi SZC

Szentesi Pollák Antal Technikum

VIZSGAREMEK

Készítette:

- Karikó-Tóth Tibor
- Sipos Gabriella
- Szabics Tibor

Szentes, 2025

Tartalomjegyzék

Bevezető **3**
Miért éttermi asztalfoglaló rendszer? 3
Mi a célunk? 3
1. Fejlesztői dokumentáció **3**
1.1. Fejlesztői környezet 4
1.2 Github és Git környezet 6
1.3 A program általános áttekintése 7
2. Adatszerkezet, a program felépítése **8**
2.1 Frontend (Vue 3 + Vite) szerkezet 9
2.2. Backend (Node.js + Express + Prisma) szerkezet 10
2.2.1. Frontend ↔ Backend kommunikáció 10
2.2.2. Összegzés 11
2.3. A Backend 11
2.3.1. Inicializáció és Hitelesítés (Index.js) 12
2.3.2. API Útvonalak (Controller Layer) 14
2.3.3. Adatelérési Részletek (Service és Schema) 14
2.3.4. Kapcsolatok (package.json) 14
2.4. A Frontend 15
2.4.1. A fő elemek 15
2.4.2. Az oldalkomponensek és az adatkapcsolatok 15
2.5 Teszt dokumentáció 19
2.5.1. Backend tesztek: 20
2.5.2. Frontend tesztek: 23
2.5.3. Teszt eredmények: 25
3. Felhasználói dokumentáció **25**
3.1 Navigáció a weboldalon és a legfontosabb funkciók (felső menüsor) 27
3.2 Hitelesítés és Profil Kezelése 27
3.3 Asztalfoglalás 28
3.4 Vélemények Írása 29
3.5 Hibaelhárítás 30

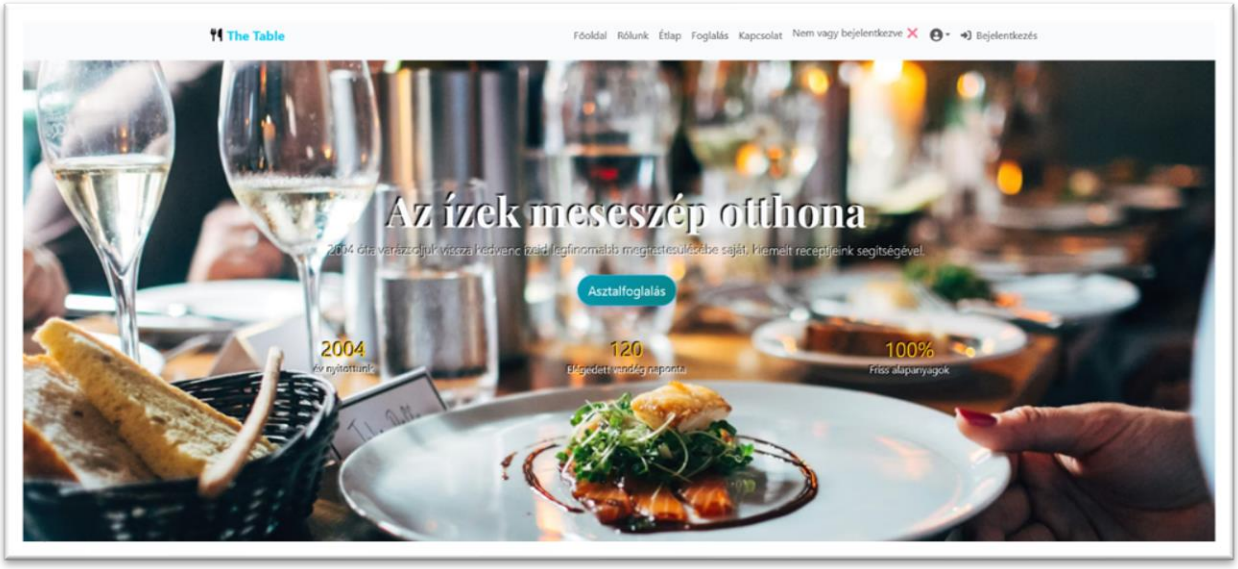
Bevezető

Miért éttermi asztalfoglaló rendszer?

Az éttermi asztalfoglaló rendszer webprojekt közös ötletelés eredményeképp jött létre. Tudjuk, hogy az interneten sokféle hasonló rendszer működik. Ugyanakkor ezen oldalak esetében többször futottunk bele különböző akadályokba, az átlagos felhasználó számára nehezen kezelhető vagy nem jól áttekinthető megoldásokba. Mi egy olyan gyakorlati lehetőségként tekintettünk a feladatra, amelyben megmutathatjuk az elsajátított ismeretek felhasználási lehetőségeit. Mintha kezdő vállalkozásként megbízást kaptunk volna egy étterem saját webes rendszerének megalkotására a logó és dizájnelemektől egészen a jól strukturált adatbáziskezelésig.

Mi a célunk?

Egy olyan letisztult látványvilágú és egyben jól használható weboldalt akartunk létrehozni, amely segítségével könnyen és kényelmesen lehet asztalt foglalni az étteremben. A biztonságos adatkezelést felmutató rendszer gyors és világos visszajelzést ad a felhasználónak a foglalás részleteiről. Az alkalmazásunk segítségével elkerülhető a felesleges várakozás, bosszankodás. Reményeink szerint egy jól használható alternatívát tudunk kínálni a piacon lévő hasonló fejlesztések mellett.



1. Fejlesztői dokumentáció

1.1. Fejlesztői környezet

Az általunk elkészített vizsgaremek három nagy egységre osztható:

- frontend,
- backend programelemek
- html weblap

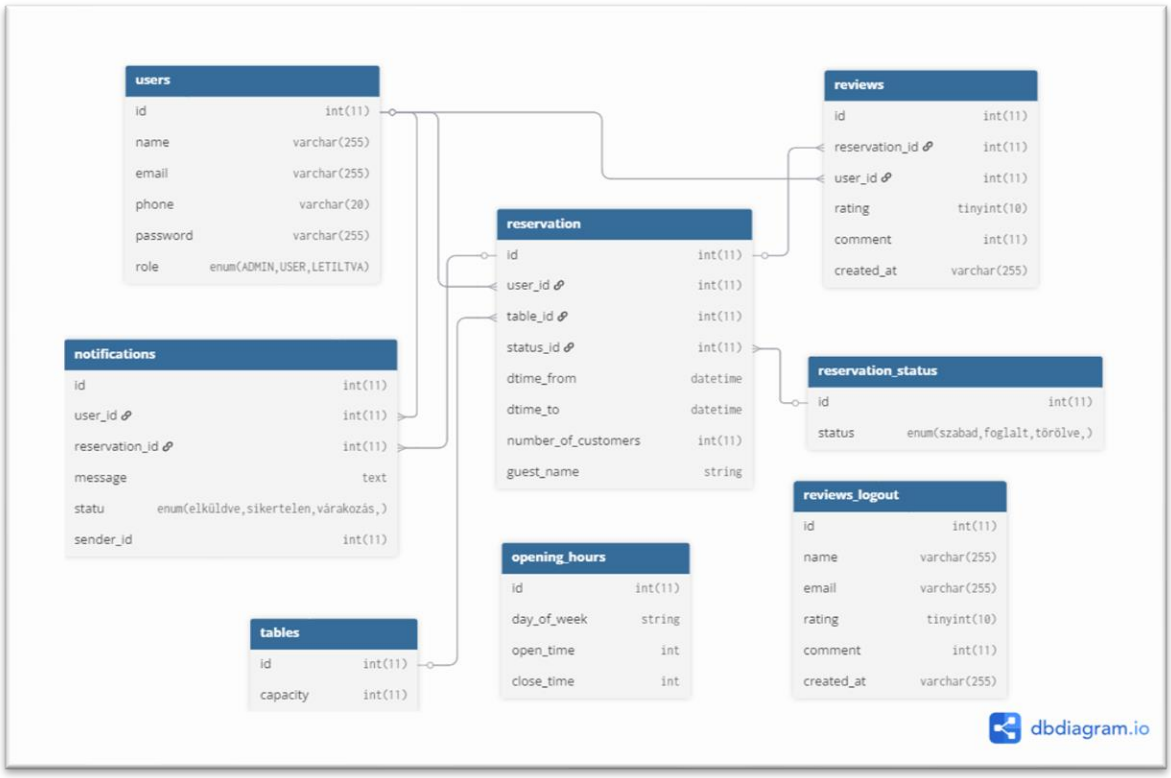
Elsődlegesen azokat a fejlesztőeszközöket használtuk, amelyekben kellő jártasságot szereztünk a képzés során. Fontos szempont volt, hogy ezek a freeware programok ingyenek és széles körben használatosak; bőséges leírással, tapasztalattal.

A Visual Studio Code program hatékonyan segítette a weboldal kialakításához szükséges programelemek megírását, kódolását.

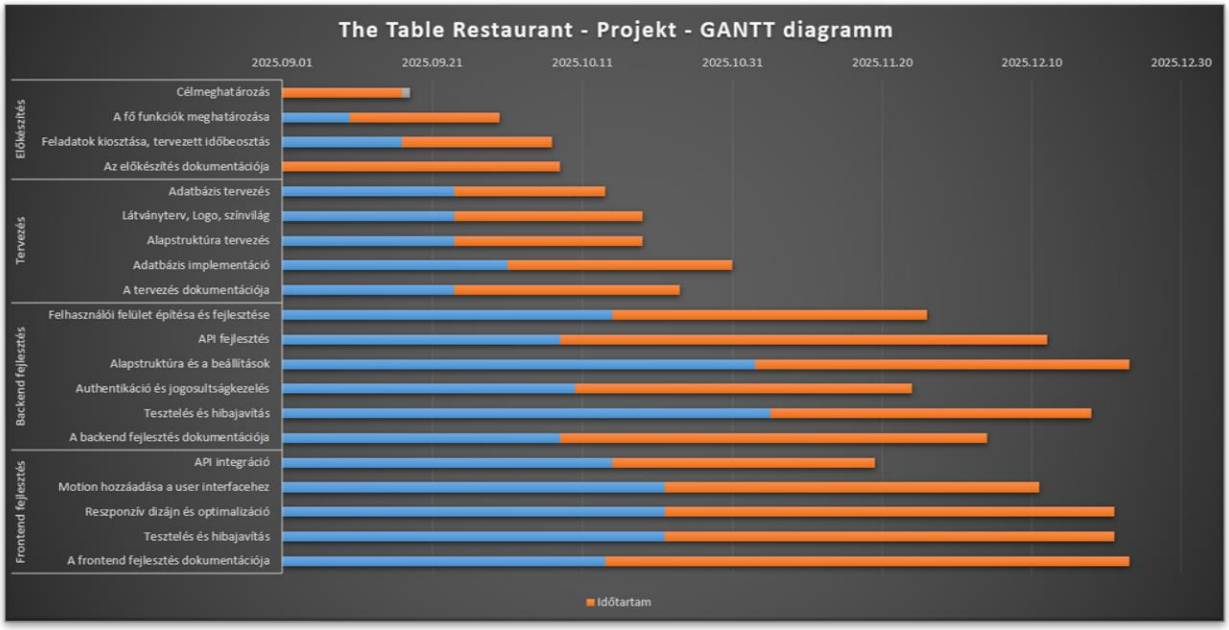
A XAMPP, mivel egyben tartalmazza az Apache webkiszolgálót, a MySQL adatbáziskezelőt és a PHP-t, biztosította, hogy valós webkiszolgáló nélkül offline módban tudjuk elvégezni a fejlesztést és a szükséges tesztelést.

A MySQL a létrehozott adatbázis kezelésében és folyamatos naprakészen tartásában jelentett segítséget.

Ezek mellé társultak a látványelemek (logó, képek, színvilág) létrehozásában a különböző grafikus programok.



A The Table adatbázis szerkezete.



A GANTT diagramm, projekttervező idővonal.

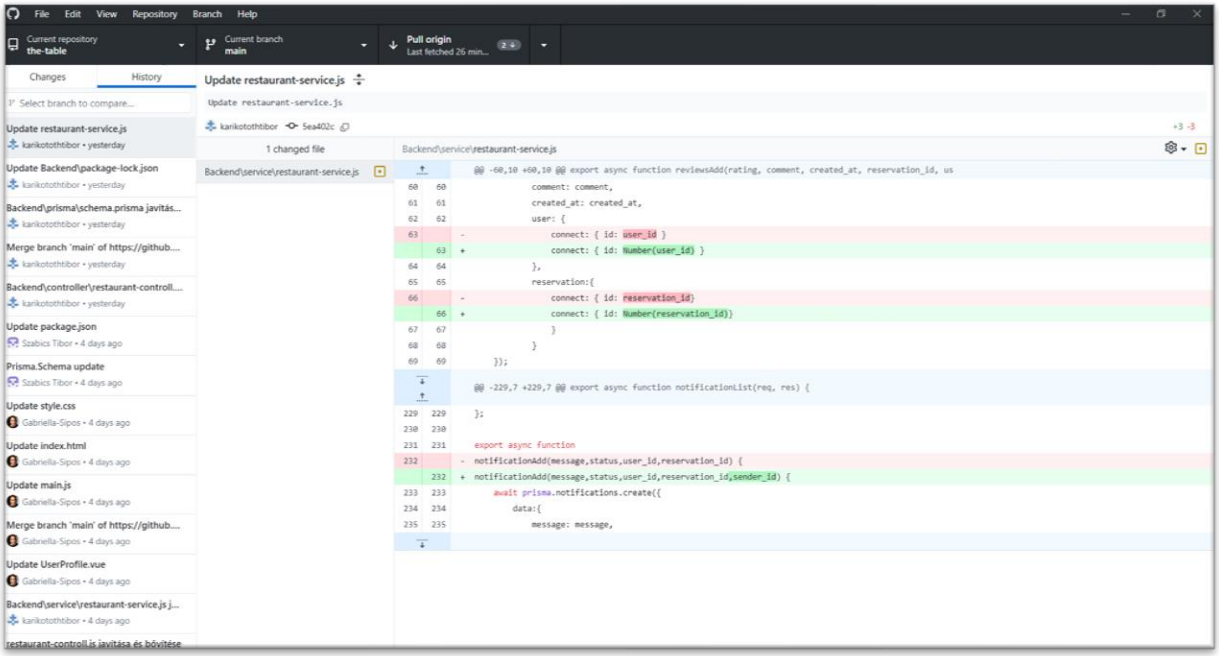
1.2 Github és Git környezet

A feladat komplexitása és az előírások miatt szükséges volt egy közös fejlesztői felület létrehozása és működtetése. Erre a GitHub bizonyult megfelelő opciónak. Mindhárman regisztráltunk a felületre, majd egyikünk létrehozta admin jogosultsággal a projektet. (GitHub/the-table).

Ezt követően minden tag meghívást kapott a közös projekt felületre. Így létrejött mindenki saját accountja, így a programfejlesztés lépései mindenki számára átláthatóvá, könnyen követhetővé váltak. A fejlesztéshez kapcsolódó egyéb fájlok (pl. dokumentáció) szintén közös elérhetőséget kaptak.

A saját gépeken feltelepített egyéni környezet (GitHub desktop) és a felhőtárhely között kiépült a megfelelő kapcsolat.

Az elkészült projekt elemeiként megtalálhatóak a közös felületen a programkódok, az adatbázis és az ehhez kapcsolódó leírás, dokumentáció. Ezzel vált teljes körűvé a The Table projekt.



1.3 A program általános áttekintése

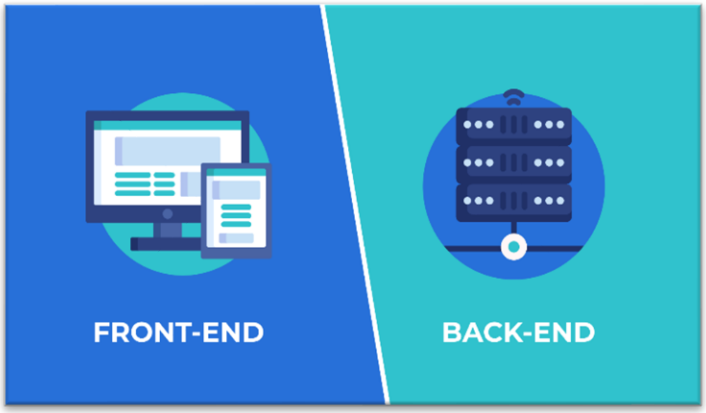
Hogyan működik a The Table Étterem webes alkalmazása?

Tekintsük úgy a programot, mint egy **éttermet teljes személyzettel**, ahol mindenki más feladatért felel.

1. Az étterem: a teljes alkalmazás

A program két fő részből áll, amelyek állandóan beszélgetnek egymással:

- **Az étterem belső tere** (Frontend / Vue.js): Ez az amit a felhasználó a böngészőjében lát. Ez a **felhasználói felület**, ami fogadja a kéréseket (kattintások, űrlapok).
- **A konyha és az iroda** (Backend / Node.js): Amit a felhasználó nem lát. Ez dolgozza fel az érkező kéréseket (pl. "foglalj asztalt"), ellenőrzi az adatokat és tárolja azokat.



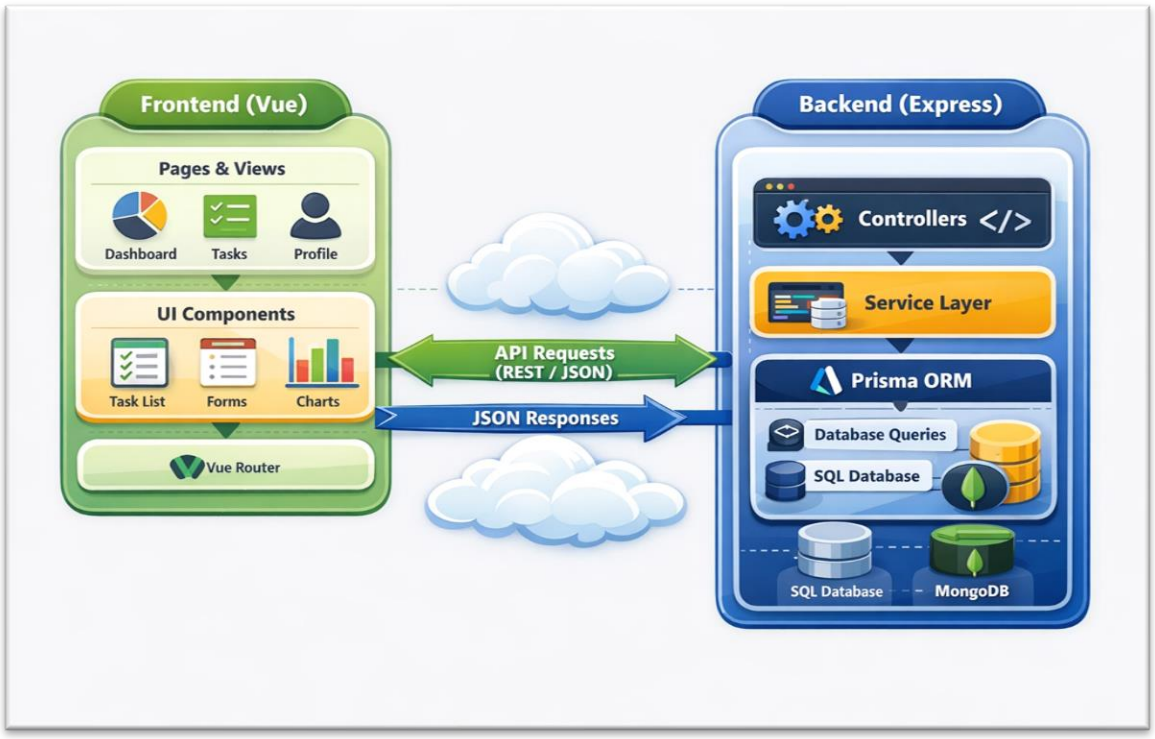
Az étterem belső tere (Frontend / Vue.js Fájlok)

A Vue.js fájlok határozzák meg a látványt és az interakciót:

Feladat	Funkció	Kód
Bemutatja az étterem általános képét és egy listát az eddigi vendégek véleményeiből (/review).	Főoldal Néhány vélemény	Home.vue
Az asztalfoglaláshoz szükséges adatok kitöltése. Innen megy át a kérés a Konyhába (/reservationadd).	Asztalfoglalás	Reservation.vue
Eldönti, hogy be van-e jelentkezve. Ha igen, bemutatja a Profilját; ha nem, akkor a Bejelentkezési űrlapra vált. (AuthForm.vue).	Recepció / Bejárat	Login.vue

Bemutatja az étterem kínálatát. előételek, főételek, desszertek, köretek, saláták, italok (Az ételek esetében jelzi vegánilletve laktózmentes lehetőségeket, az italoknál az alkoholmentes és az alkoholos italokat.	Kínálat	Etlap.vue
A teljeskörű honlap adminisztráció kezelése o admin szintű jogosultságokkal	Adminisztráció	Admin.vue
Ez utasítja a böngészőt, hogy melyik címen (/reservation, /login) melyik felületet kell megjeleníteni.	Térkép / Útmutató	routes.js

2. Adatszerkezet, a program felépítése



Architektúra

- Felhasználó
- Böngésző / Vue.js Frontend
- HTTP (REST / JSON
- Express.js Backend API
- Prisma ORM
- Adatbázis

2.1 Frontend (Vue 3 + Vite) szerkezet

A frontend mappa felépítése, kapcsolódása:

Frontend/

─ src/

├─ main.js -> Vue app indítása

├─ App.vue -> Gyökérkomponens

├─ config/

│ └─ routes.js -> Útvonalak (routing)

├─ pages/ -> Oldalszintű komponensek

│ ├─ Home.vue

│ ├─ Login.vue

```
│   ├─ Etlap.vue
│   ├─ Reservation.vue
│   └─ Admin.vue
└─ components/          -> komponensek
    ├─ Header.vue
    ├─ Footer.vue
    ├─ AuthForm.vue
    ├─ UserList.vue
    └─ UserProfile.vue
└─ pictures/             -> statikus képek
```

Kapcsolatok a frontendben:

- a pages komponensek használják a components elemeit
- HTTP kérések az Express backend felé (pl. bejelentkezés, étlap, foglalás)

2.2. Backend (Node.js + Express + Prisma) szerkezet

A backend mappa felépítése, kapcsolódása:

```
Backend/
─ index.js              -> Express szerver belépési pont
─ swagger.js            -> API dokumentáció
─ controller/
  └─ restaurant-controll.js
      -> HTTP request/response kezelés
─ service/
  └─ restaurant-service.js
      -> Üzleti logika
─ prisma/
  └─ schema.prisma      -> Adatmodellek definíciója
└─ generated/prisma/
  -> Prisma kliens (automatikusan generált)
```

Kapcsolatok – backend szint:

Controller – Service - Prisma Client.- Database

- **Controller**: végpontok (endpointok), validálás, válaszok
- **Service**: üzleti logika (pl. éttermek, felhasználók, foglalások)
- **Prisma**: adatbázis műveletek

2.2.1. Frontend ↔ Backend kommunikáció

- Vue Component
  - fetch / axios
- Express Route (controller)
- Service
- Prisma → Adatbázis
  - JSON formátumú adatok
  - REST-alapú API
  - Authentikáció JWT segítségével (jsonwebtoken, bcrypt)

2.2.2. Összegzés

- **Frontend**: Vue 3 SPA, komponens-alapú felépítés
- **Backend**: Express API, tiszta controller–service architektúra
- **Adatkezelés**: Prisma ORM + relációs adatbázis
- **Kapcsolat**: REST API, HTTP kérések

2.3. A Backend

A backend felépítése

A rendszer egy teljes **háromszintű architektúra**

- Vue.js,
- Express/Node.js/Prisma,
- MySQL

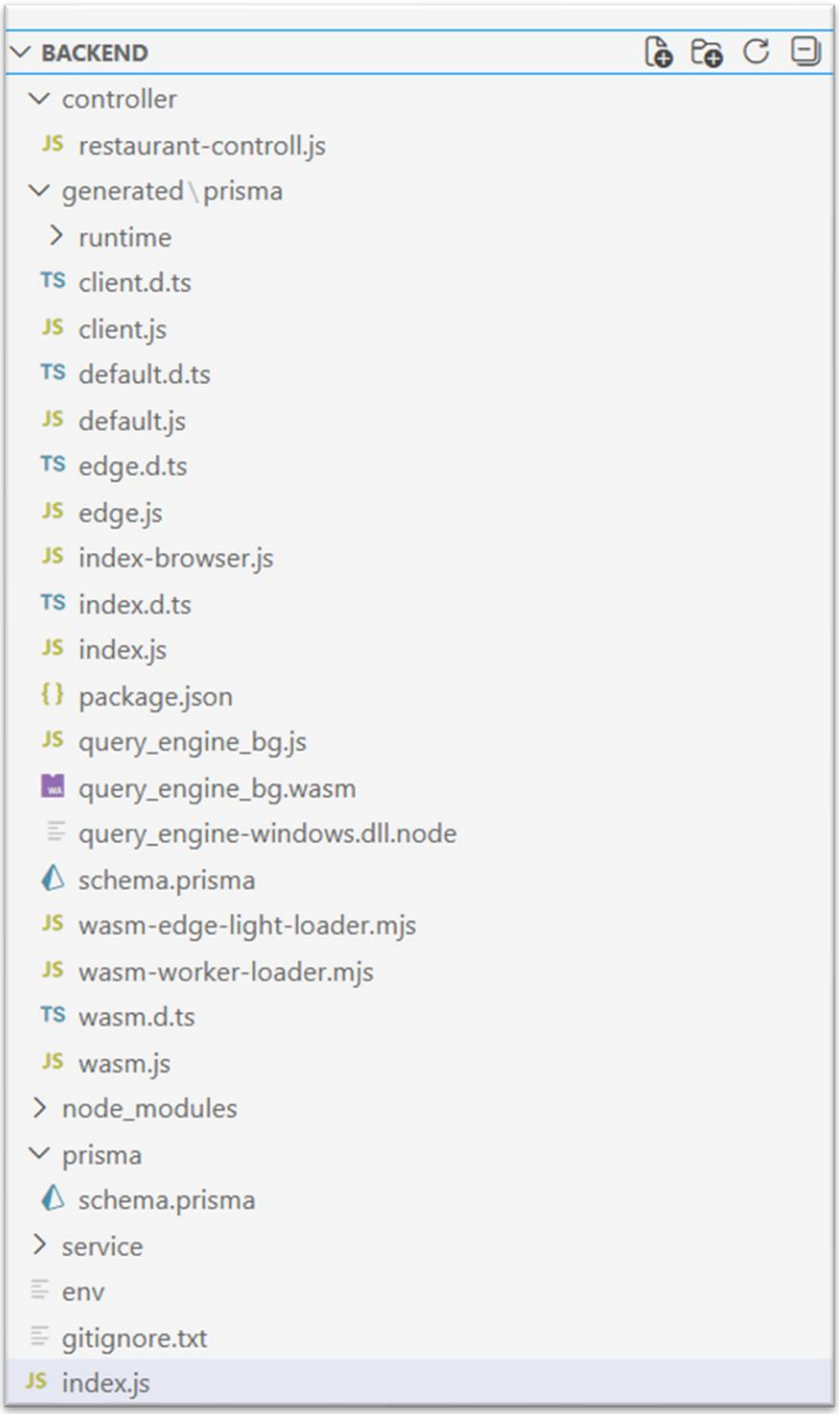
formájában valósul meg, JWT alapú hitelesítéssel.

2.3.1. Inicializáció és Hitelesítés (Index.js)

Az index.js a szerver belépési pontja, amely beállítja a program részére a futtatási környezetet:

- Platform: Node.js és Express.js.
- Port: A szerver a 3300-as porton fut.
- Adatbázis Kapcsolat: Az Express alkalmazás inicializálja a PrismaClient-et, amely az .env fájlban meghatározott MySQL adatbázishoz csatlakozik (mysql://root@localhost:3306/restaurant).
- CORS: Engedélyezi a kommunikációt a frontenddel, amely a http://localhost:5173 címen fut.
- Hitelesítés (Auth):

- A jsonwebtoken (JWT) és bcryptjs könyvtárakat használja.
- A JWT_SECRET környezeti változót használja a token aláírásához.
- Az authenticate middleware minden védett útvonalhoz biztosítja, hogy érvényes JWT token legyen a kérés fejlécében.
- A /login és /register végpontok közvetlenül itt vannak definiálva, és a users táblát módosítják.



2.3.2. API Útvonalak (Controller Layer)

A backend két fő útvonal-kezelő fájlra oszlik:

Funkció	Feladat	fájl neve
Kezeli a /login, /register, /user/:id (profil) és /password/:id végpontokat, JWT ellenőrzéssel.	Hitelesítés és Felhasználói profil	index.js
Kezeli az éttermi funkciókat: vélemények, foglalások, státuszok és asztalok CRUD műveleteit.	Foglalások kezelése	restaurant-controll.js

2.3.3. Adatelérési Részletek (Service és Schema)

- restaurant-service.js (Service Layer): Ez a fájl tartalmazza az üzleti logikát (a weblap működési felületét) és a Prisma ORM hívásokat az adatok kezelésére (pl. reviewsList(), reservationAdd()).
- schema.prisma (Adatszerkezet): A rendszer hátterében futó MySQL adatbázis sémája (táblák, mezők, és kapcsolatok) a Prisma modellnek megfelelően van definiálva.

2.3.4. Kapcsolatok (package.json)

A projekt a következő fontos modulokra támaszkodik a működéshez:

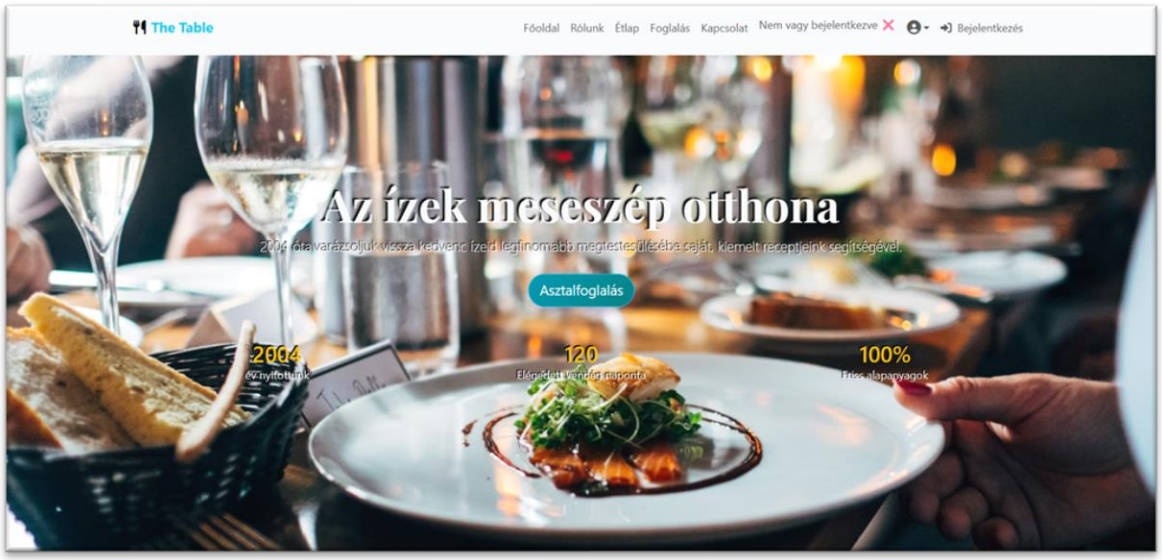
- express, cors
- @prisma/client, prisma
- bcryptjs (jelszó hash-eléshez)
- jsonwebtoken (hitelesítéshez)

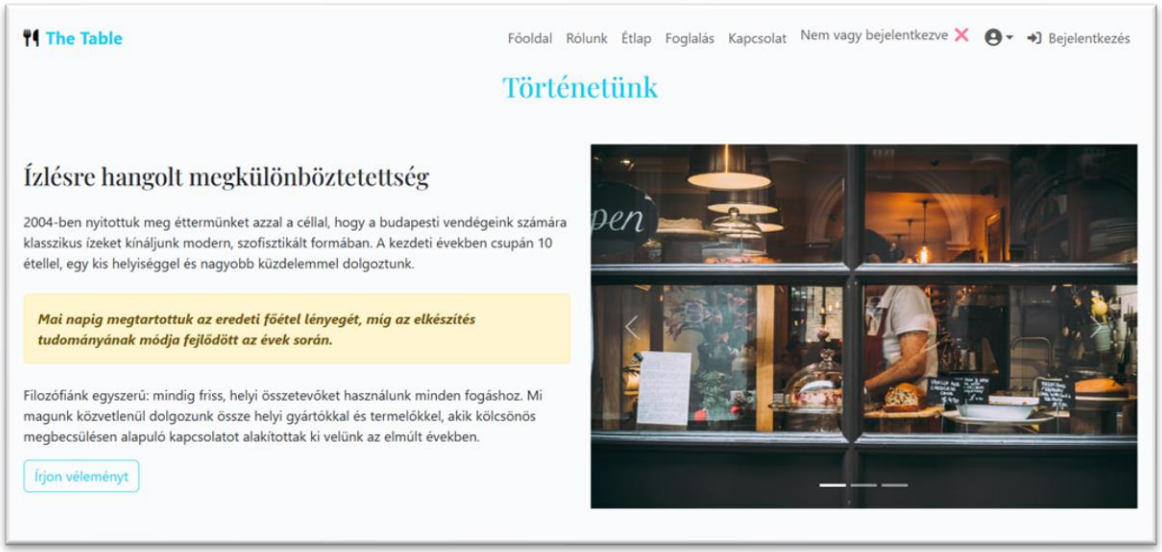
2.4. A Frontend

A frontend alapvető felépítése, alapelemei

2.4.1. A fő elemek

- App.vue: A gyökérkomponens, ami kizárólag a <router-view>-t adja meg, így az aktuális útvonalhoz tartozó oldalkomponenst jeleníti meg.
- routes.js: Ez határozza meg a program útvonalakat, illetve a hozzájuk kapcsolódó elemeket: Home.vue, Login.vue, Reservation.vue.
- RouterLink/router-link: Ezek a Vue Router elemek biztosítják a navigációt az egyes oldalelemek között.





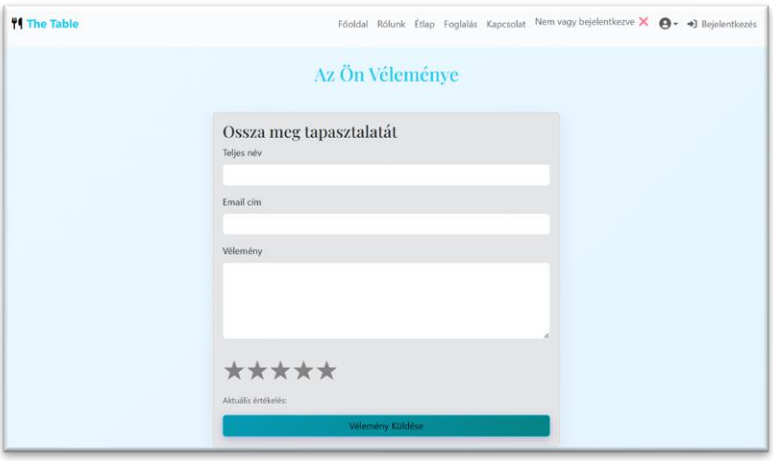
2.4.2. Az oldalkomponensek és az adatkapcsolatok

A. Főoldal (Home.vue)

Ez az oldal kezeli a legtöbb aszinkron adatot:

- user: A bejelentkezett felhasználó adatai (id, name, email).
  - A getUser() API hívás (http://localhost:3300/user) tölti be.
- reviews: Az összes vélemény listája.
  - A getReviews() API hívás (http://localhost:3300/review és http://localhost:3300/reviewslogoutList) tölti be.
- formData: A véleményíró űrlap adatai (rating, comment, email, name).
  - Ez egy defineModel, ami kezeli az űrlap beviteli mezőit.
- Adatküldés: A submitReviewForm függvény a user.value.id alapján két különböző API végpontra küldi az adatokat (bejelentkezett vagy kijelentkezett felhasználó esetén).

Funkció	Metódus	Adatkapcsolat
Lekéri a bejelentkezett felhasználók véleményeit.	GET	http://localhost:3300/review
Lekéri a kijelentkezett felhasználók véleményeit.	GET	http://localhost:3300/reviewslogoutList
Új vélemény felvitele (bejelentkezett user).	POST	http://localhost:3300/reviewsadd
Új vélemény felvitele (kijelentkezett user, névvel / e-mail címmel).	POST	http://localhost:3300/reviewslogoutadd



B. Bejelentkezés/Regisztráció (Login.vue)

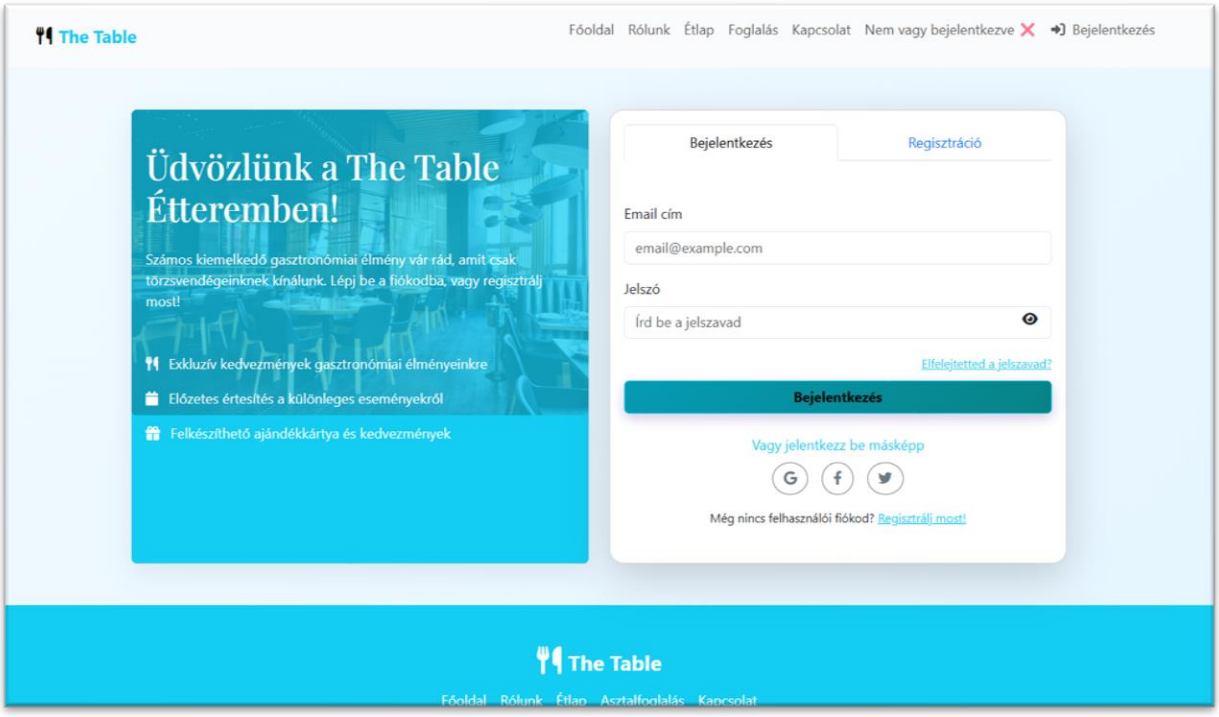
A hitelesítési állapotot kezelő komponens:

- isLoggedIn: jelzi, hogy a felhasználó be van-e jelentkezve.
- AuthForm.vue: Bejelentkezés/regisztráció űrlapja. Ha sikeres a bejelentkezés, frissíti az isLoggedIn értéket.

Funkció	Metódus	Adatkapcsolat
Felhasználó bejelentkeztetése	POST	http://localhost:3300/login
Új felhasználó regisztrálása	POST	http://localhost:3300/register

- UserProfile.vue: Felhasználói profil nézet. Csak akkor jelenik meg, ha az isLoggedIn állítás igaz. Létrehoz egy @logged-out eseményt, ami kezeli a kijelentkezést.

Funkció	Metódus	Adatkapcsolat
Felhasználó kijelentkeztetése	POST	http://localhost:3300/logout



C. Foglalás (Reservation.vue)

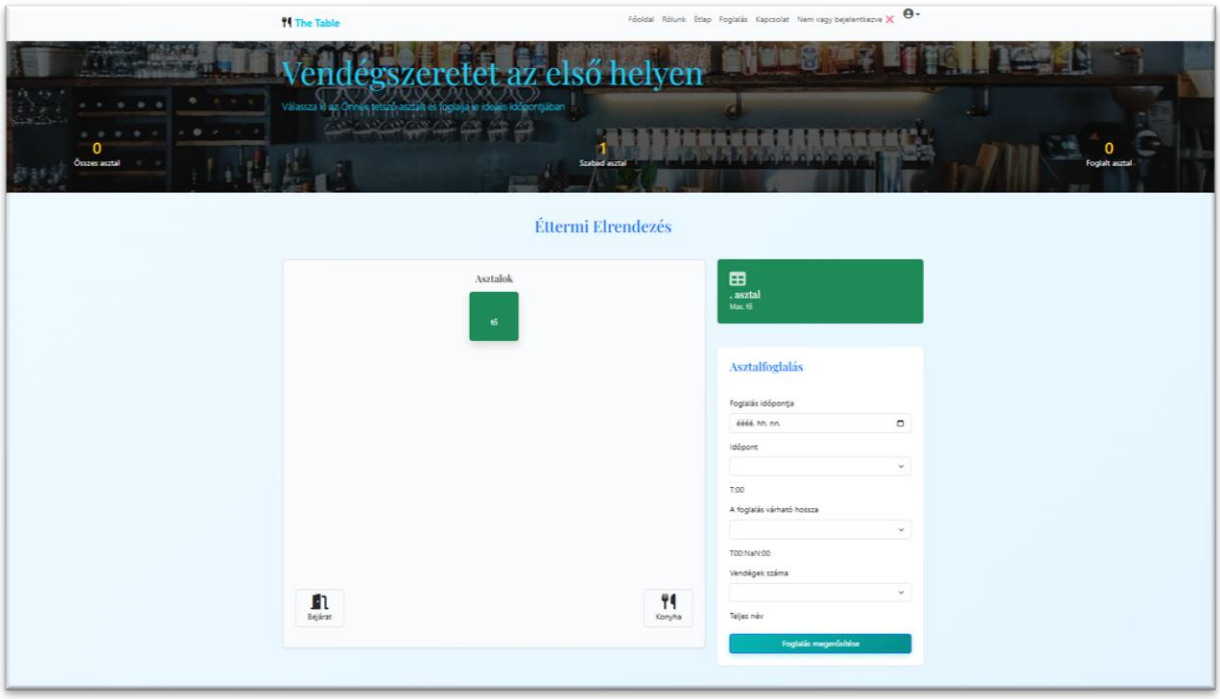
Ez az oldal a foglaláshoz szükséges adatokkal dolgozik:

- user: A bejelentkezett felhasználó adatai.
    - A getUser() API hívással töltődik be.
- table: Az éttermi asztalok adatai.
    - A getTable() API hívással (http://localhost:3300/table) töltődik be.
- formData: A foglalási űrlap adatai (table_id, status_id).
    - A reservationAdd() API hívással küldi el az adatokat.

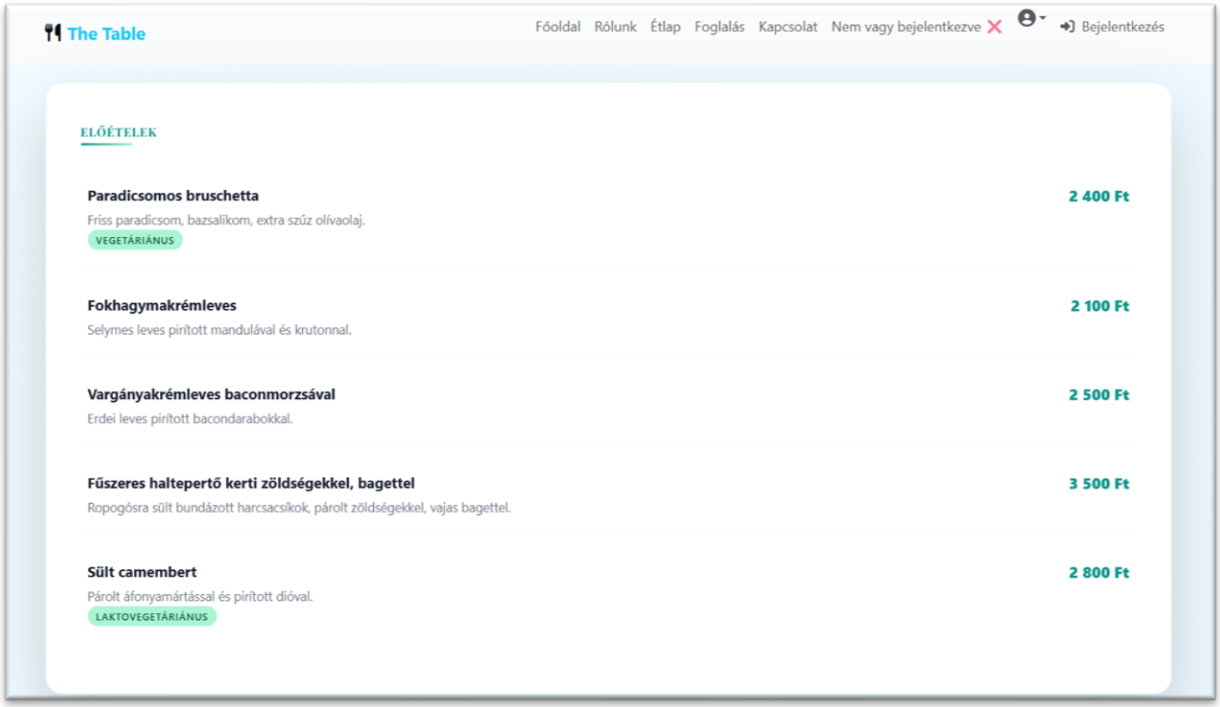
Funkció	Metódus	Adatkapcsolat
Lekéri a bejelentkezett felhasználót	GET	http://localhost:3300/user
Lekéri az asztalok adatait	GET	http://localhost:3300/table
Új asztalfoglalás rögzítése a megadott adatokkal	POST	http://localhost:3300/reservationadd

- Kapcsolódó táblák

Funkció	Kapcsolat	Kapcsolódó Táblák	id
Egy felhasználó több foglalást is kezdeményezhet	Foglalás tulajdonosa	users ↔ reservation	user_id
Egy asztalhoz több foglalás is tartozhat (de különböző idősávokban)	Foglalás helye	tables ↔ reservation	table_id
Egy státusz (szabad v. foglalt v. törölve) több foglaláshoz is tartozhat.	Foglalás állapota	reservation status ↔ reservation	status_id



Az étlap bőséges választékot kínál.



2.5 Teszt dokumentáció

A tesztelés célja, hogy a rendszer viselkedését ellenőrizhetővé tegye, és minél hamarabb felfedje a hibákat a fejlesztés során

Tesztszintek röviden:

- Unit tesztek: Kis, izolált egységeket (függvény, komponens) vizsgálnak, gyorsak és fejlesztő közeli visszajelzést adnak.
- Integrációs tesztek: Több komponens vagy szolgáltatás együttműködését ellenőrzik (pl. backend endpoint + adatbázis).
- End-to-end (E2E) tesztek: Valós felhasználói folyamatokat szimulálnak a teljes rendszeren keresztül, pl. „regisztráció a felületen keresztül”

Backend-ben 3 szintet ellenőriztünk, az adatbázis teszt, unit teszt, integration teszt.

3 rész lett tesztelve (jest teszt): regisztráció, foglalás és az asztal státusz lekérdezés, ebből a regisztráció tesztet mutatom be a dokumentációba.

Frontendben ugyancsak 3 programrész lett tesztelve (vitest és playwrigth): a regisztráció , bejelentkezés és a véleményezés.Itt csak a regisztráció tesztre hozok példát.

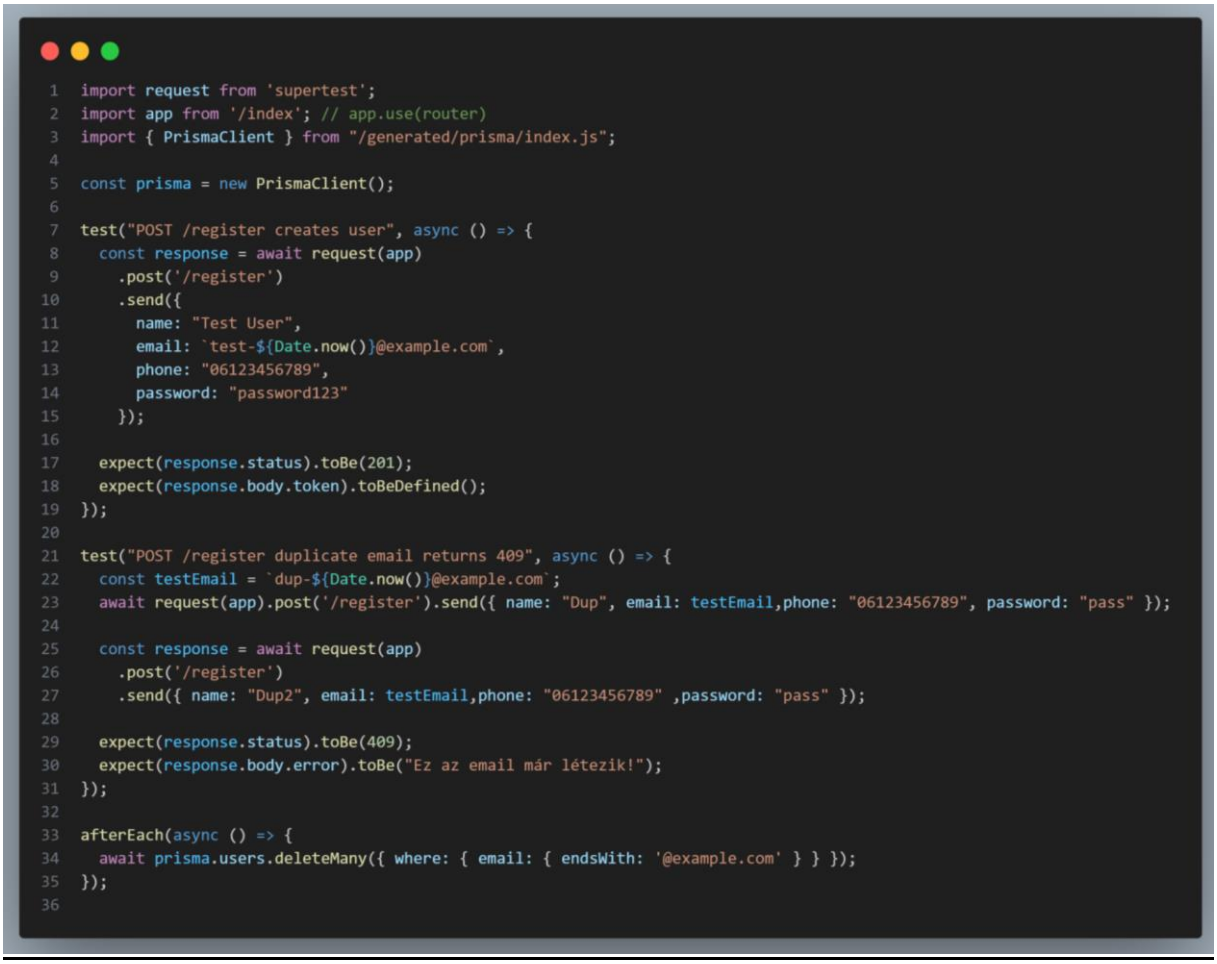
2.5.1. Backend tesztek:

register.db.test.js:

```
import { PrismaClient } from "/generated/prisma/index.js";
import bcrypt from 'bcrypt';

const prisma = new PrismaClient();

test("Prisma users.create works with hashed password", async () => {
  const hashedPassword = await bcrypt.hash("password123", 10);
  const data = await prisma.users.create({
    data: {
      name: "Test User",
      email: "test@example.com",
      phone: "06123456789",
      password: hashedPassword,
      role: "USER",
    },
  });

  expect(data.id).toBeDefined();
  expect(data.email).toBe("test@example.com");
  expect(data.phone).toBe("06123456789");
  expect(data.role).toBe("USER");
  expect(await bcrypt.compare("password123", data.password)).toBe(true);
}, 15000);

afterEach(async () => {
  await prisma.users.deleteMany({ where: { email: "test@example.com" } });
});

afterAll(async () => {
  await prisma.$disconnect();
});

```

Itt közvetlenül Prisma users.create hívást tesztel: létrehoz egy user-t hash-elt jelszóval, majd ellenőrzi, hogy az adatok (email, telefon, szerepkör) jól mentek-e el, és hogy a jelszó tényleg bcrypt-tel van hash-elve.

register.integration. test. js



```
import request from 'supertest';
import app from '/index'; // app.use(router)
import { PrismaClient } from "/generated/prisma/index.js";

const prisma = new PrismaClient();

test("POST /register creates user", async () => {
  const response = await request(app)
    .post('/register')
    .send({
      name: "Test User",
      email: `test-${Date.now()}@example.com`,
      phone: "06123456789",
      password: "password123"
    });

  expect(response.status).toBe(201);
  expect(response.body.token).toBeDefined();
});

test("POST /register duplicate email returns 409", async () => {
  const testEmail = `dup-${Date.now()}@example.com`;
  await request(app).post('/register').send({ name: "Dup", email: testEmail,phone: "06123456789", password: "pass" });

  const response = await request(app)
    .post('/register')
    .send({ name: "Dup2", email: testEmail,phone: "06123456789" ,password: "pass" });

  expect(response.status).toBe(409);
  expect(response.body.error).toBe("Ez az email már létezik!");
});

afterEach(async () => {
  await prisma.users.deleteMany({ where: { email: { endsWith: '@example.com' } } });
});

```

Integration teszt a teljes /register endpointra supertest-tel:

Sikeres regisztráció után 201-et és tokent vár.

Dupla email esetén 409-et és a "Ez az email már létezik!" hibaüzenetet vár.

register.service.test.js

```
import { PrismaClient } from "/generated/prisma/index.js";
import bcrypt from 'bcrypt';

const prisma = new PrismaClient();

test('register teljes logika működik mock nélkül', async () => {
  const req = {
    body: {
      name: 'Teszt Elek',
      email: `unit-${Date.now()}@example.com`,
      phone: '06123456789',
      password: 'titkos123'
    }
  };

  const res = {
    statusCalledWith: null,
    jsonCalledWith: null,
    status: function(code) {
      this.statusCalledWith = code;
      return this;
    },
    json: function(data) {
      this.jsonCalledWith = data;
      return this;
    }
  };

  // TELJES route kód bemásolása (generateToken inline-ra cserélve)
  const { name, email, phone, password } = req.body;

  if (!name || !email || !password || !phone) {
    res.status(400).json({ error: "Név, email megadása kötelező" });
  } else {
    try {
      const hashedPassword = await bcrypt.hash(password, 10); // VALÓS bcrypt
      const newUser = await prisma.users.create({
        data: { name, email, phone, password: hashedPassword, role: "USER" }
      });

      // Inline token (JWT vagy base64)
      const token = Buffer.from(JSON.stringify({ id: newUser.id, email: newUser.email })).toString('base64');

      res.status(201).json({ token });
    } catch (error) {
      if (error.code === "P2002") {
        res.status(409).json({ error: "Ez az email már létezik!" });
      } else {
        res.status(500).json({ error: "Szerver hiba!" });
      }
    }
  }

  // Ellenőrzés
  expect(res.statusCalledWith).toBe(201);
  expect(res.jsonCalledWith.token).toBeDefined();
  expect(res.jsonCalledWith.token.length).toBeGreaterThan(10);
}, 10000);

afterEach(async () => {
  await prisma.users.deleteMany({ where: { email: { endsWith: '@example.com' } } });
});

```

A route teljes logikáját futtatja (validáció, bcrypt hash, Prisma create, token generálás).

Ellenőrzi, hogy siker esetén 201 státusz és egy elég hosszú token érkezik.

2.5.2. Frontend tesztek:

register.test.js (Vitest + Vue Test Utils):

```
// tests/register.test.js - VÉGLEGES ZÖLD VERZIÓ
import { describe, it, expect, vi, beforeEach } from 'vitest'
import { mount, flushPromises } from '@vue/test-utils'
import AuthForm from '../src/components/AuthForm.vue'

describe('AuthForm - Register', () => {
  beforeEach(() => {
    vi.clearAllMocks()
    window.localStorage.getItem = vi.fn(() => null)
    window.localStorage.setItem = vi.fn()
    window.bootstrap = { Modal: { getOrCreateInstance: vi.fn(() => ({ show: vi.fn() })) } }
      window.fetch = vi.fn().mockImplementation((url) => {
    console.log(`🎯 Mocked fetch: ${url}`)
    return Promise.resolve({
      ok: true,
      json: () => Promise.resolve({
        message: url.includes('check-email') ? 'Az e-mail cím elérhető.' : {}
      })
    })
  })
  })

  // ✅ Ezek működnek – hagyd meg
  it('register tab exists and switches', async () => {
    const wrapper = mount(AuthForm)
    await wrapper.find('[data-testid="register-tab"]').trigger('click')
    expect(wrapper.find('#reg-name').exists()).toBe(true)
  })

  it('register form fields exist', async () => {
    const wrapper = mount(AuthForm)
    await wrapper.find('[data-testid="register-tab"]').trigger('click')
    expect(wrapper.find('#reg-name').exists()).toBe(true)
  })

  it('register without checkbox shows warning modal', async () => {
    const wrapper = mount(AuthForm)
    await wrapper.find('[data-testid="register-tab"]').trigger('click')
    const forms = wrapper.findAll('form')
    await forms[1].trigger('submit.prevent')
    await flushPromises()
    expect(window.bootstrap.Modal.getOrCreateInstance).toHaveBeenCalled()
  })

  it('checkEmailAvailability works', async () => {

    const wrapper = mount(AuthForm)
    const result = await wrapper.vm.checkEmailAvailability('test@example.com')
    expect(result).toBe(true)
    expect(window.fetch).toHaveBeenCalledWith('http://localhost:3300/check-email', expect.any(Object))
  })

  // ✅ ÚJ: teszteli, HOGY a register FÜGGVÉNY legalább fut
  it('register function is called and validates', async () => {
    const wrapper = mount(AuthForm)
    wrapper.vm.name = 'Kovács János Csaba'
    wrapper.vm.email = 'test@example.com'
    wrapper.vm.password = 'AbcDefG1!@#'
    wrapper.vm.passwordConfirm = 'AbcDefG1!@#'
    wrapper.vm.phone = '+36301234567'

    window.fetch.mockResolvedValueOnce({
      ok: true,
      json: () => Promise.resolve({ message: 'Elérhető' })
    })

    await wrapper.vm.register()
    await flushPromises()

    // ✅ Legalább addig eljut, hogy checkEmailAvailability meghívódjon
    expect(window.fetch).toHaveBeenCalledWith('http://localhost:3300/check-email', expect.any(Object))
  })
})

```

- Az AuthForm komponens regisztrációs részét teszteli: létezik-e a regisztrációs fül és mezők, működik-e a váltás a tabok között.
- Mockolja a fetch-et és a Bootstrap modalt, ellenőrzi, hogy checkbox nélkül figyelmeztető modal jön, hogy az email-ellenőrzés (checkEmailAvailability) hívódik, és hogy a register függvény lefut érvényes adatokkal.

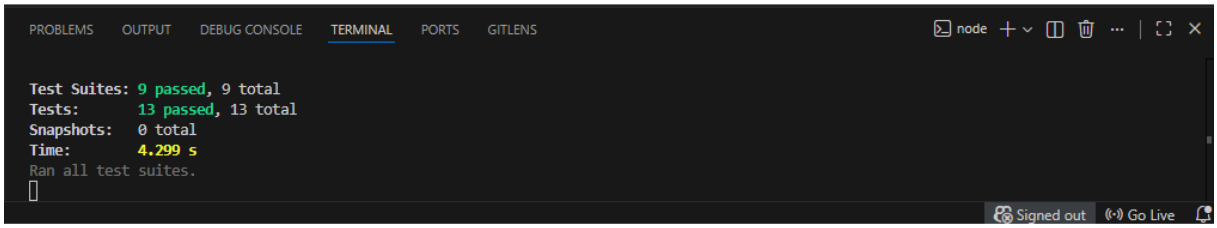
register.e2e.spec.js (Playwright E2E):

```
// tests/register.e2e.spec.js
import { test, expect } from '@playwright/test'

test.describe('AuthForm - Regisztráció E2E', () => {
  test('sikeres regisztráció és siker modal', async ({ page }) => {
    // Oldal megnyitása
    await page.goto('http://localhost:5173/login')

    // Regisztráció fülre váltás
    await page.getByTestId('register-tab').click()

    // Adatok kitöltése
    await page.fill('#reg-name', 'Kovács János')
    await page.fill('#reg-tel', '06301234567')
    await page.fill('#reg-email', 'playwright+test@example.com')
    await page.fill('#reg-password', 'AbcDef12!@')
    await page.fill('#reg-confirm', 'AbcDef12!@')

    // Adatvédelmi checkbox
    await page.check('#terms')

    // Regisztráció gomb
    await page.locator('form').filter({ hasText: 'Már van felhasználói fiókod?' }).getByRole('button', { name: 'Regisztráció' }).click()

    // Várjuk a siker modalt
    const modal = page.locator('.modal.show')
    await expect(modal).toBeVisible()

    // Siker szöveg ellenőrzése
    await expect(modal).toContainText('Sikeres regisztráció!')
  })
})
```

- Valódi böngészőben nyitja meg a login oldalt, átvált a regisztráció fülre, kitölti az összes mezőt és bepipálja a feltételek checkboxot.
- Rákattint a „Regisztráció” gombra, majd elvárja, hogy megjelenjen a siker modal, benne a „Sikeres regisztráció!” szöveggel.

2.5.3. Teszt eredmények:

Backend (jest) test:



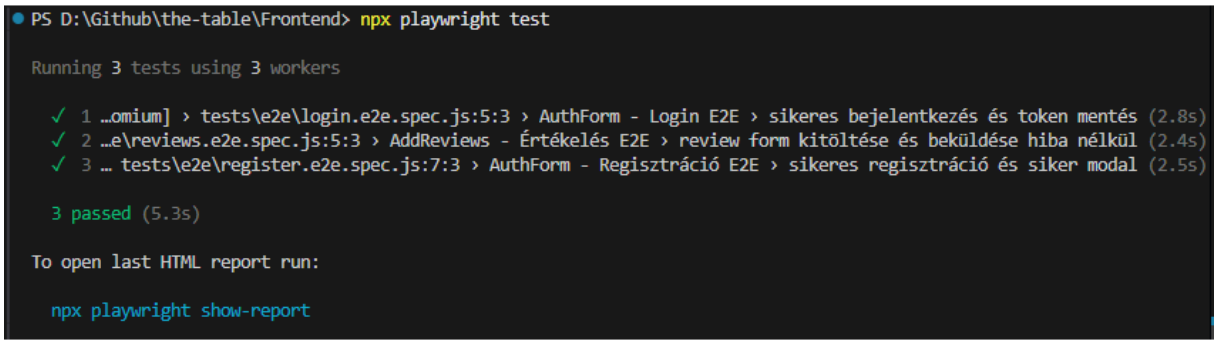
Minden teszt sikeresen lefutott.

Frontend test:

Unit test (vitest)

Minden teszt sikeresen lefutott.

Playwrigth test:



Minden teszt sikeresen lefutott.

3. Felhasználói dokumentáció

The Table Étterem (Vue.js + Node.js)

Ez a dokumentum bemutatja a The Table Étterem webes alkalmazásának használatát Az alkalmazás elsődleges céljai:

- az étterem bemutatása,
- felhasználói vélemények kezelése,
- időponthoz rendelhető előzetes asztalfoglalás.

Projeckt inditása:

1.XAMPP program indítása.A MySQL és Apache szerverek elindítása.

2. A phpmyadmin oldal megnyitása és a /docs mappában lévő restaurant-database.sql fájl importálása.

Backend indítása:

3. Visual Studio Code program elindítása majd a the table/Backend mappa megnyitása.

4.Terminálba be kell írni : **npm install**

5. Az .env fájl tartalmát módosítani kell a /docs/ .env.example fájlból.

6.A backend elindításához a **node .** parancsot használjuk.

Frontend elindítása:

7. A Visual Studio Code program elindítása majd a the table/Frontend mappa megnyitása.

8.Terminálba be kell írni: **npm install**

9. Az **npm run dev** utasítással elindul a program a http://localhost:5173/.

Ha szükséges a Vue-cal alkalmazás telepítése:**npm i vue-cal@5.0.1-rc.33**

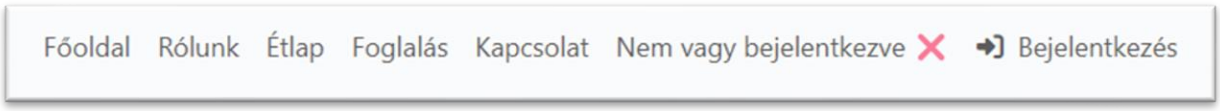
A program indulása után belépési adatok:

Felhasználó: e-mail: test1@test.hu Jelszó: Test65eset!

Admin: e-mail: admin@admin.hu Jelszó: Admin9!+

3.1 Navigáció a weboldalon és a legfontosabb funkciók (felső menüsor)

- Főoldal (/)
  - Bemutatja az étterem bemutatkozó, kreatív megjelenésű fő információs oldalát, a vélemények listáját és lehetőségként egy véleményíró űrlapot.
- Bejelentkezés / Regisztráció (/login)
  - Lehetővé teszi az új felhasználók regisztrációját és a már regisztrációval rendelkező felhasználók bejelentkezését.
- Asztalfoglalás (/reservation)
  - A foglalási űrlap segítségével lehet időpont megadásával asztalt foglalni. A foglalásnál az ütközéseket automatikusan kerüli a program.
- Felhasználói profil (/user)
  - Bejelentkezés után itt lehet megtekinteni és módosítani a felhasználó rögzített adatait, illetve itt tud a felhasználó kijelentkezni.



3.2 Hitelesítés és Profil Kezelése

1. Regisztráció

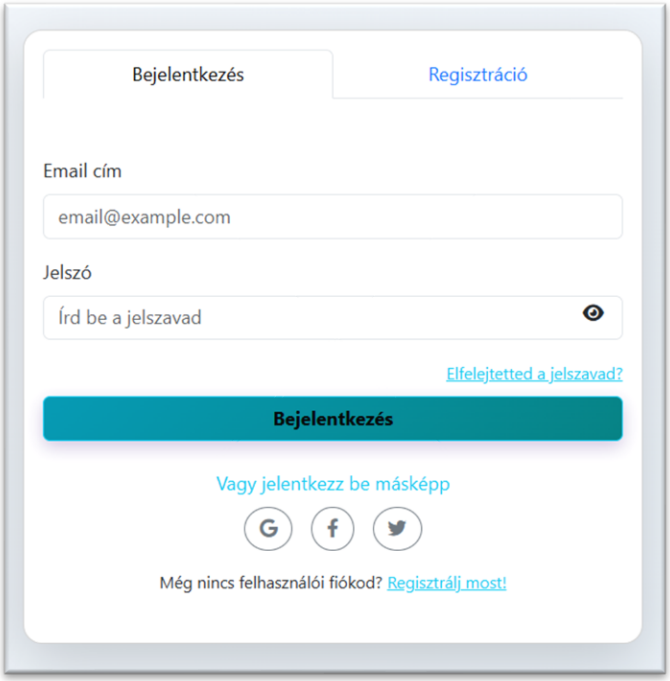
- Kattintson a Bejelentkezés / Regisztráció linkre (/login).
- Kattintson a "Regisztráció" fülre vagy linkre.
- Töltse ki a szükséges mezőket: Név, E-mail, Telefonszám és Jelszó. (kötelezően megadandó elemek)
- Kattintson a "Regisztrálok" gombra.
- Sikeres regisztráció után automatikusan bejelentkezik a rendszerbe.

2. Bejelentkezés (Login)

- Kattintson a Bejelentkezés / Regisztráció linkre (/login).
- Adja meg a regisztrált E-mail címét és Jelszavát.
- Kattintson a "Bejelentkezés" gombra.
- Sikeres bejelentkezés esetén a rendszer eltárol egy biztonsági tokent, és átirányítja a Felhasználói Profil oldalra, vagy megjeleníti a profil nézetet.

3. Kijelentkezés (Logout)

- Navigáljon a Felhasználói profil oldalra (/user vagy /login).
- Kattintson a "Kijelentkezés" gombra.
- A rendszer eltávolítja a biztonsági tokent, és Ön kijelentkezik az alkalmazásból.



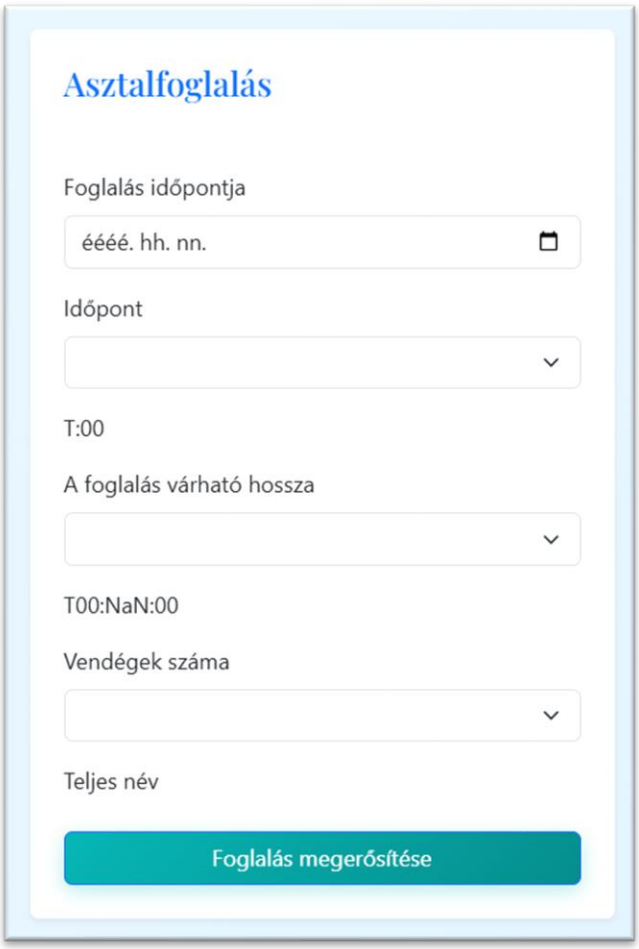
3.3 Asztalfoglalás

Az asztalfoglaláshoz látogasson el az Asztalfoglalás oldalra (/reservation).

1. Foglalási Űrlap Kitöltése

A foglaláshoz az alábbi adatok megadására van szükség:

- Asztal azonosítója (Table ID): Az éttermi asztal azonosítója
- Státusz azonosítója (Status ID): A foglalás aktuális státuszának kódja
- Kezdő időpont (dtime_from): A foglalás kívánt kezdete.
- Befejező időpont (dtime_to): A foglalás várható befejezése.
- Vendégek száma (Number of Customers): A lefoglalni kívánt asztalnál a várható vendégek számának megadása.



2. Foglalás Elküldése

- Töltse ki pontosan az űrlapot.
- Kattintson a "Foglalás elküldése" gombra (a kódban reservationAdd() függvény).
- A rendszer elküldi a foglalási kérést a szervernek. Sikeres foglalás esetén értesítést kap.

3.4 Vélemények Írása

A véleményeket a Főoldalon írhatja meg.

1. Vélemény Űrlap Kitöltése

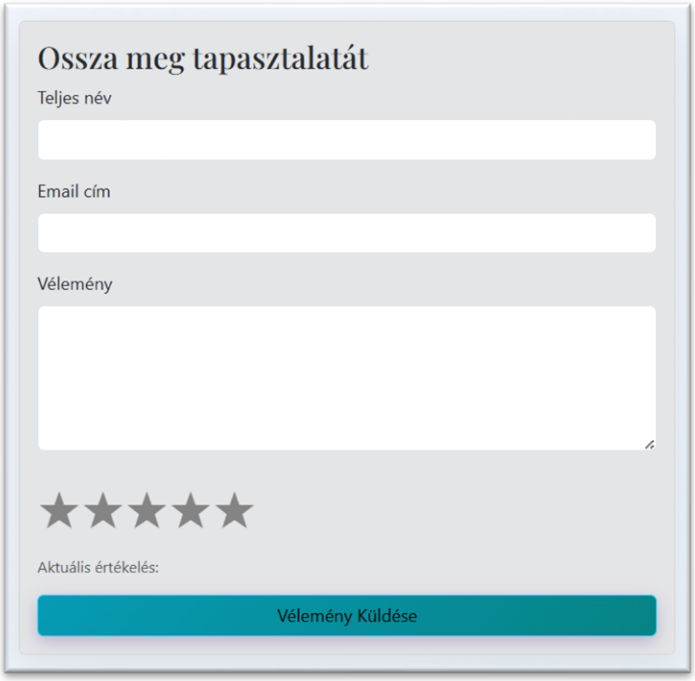
A vélemény űrlapon a következő mezőket találja:

- Értékelés (Rating): Az étterem értékelése (pl. 1-5).
- Vélemény (Comment): A beírandó szöveges értékelés.
- Név és E-mail:

- Bejelentkezett felhasználóként: Ezek a mezők automatikusan kitöltődnek az Ön profiladatai alapján (vagy eltűnnek, és a user_id kerül elküldésre a /reviewsadd végpontra).
- Kijelentkezett felhasználóként: Meg kell adnia a Nevét és E-mail címét (az adatok a /reviewslogoutadd végpontra kerülnek).

2. Vélemény Elküldése

- Töltse ki a szükséges mezőket.
- Kattintson a "Vélemény elküldése" gombra (a kódban submitReviewForm() függvény).
- A rendszer elmenti a véleményét a megfelelő adatbázis táblába.

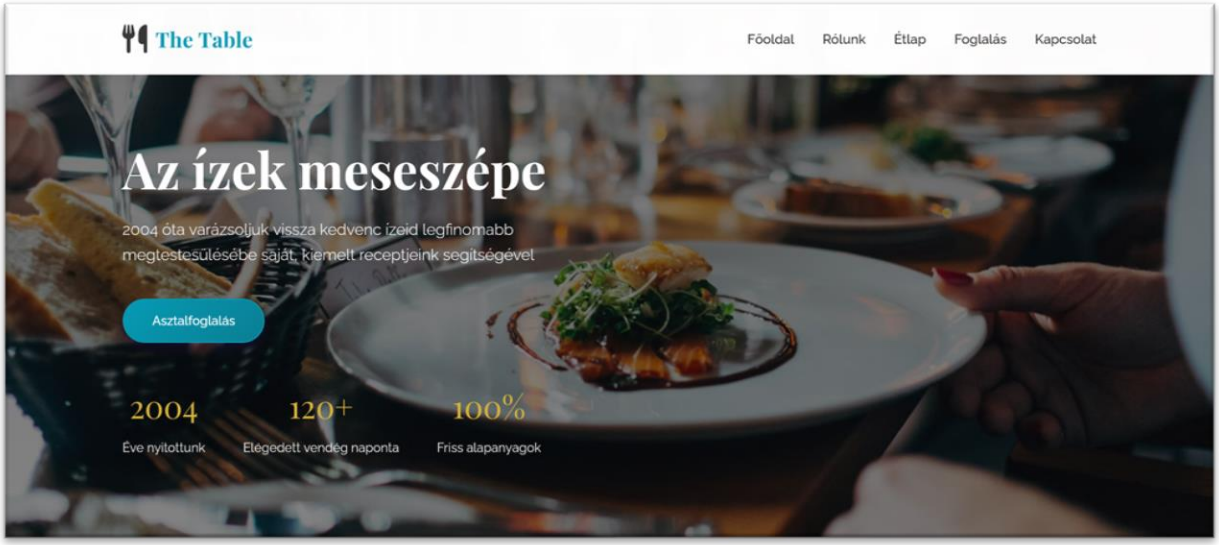


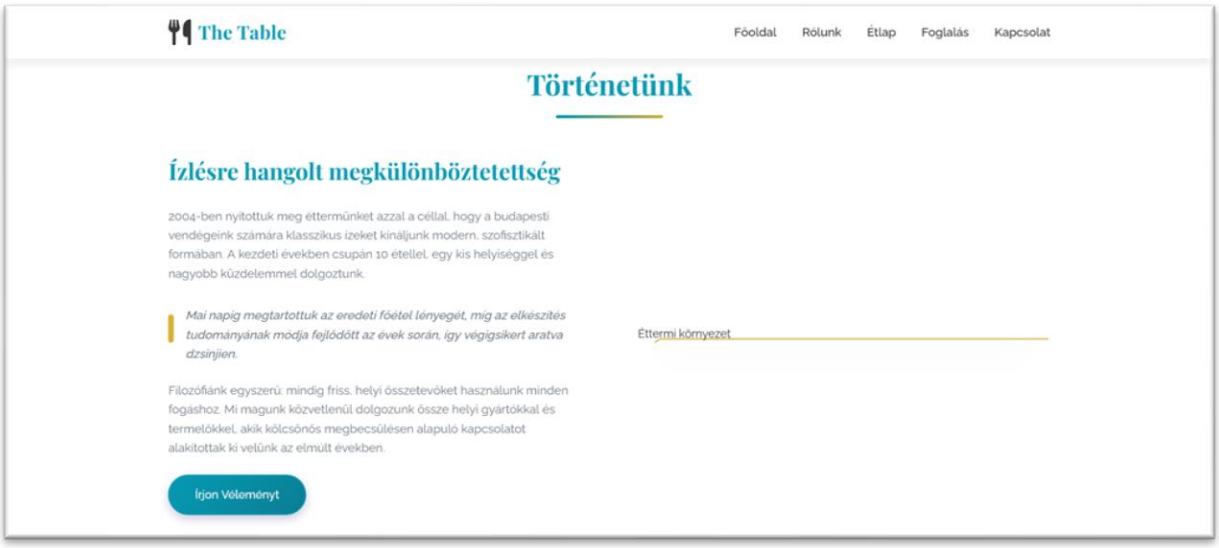
3.5 Hibaelhárítás

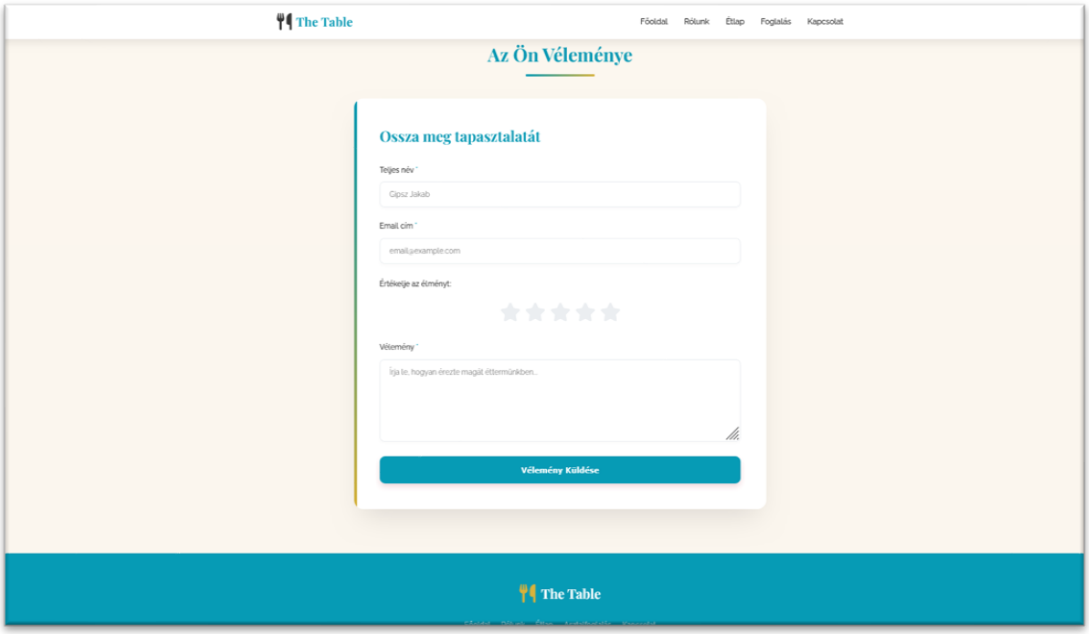
Probléma	Lehetséges Ok	Megoldás
Nem sikerül bejelentkezni	Hibás e-mail vagy jelszó	Ellenőrizze az e-mail címet és a jelszót! Próbálja meg újra a regisztrációt, ha elfelejtette a jelszavát!

A Foglalás gomb nem csinál semmit	Hiányzó adat az űrlapon	Ellenőrizze, hogy minden kötelező mezőt kitöltött-e, különös tekintettel az ID-kre (asztal és státusz)!
"Nem sikerült a felhasználót lekérni!" hibaüzenet	A backend szerver nem fut, vagy nem a 3300-as porton	Ellenőrizze, hogy a backend elindult-e!
Nem jelennek meg a vélemények	Probléma van a /review vagy /reviewslogoutList API hívásokkal	Ellenőrizze a szerver és az adatbázis kapcsolatát!

A tervezet képei:




The Table
Főoldal
Rólunk
Étlap
Foglalás
Kapcsolat
Történetünk
Ízlésre hangolt megkülönböztetettség
2004-ben nyitottuk meg éttermünket azzal a céllal, hogy a budapesti vendégeink számára klasszikus ízeket kínáljunk modern, szofisztikált formában. A kezdeti években csupán 10 étellel, egy kis helyiséggel és nagyobb küzdelemmel dolgoztunk.
Mai napig megtartottuk az eredeti főétel lényegét, míg az elkészítés tudományának módja fejlődött az évek során, így végigsikert aratva dzsinjien.
Filozófiánk egyszerű: mindig friss, helyi összetevőket használunk minden fogáshoz. Mi magunk közvetlenül dolgozunk össze helyi gyártókkal és termelőkkel, akik kölcsönös megbecsülésen alapuló kapcsolatot alakítottak ki velünk az elmúlt években.
Étermi környezet
Írjon Véleményt


The Table
Főoldal
Rólunk
Étlap
Foglalás
Kapcsolat
Az Ön Véleménye
Ossza meg tapasztalatát
Teljes név *
Gipsz Jakab
Email cím *
email@example.com
Értékelje az élményt:
Vélemény *
Írja le, hogyan érezte magát éttermünkben...
Vélemény Küldése
The Table


The Table
Főoldal
Rólunk
Étlap
Asztalfoglalás
Kapcsolat
© 2004-2023 Minden jog fenntartva | The Table Étterem