JEGYZŐKÖNYV

Webprogramozás II.
Féléves beadandó

Készítette: **Orosz Kristóf**
Neptunkód: **EYZWG9**
Dátum: 2025. december 05.

Sárospatak, 2025

Tartalomjegyzék

Bevezetés

A projekt során egy olyan webes alkalmazást készítettem, amely a lakók és bejelentések kezelését, valamint a felhasználók bejelentkezését és regisztrációját valósítja meg. A rendszert Angular frontenddel és Node.js backenddel fejlesztettem. A jegyzőkönyvben bemutatom a fejlesztés fő lépéseit és a komponensek működését.

Adatbázis létrehozása

Az adatbázist MariaDB-ben hoztam létre, *webprogramozas* néven, és a feladat igényei alapján több külön táblát alakítottam ki. A táblák szerkezete úgy készült el, hogy a lakók, a felhasználók és a bejelentések adatai jól elkülönüljenek, és később is könnyen bővíthetők legyenek.

```
CREATE TABLE lakok (
    id INT AUTO_INCREMENT PRIMARY KEY,
    felhasznaloId INT NOT NULL,
    nev VARCHAR(100) NOT NULL,
    lakasszam VARCHAR(10) NOT NULL,
    emelet INT DEFAULT 0,
    telefonszam VARCHAR(20),
    koltseg DECIMAL(10,2) DEFAULT 0,
    megjegyzes VARCHAR(255)
);


INSERT INTO lakok (felhasznaloId, nev, lakasszam, emelet, telefonszam, koltseg, megjegyzes) VALUES
(1, 'Kiss Anna', '4', 2, '06701234567', 4500, 'Csendes lakó'),
(2, 'Szabó Bence', '11', 3, '06703334444', 5100, 'Pontos fizető'),
(3, 'Tóth Eszter', '7', 1, '06704567890', 3800, 'Kisgyermekes'),
(4, 'Horváth Máté', '2', 0, '06707778899', 4200, 'Jól fizet'),
(5, 'Nagy Dóra', '15', 5, '06706665555', 6000, 'Sok csomagot kap'),
(6, 'Farkas Gergő', '9', 3, '06709991122', 4700, 'Rendszeresen segít'),
(7, 'Kerekes Lili', '18', 4, '06702223344', 6900, 'Ritkán van otthon'),
(8, 'Molnár Balázs', '3', 1, '06705554433', 5200, 'Barátságos'),
(9, 'Varga Noémi', '12', 3, '06708887766', 4100, 'Diák'),
(10, 'Barta Kristóf', '6', 2, '06701112222', 5600, 'Új lakó'),
(11, 'Jakab Petra', '10', 3, '06709998877', 3900, 'Figyelmeztetve volt zaj miatt'),
(12, 'Fülöp Norbert', '5', 1, '06703335555', 6100, 'Jól fizet'),
(13, 'Oroszi Veronika', '14', 4, '06701239876', 3400, 'Csendes és segítőkész'),
(14, 'Lakatos Márió', '8', 2, '06704445555', 5800, 'Felújítást tervez'),
(15, 'Simon Karolina', '20', 6, '06707776655', 7200, 'Mindig időben fizet');
```

```
CREATE TABLE bejelentesek (
    id INT AUTO_INCREMENT PRIMARY KEY,
    lakoId INT NOT NULL,
    tipus VARCHAR(100) NOT NULL,
    leiras TEXT NOT NULL
);

INSERT INTO bejelentesek (lakoId, tipus, leiras) VALUES
(1, 'Vízszivárgás', 'A fürdőszobában a csap alól csöpög a víz.'),
(2, 'Fűtés probléma', 'A radiátor nem melegszik fel teljesen a nappaliban.'),
(3, 'Világítás hiba', 'A konyhában a mennyezeti lámpa villog.'),
(4, 'Zajpanasz', 'A fölöttünk lakó késő estig hangosan zenét hallgat.'),
(5, 'Lift meghibásodás', 'A lift a 3. emeletről nem indul el lefelé.'),
(6, 'Kártevők', 'Hangyák jelentek meg a konyhapult mellett.'),
(7, 'Beázás', 'Az erkély ajtónál befolyik az esővíz.'),
(8, 'Repedt fal', 'A nappali egyik fala hosszában megrepedt.'),
(9, 'Légkondi hiba', 'A légkondicionáló kellemetlen szagot áraszt indításkor.'),
(10, 'Gázszivárgás gyanú', 'A konyhában erős gázszag érződött reggel.'),
(11, 'Csőtörés', 'A WC mögötti csőből folyamatosan szivárog a víz.'),
(12, 'Kulcsprobléma', 'A bejárati ajtó zárja nehezen fordul el.'),
(13, 'Penész', 'A hálószobában penész jelent meg a sarokban.'),
(14, 'Elszívó hiba', 'A fürdőszobai elszívó nem kapcsol be.'),
(15, 'Közös terület koszossága', 'A lépcsőházban hetek óta nem volt felmosva.');

CREATE TABLE felhasznalok (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nev VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    jelszo VARCHAR(100) NOT NULL
);
```

Adatbázis kapcsolat létrehozása

Az adatbáziskapcsolatot a mysql2 modullal valósítottam meg, és egy külön fájlban hoztam létre a konfigurációt. A kapcsolat létrehozásakor megadtam a szükséges szerveradatokat, majd biztosítottam, hogy a rendszer megfelelő hibakezelést alkalmazzon sikertelen csatlakozás esetén.

```
const mysql = require('mysql2');
// Itt betöltöm a mysql2 modult, hogy tudjak kapcsolódni az adatbázishoz.

const kapcsolat = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "webprogramozas",
  charset: "utf8mb4"
  // Itt beállítom az adatbázis kapcsolat konfigurációját.
});

kapcsolat.connect(err => {
  // Itt megpróbálom felépíteni a kapcsolatot az adatbázissal.
  if (err) {
    console.error("Nem sikerült kapcsolódni a MariaDB-hez:", err);
    // Ha hiba van, akkor kiírom, hogy nem tudtam csatlakozni.
    return;
  }
  console.log("Sikeres csatlakozás a MariaDB adatbázishoz!");
  // Ha nincs hiba, akkor jelzem, hogy sikeresen csatlakoztam.
});

module.exports = kapcsolat;
// Itt exportálom a kapcsolatot, hogy más fájlokban is használhassam.
```

Belépési, regisztrációs, komponens működése

A belépési és regisztrációs felületet Angularban készítettem el, és teljesen működőképes kommunikációt alakítottam ki a Node.js backenddel. A bejelentkezés során az e-mail és jelszó ellenőrzése a backendben zajlik SQL lekérdezéssel, hibakezeléssel együtt. A regisztráció során a felhasználó adatait validálom, majd eltárolom az adatbázisban. Mindkét komponenshez modern, reszponzív felületet készítettem, amely támogatja a hibajelzések és sikerüzenetek megjelenítését.

Belépési komponens

```
<!-- Itt jelenítem meg a belépési oldal címét. -->
<h2>Belépés</h2>

<!-- Itt hozom létre a bejelentkezési űrlap konténerét. -->
<div class="form-container">

  <!-- Itt kérem be a felhasználó email címét a belépéshez. -->
```

```
  <label>Email:</label>
  <input type="email" [(ngModel)]="adat.email">

  <!-- Itt veszem át a felhasználó jelszavát. -->
  <label>Jelszó:</label>
  <input type="password" [(ngModel)]="adat.jelszo">

  <!-- Itt hívom meg a belépés funkciót, amikor a gombra kattintok. -->
  <button (click)="belepes()">Belépés</button>

  <!-- Itt jelenítem meg a hibaüzenetet, ha a belépés sikertelen volt. -->
  <p class="hiba" *ngIf="hibauzenet">{{ hibauzenet }}</p>

</div>
```

```
/* ---- Alap elrendezés ---- */
.form-container {
  max-width: 400px;
  margin: 30px auto;
  padding: 25px;
  background: #ffffff;
  border-radius: 12px;
  box-shadow: 0 4px 12px rgba(0,0,0,0.1);
  display: flex;
  flex-direction: column;
  gap: 12px;
}

/* ---- Cím ---- */
h2 {
  text-align: center;
  margin-bottom: 15px;
  color: #2c3e50;
  font-size: 26px;
  font-weight: bold;
}

/* ---- Label ---- */
label {
  font-size: 15px;
  font-weight: 500;
  color: #34495e;
}

/* ---- Input mezők ---- */
input, textarea {
  width: 100%;
  padding: 10px 12px;
```

```
  font-size: 15px;
  border: 1px solid #dcdcdc;
  border-radius: 6px;
  outline: none;
  transition: 0.2s;
  background-color: #fafafa;
}

input:focus, textarea:focus {
  border-color: #3498db;
  background-color: #fff;
  box-shadow: 0 0 5px rgba(52,152,219,0.4);
}

/* ---- Gomb ---- */
button {
  margin-top: 10px;
  padding: 12px;
  width: 100%;
  background: #3498db;
  color: white;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  transition: 0.2s;
}

button:hover {
  background: #2980b9;
}

/* ---- Hibaüzenet ---- */
.hiba {
  margin-top: 8px;
  color: #e74c3c;
  font-weight: bold;
  text-align: center;
}

/* ---- Mobilbarát ---- */
@media (max-width: 480px) {
  .form-container {
    margin: 20px;
    padding: 20px;
  }

  h2 {
```

```
    font-size: 22px;
  }

  button {
    font-size: 15px;
  }
}
```

```
// Itt betöltöm az Express modult, hogy útvonalat hozhassak létre.
const express = require("express");
const router = express.Router();

// Itt betöltöm az adatbázis-kapcsolatot, hogy tudjak lekérdezést futtatni.
const kapcsolat = require("./kapcsolat");

// Itt kezelem a bejelentkezési kérést POST metódussal.
router.post("/", (req, res) => {

    // Itt kiveszem a beérkező emailt és jelszót a kérésből.
    const { email, jelszo } = req.body;

    // Itt ellenőrzöm, hogy a szükséges adatok meg vannak-e adva.
    if (!email || !jelszo) {
        return res.status(400).json({ error: "Hiányzó adatok!" });
    }

    // Itt állítom össze az SQL lekérdezést, amivel megkeresem a felhasználót.
    const sql = `SELECT * FROM felhasznalok WHERE email = ? AND jelszo = ?`;

    // Itt futtatom le a lekérdezést az adatbázisban.
    kapcsolat.query(sql, [email, jelszo], (err, rows) => {

        // Itt kezelem az esetleges adatbázishibát.
        if (err) {
            console.error("Hiba:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }

        // Itt ellenőrzöm, hogy létezik-e ilyen email–jelszó páros.
        if (rows.length === 0) {
            return res.status(401).json({ error: "Hibás email vagy jelszó!" });
        }

        // Itt küldöm vissza a sikeres válaszhoz tartozó adatokat.
        res.json({
            message: "Sikeres bejelentkezés!",
            user: rows[0]
        });
```

```
    });
});

// Itt exportálom az útvonalat, hogy a szerver használni tudja.
module.exports = router;
```

Regisztrációs komponens

```
<!-- Itt jelenítem meg a regisztrációs oldal címét. -->
<h2>Regisztráció</h2>

<!-- Itt hozom létre a regisztrációs űrlap konténerét. -->
<div class="form-container">

  <!-- Itt kérem be a felhasználó nevét a regisztrációhoz. -->
  <label>Név:</label>
  <input [(ngModel)]="adat.nev">

  <!-- Itt kérem be a felhasználó email címét. -->
  <label>Email:</label>
  <input type="email" [(ngModel)]="adat.email">

  <!-- Itt veszem át a felhasználó jelszavát. -->
  <label>Jelszó:</label>
  <input type="password" [(ngModel)]="adat.jelszo">

  <!-- Itt indítom el a regisztrációs adatokat elküldő függvényt. -->
  <button (click)="kuldes()">Regisztráció</button>

</div>
```

```
/* ---- Alap elrendezés ---- */
.form-container {
  max-width: 400px;
  margin: 30px auto;
  padding: 25px;
  background: #ffffff;
  border-radius: 12px;
  box-shadow: 0 4px 12px rgba(0,0,0,0.1);
  display: flex;
  flex-direction: column;
  gap: 12px;
}

/* ---- Cím ---- */
h2 {
  text-align: center;
  margin-bottom: 15px;
  color: #2c3e50;
```

```
  font-size: 26px;
  font-weight: bold;
}

/* ---- Label ---- */
label {
  font-size: 15px;
  font-weight: 500;
  color: #34495e;
}

/* ---- Input mezők ---- */
input, textarea {
  width: 100%;
  padding: 10px 12px;
  font-size: 15px;
  border: 1px solid #dcdcdc;
  border-radius: 6px;
  outline: none;
  transition: 0.2s;
  background-color: #fafafa;
}

input:focus, textarea:focus {
  border-color: #3498db;
  background-color: #fff;
  box-shadow: 0 0 5px rgba(52,152,219,0.4);
}

/* ---- Gomb ---- */
button {
  margin-top: 10px;
  padding: 12px;
  width: 100%;
  background: #3498db;
  color: white;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  transition: 0.2s;
}

button:hover {
  background: #2980b9;
}

/* ---- Hibaüzenet ---- */
```

```
.hiba {
  margin-top: 8px;
  color: #e74c3c;
  font-weight: bold;
  text-align: center;
}

/* ---- Mobilbarát ---- */
@media (max-width: 480px) {
  .form-container {
    margin: 20px;
    padding: 20px;
  }

  h2 {
    font-size: 22px;
  }

  button {
    font-size: 15px;
  }
}
```

```
// Itt betöltöm az Express modult, hogy létrehozhassam a regisztrációhoz szükséges útvonalat.
const express = require("express");
const router = express.Router();

// Itt töltöm be az adatbázis-kapcsolatot, hogy tudjak SQL parancsokat futtatni.
const kapcsolat = require("./kapcsolat");

// Itt kezelem a regisztráció POST kérését.
router.post("/", (req, res) => {

    // Itt kiveszem a név, email és jelszó értékeket a kliens által küldött adatokból.
    const { nev, email, jelszo } = req.body;

    // Itt ellenőrzöm, hogy minden szükséges mező ki lett-e töltve.
    if (!nev || !email || !jelszo) {
        return res.status(400).json({ error: "Hiányzó adatok!" });
    }

    // Itt állítom össze az SQL parancsot, hogy hozzáadjam az új felhasználót az adatbázishoz.
    const sql = `
        INSERT INTO felhasznalok (nev, email, jelszo)
        VALUES (?, ?, ?)
    `;
```

```
    // Itt futtatom le a beszúrási lekérdezést, és kezelem az eredményt.
    kapcsolat.query(sql, [nev, email, jelszo], (err, result) => {

        // Itt kezelem az esetleges adatbázishibát.
        if (err) {
            console.error("Hiba:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }

        // Itt küldöm vissza a sikeres regisztráció jelzését és az új felhasználó ID-ját.
        res.json({ message: "Sikeres regisztráció!", id: result.insertId });
    });
});

// Itt exportálom az útvonalat, hogy a szerver használni tudja.
module.exports = router;
```

Lakás komponens működése

A lakás komponensben megvalósítottam a lakók adatainak listázását, felvitelét, módosítását és törlését. Ehhez Angularos űrlapokat és állapotkezelést alkalmaztam, hogy a felhasználói élmény folyamatos és gördülékeny legyen. A műveletek a backend megfelelő REST API-jain keresztül történnek, amelyek SQL lekérdezéseket hajtanak végre az adatbázisban.

```
<!-- Itt jelenítem meg a lakók kezeléséhez tartozó fő konténert. -->
<div class="container mt-4">

  <!-- Itt írom ki az oldal címét. -->
  <h1 class="mb-4">Lakók</h1>

  <!-- Itt nyitom meg az új lakó felvételéhez tartozó űrlapot. -->
  <button class="btn btn-success mb-3" (click)="ujLakoMegnyit()">
    + Új lakó felvétele
  </button>

  <!-- Itt jelenítem meg az új lakó felvételéhez szükséges űrlapot, ha nem szerkesztési
módban vagyok. -->
  <div class="card mb-4" *ngIf="ujFormLathato && !szerkesztesMod">
    <div class="card-body">

      <!-- Itt jelzem, hogy új lakót viszek fel az adatbázisba. -->
      <h4 class="mb-3">Új lakó felvétele</h4>

      <!-- Itt küldöm be az új lakó adatait a komponens felé. -->
      <form (ngSubmit)="addLako()">

        <!-- Itt adom meg az új lakó felhasználó ID-ját. -->
        <div class="mb-2">
          <label>Felhasználó ID:</label>
```

```
    <input type="number" class="form-control"
           [(ngModel)]="ujLako.felhasznaloId" name="felhid" required>
  </div>

  <!-- Itt veszem fel a lakó nevét. -->
  <div class="mb-2">
    <label>Név:</label>
    <input type="text" class="form-control"
           [(ngModel)]="ujLako.nev" name="nev" required>
  </div>

  <!-- Itt kezelem a lakás alapadatait (lakásszám, emelet, telefonszám). -->
  <div class="row">
    <div class="col-md-4 mb-2">
      <label>Lakásszám:</label>
      <input type="text" class="form-control"
             [(ngModel)]="ujLako.lakasszam" name="lakas" required>
    </div>
    <div class="col-md-4 mb-2">
      <label>Emelet:</label>
      <input type="number" class="form-control"
             [(ngModel)]="ujLako.emelet" name="emelet">
    </div>
    <div class="col-md-4 mb-2">
      <label>Telefonszám:</label>
      <input type="text" class="form-control"
             [(ngModel)]="ujLako.telefonszam" name="tel">
    </div>
  </div>

  <!-- Itt adom meg a lakóhoz tartozó költséget. -->
  <div class="mb-2">
    <label>Költség:</label>
    <input type="number" class="form-control"
           [(ngModel)]="ujLako.koltseg" name="koltseg">
  </div>

  <!-- Itt írok be esetleges megjegyzést a lakóhoz. -->
  <div class="mb-2">
    <label>Megjegyzés:</label>
    <input type="text" class="form-control"
           [(ngModel)]="ujLako.megjegyzes" name="megj">
  </div>

  <!-- Itt küldöm be az új lakó adatait, vagy visszalépek. -->
  <button type="submit" class="btn btn-success mt-2">Lakó felvétele</button>
  <button type="button" class="btn btn-secondary mt-2 ms-2"
          (click)="ujFormLathato = false">
    Mégse
```

```
        </button>

      </form>

    </div>
  </div>

  <!-- Itt jelenítem meg a lakó módosításához tartozó űrlapot, ha szerkesztési mód aktív. -
->
  <div class="card mb-4" *ngIf="szerkesztesMod">
    <div class="card-body">

      <!-- Itt jelzem, hogy egy meglévő lakó adatait módosítom. -->
      <h4 class="mb-3">Lakó módosítása</h4>

      <!-- Itt küldöm el a módosított adatokat. -->
      <form (ngSubmit)="mentModositas()">

        <!-- Itt módosítom a lakó nevét. -->
        <div class="mb-2">
          <label>Név:</label>
          <input type="text" class="form-control"
                 [(ngModel)]="modositando.nev" name="nev_m" required>
        </div>

        <!-- Itt szerkesztem a lakó felhasználó ID-ját. -->
        <div class="mb-2">
          <label>Felhasználó ID:</label>
          <input type="number" class="form-control"
                 [(ngModel)]="modositando.felhasznaloId" name="felhid_m" required>
        </div>

        <!-- Itt adom meg a lakásszámot módosítás esetén. -->
        <div class="mb-2">
          <label>Lakásszám:</label>
          <input type="text" class="form-control"
                 [(ngModel)]="modositando.lakasszam" name="lakas_m">
        </div>

        <!-- Itt módosítom az emeletet. -->
        <div class="mb-2">
          <label>Emelet:</label>
          <input type="number" class="form-control"
                 [(ngModel)]="modositando.emelet" name="emelet_m">
        </div>

        <!-- Itt adom meg a lakó telefonszámát. -->
        <div class="mb-2">
          <label>Telefonszám:</label>
```

```
          <input type="text" class="form-control"
                 [(ngModel)]="modositando.telefonszam" name="tel_m">
        </div>

        <!-- Itt szerkesztem a lakó költségét. -->
        <div class="mb-2">
          <label>Költség:</label>
          <input type="number" class="form-control"
                 [(ngModel)]="modositando.koltseg" name="koltseg_m">
        </div>

        <!-- Itt módosítom a lakóhoz tartozó megjegyzést. -->
        <div class="mb-2">
          <label>Megjegyzés:</label>
          <input type="text" class="form-control"
                 [(ngModel)]="modositando.megjegyzes" name="megj_m">
        </div>

        <!-- Itt mentem vagy elvetem a módosításokat. -->
        <button type="submit" class="btn btn-success mt-2">Módosítás mentése</button>
        <button type="button" class="btn btn-secondary mt-2 ms-2"
                (click)="szerkesztesMod = false">
          Mégse
        </button>

      </form>

    </div>
  </div>

  <!-- Itt jelenítem meg a lakókat egy kártyás elrendezésben. -->
  <div class="row">
    <div class="col-md-4 mb-3" *ngFor="let l of lakok">
      <div class="card shadow-sm h-100">
        <div class="card-body">

          <!-- Itt mutatom a lakó nevét címként. -->
          <h5 class="card-title">{{ l.nev }}</h5>

          <!-- Itt jelenítem meg a lakó lakásadatokat és megjegyzést. -->
          <p><strong>Lakásszám:</strong> {{ l.lakasszam }}</p>
          <p><strong>Emelet:</strong> {{ l.emelet }}</p>
          <p><strong>Telefonszám:</strong> {{ l.telefonszam }}</p>
          <p><strong>Költség:</strong> {{ l.koltseg }} Ft</p>
          <p><strong>Megjegyzés:</strong> {{ l.megjegyzes }}</p>

          <!-- Itt indítom el a lakó szerkesztését vagy törlését. -->
          <button class="btn btn-primary me-2"
(click)="szerkesztLako(l)">Módosítás</button>
```

```
          <button class="btn btn-danger" (click)="torolLako(l.id)">Törlés</button>

        </div>
      </div>
    </div>
  </div>

</div>
```

```
/* Itt adok egy alap felső margót a teljes tartalomnak. */
.container {
  margin-top: 20px;
}

/* Itt állítom be a kártyák megjelenését, hátterét és árnyékát. */
.card {
  background: #f8f9fa;
  border-radius: 12px;
  border: 1px solid #ddd;
  box-shadow: 0 2px 8px rgba(0,0,0,0.08);
  transition: transform 0.15s ease, box-shadow 0.15s ease;
}

/* Itt érem el, hogy hover esetén a kártya kicsit kiemelkedjen. */
.card:hover {
  transform: translateY(-4px);
  box-shadow: 0 6px 20px rgba(0,0,0,0.15);
}

/* Itt adok kényelmes belső margót a kártyák tartalmának. */
.card-body {
  padding: 20px;
}

/* Itt formázom a kártyák címsorát. */
.card-title {
  font-weight: 600;
  font-size: 1.4rem;
  margin-bottom: 15px;
  color: #333;
}

/* Itt szabályozom a form mezők stílusát, hogy egységes legyen. */
form .form-control {
  border-radius: 8px;
  padding: 10px;
  font-size: 1rem;
}
```

```
/* Itt állítom be a címkék (label) alap megjelenését. */
label {
  font-weight: 500;
  margin-bottom: 4px;
}

/* Itt teszem egységesen lekerekítettebbé a gombokat. */
button {
  border-radius: 8px !important;
  padding: 8px 16px;
}

/* Itt formázom a zöld (siker) gombot. */
.btn-success {
  background-color: #28a745;
  border-color: #28a745;
  font-weight: 500;
}

/* Itt adok hover hatást a siker gombnak. */
.btn-success:hover {
  background-color: #218838;
}

/* Itt formázom a kék (elsődleges) gombot. */
.btn-primary {
  background-color: #007bff;
  border-color: #007bff;
  font-weight: 500;
}

/* Itt adom meg a hover színét a kék gombnak. */
.btn-primary:hover {
  background-color: #0069d9;
}

/* Itt formázom a piros (veszély) gombot. */
.btn-danger {
  background-color: #dc3545;
  border-color: #dc3545;
  font-weight: 500;
}

/* Itt adok hover színt a piros gombnak. */
.btn-danger:hover {
  background-color: #c82333;
}
```

```
/* Itt állítom be a másodlagos gomb általános stílusát. */
.btn-secondary {
  font-weight: 500;
  border-radius: 8px !important;
}

/* Itt szabályozom az .mb-3 elem alsó margóját. */
.mb-3 {
  margin-bottom: 1.2rem !important;
}

/* Itt adok nagyobb felső margót az .mt-4 elemnek. */
.mt-4 {
  margin-top: 1.5rem !important;
}
```

```
// Itt betöltöm az Express modult, hogy létrehozhassam a lakókat kezelő útvonalakat.
const express = require("express");
const router = express.Router();

// Itt töltöm be az adatbázis kapcsolatot, hogy SQL lekérdezéseket futtathassak.
const kapcsolat = require("./kapcsolat");


// Itt kezelem a GET kérést, amivel lekérem az összes lakót.
router.get("/", (req, res) => {
    const sql = "SELECT * FROM lakok";

    // Itt futtatom le a lekérdezést, és visszaküldöm az eredményt.
    kapcsolat.query(sql, (err, result) => {
        if (err) {
            console.error("Hiba a lekérdezés során:", err);
            return res.status(500).json({ error: "Adatbázis hiba" });
        }
        res.json(result);
    });
});


// Itt kezelem az új lakó felvételét POST kérés segítségével.
router.post("/", (req, res) => {

    // Itt bontom ki az érkező adatokat a request törzséből.
    const {
        felhasznaloId,
        nev,
        lakasszam,
        emelet,
        telefonszam,
```

```
        koltseg,
        megjegyzes
    } = req.body;

    // Itt állítom össze az SQL parancsot az új lakó adatainak rögzítéséhez.
    const sql = `
        INSERT INTO lakok
        (felhasznaloId, nev, lakasszam, emelet, telefonszam, koltseg, megjegyzes)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    `;

    // Itt készítem elő az adatokat tömbben a beszúráshoz.
    const adatok = [
        felhasznaloId,
        nev,
        lakasszam,
        emelet,
        telefonszam,
        koltseg,
        megjegyzes
    ];

    // Itt hajtom végre a beszúrási lekérdezést.
    kapcsolat.query(sql, adatok, (err, result) => {
        if (err) {
            console.error("Hiba az új lakó felvételekor:", err);
            return res.status(500).json({ error: "Adatbázis hiba" });
        }

        res.json({ message: "Lakó sikeresen felvéve!", id: result.insertId });
    });
});


// Itt kezelem a PUT kérést, amely egy meglévő lakó adatainak módosítására szolgál.
router.put("/:id", (req, res) => {
    const id = req.params.id;

    // Itt szedem ki a módosított adatokat a kérésből.
    const {
        felhasznaloId,
        nev,
        lakasszam,
        emelet,
        telefonszam,
        koltseg,
        megjegyzes
    } = req.body;
```

```
    // Itt állítom össze az SQL parancsot, amely frissíti a lakó adatait.
    const sql = `
        UPDATE lakok SET
        felhasznaloId = ?,
        nev = ?,
        lakasszam = ?,
        emelet = ?,
        telefonszam = ?,
        koltseg = ?,
        megjegyzes = ?
        WHERE id = ?
    `;

    // Itt rendezem egy tömbbe az adatokat a frissítéshez.
    const adatok = [
        felhasznaloId,
        nev,
        lakasszam,
        emelet,
        telefonszam,
        koltseg,
        megjegyzes,
        id
    ];

    // Itt hajtom végre a módosító SQL parancsot.
    kapcsolat.query(sql, adatok, (err, result) => {
        if (err) {
            console.error("Hiba módosítás közben:", err);
            return res.status(500).json({ error: "Adatbázis hiba" });
        }

        // Itt kezelem azt az esetet, ha nem létező lakót próbálok módosítani.
        if (result.affectedRows === 0) {
            return res.status(404).json({ message: "Nincs ilyen lakó!" });
        }

        res.json({ message: "Lakó módosítva!" });
    });
});


// Itt kezelem a lakó törlését DELETE kérés segítségével.
router.delete("/:id", (req, res) => {
    const id = req.params.id;

    const sql = "DELETE FROM lakok WHERE id = ?";

    // Itt futtatom a törlési lekérdezést.
```

```
    kapcsolat.query(sql, [id], (err, result) => {
        if (err) {
            console.error("Hiba törlés közben:", err);
            return res.status(500).json({ error: "Adatbázis hiba" });
        }

        // Itt jelzem, ha nem található a törölni kívánt lakó.
        if (result.affectedRows === 0) {
            return res.status(404).json({ message: "Nincs ilyen lakó!" });
        }

        res.json({ message: "Lakó törölve!" });
    });
});


// Itt exportálom az útvonalakat, hogy a szerver használni tudja.
module.exports = router;
```

Bejelentések komponens működése

A bejelentések komponensben lehetőséget biztosítottam új bejelentés létrehozására, a meglévők szerkesztésére, valamint törlésére. Ehhez hasonlóan a lakók kezeléséhez, Angular űrlapokat, API-hívásokat és szerkesztési módot használtam. A backend oldalon elkészítettem az ehhez szükséges GET, POST, PUT és DELETE útvonalakat. A komponens kártyás, reszponzív megjelenést kapott, amely áttekinthetővé teszi a bejelentéseket.

```
<!-- Itt hozom létre a teljes oldal konténerét, ahol a bejelentéseket kezelem. -->
<div class="container mt-4">

  <!-- Itt jelenítem meg az oldal címét. -->
  <h1 class="mb-4">Bejelentések</h1>

  <!-- Itt jelenik meg a gomb, amivel megnyitom az új bejelentés űrlapot. -->
  <button class="btn btn-success mb-3" (click)="ujBejelentesMegnyit()">
    + Új bejelentés
  </button>

  <!-- Itt teszem láthatóvá az Új bejelentés űrlapot, ha nem szerkesztési módban vagyok. --
>
  <div class="card mb-4" *ngIf="ujFormLathato && !szerkesztesMod">
    <div class="card-body">

      <!-- Itt adom meg, hogy új bejelentést hozok létre. -->
      <h4 class="mb-3">Új bejelentés</h4>

      <!-- Itt küldöm be az új bejelentés adatait a komponensbe. -->
      <form (ngSubmit)="addBejelentes()">
```

```
      <!-- Itt viszem fel az új bejelentéshez tartozó lakó ID-t. -->
      <div class="mb-2">
        <label>Lakó ID:</label>
        <input type="number" class="form-control"
               [(ngModel)]="ujBejelentes.lakoId" name="lakoid" required>
      </div>

      <!-- Itt adom meg a bejelentés típusát. -->
      <div class="mb-2">
        <label>Típus:</label>
        <input type="text" class="form-control"
               [(ngModel)]="ujBejelentes.tipus" name="tipus" required>
      </div>

      <!-- Itt írom le részletesen a bejelentés tartalmát. -->
      <div class="mb-2">
        <label>Leírás:</label>
        <textarea class="form-control"
               [(ngModel)]="ujBejelentés.leiras" name="leiras" required></textarea>
      </div>

      <!-- Itt küldöm el az űrlapot, vagy lépek vissza. -->
      <button type="submit" class="btn btn-success mt-2">Hozzáadás</button>
      <button type="button" class="btn btn-secondary ms-2 mt-2"
              (click)="ujFormLathato = false">Mégse</button>

    </form>

  </div>
</div>

<!-- Itt nyitom meg a módosítás űrlapot, ha egy bejelentést szerkesztek. -->
<div class="card mb-4" *ngIf="szerkesztesMod">
  <div class="card-body">

    <!-- Itt jelzem, hogy egy meglévő bejelentést módosítok. -->
    <h4 class="mb-3">Bejelentés módosítása</h4>

    <!-- Itt kezelem a módosított adatok mentését. -->
    <form (ngSubmit)="mentModositas()">

      <!-- Itt tudom átírni a bejelentés lakó ID-jét. -->
      <div class="mb-2">
        <label>Lakó ID:</label>
        <input type="number" class="form-control"
               [(ngModel)]="modositando.lakoId" name="lakoid_m" required>
      </div>
```

```
      <!-- Itt módosítom a bejelentés típusát. -->
      <div class="mb-2">
        <label>Típus:</label>
        <input type="text" class="form-control"
               [(ngModel)]="modositando.tipus" name="tipus_m" required>
      </div>

      <!-- Itt szerkesztem a bejelentés leírását. -->
      <div class="mb-2">
        <label>Leírás:</label>
        <textarea class="form-control"
               [(ngModel)]="modositando.leiras" name="leiras_m" required></textarea>
      </div>

      <!-- Itt mentem a módosításokat vagy lépek vissza. -->
      <button type="submit" class="btn btn-success mt-2">Mentés</button>
      <button type="button" class="btn btn-secondary ms-2 mt-2"
              (click)="szerkesztesMod = false">Mégse</button>

    </form>

  </div>
</div>

<!-- Itt jelenítem meg az összes bejelentést kártyák formájában. -->
<div class="row">
  <div class="col-md-4 mb-3" *ngFor="let b of bejelentesek">

    <!-- Itt hozom létre a bejelentés kártyáját. -->
    <div class="card shadow-sm h-100">
      <div class="card-body">

        <!-- Itt mutatom a bejelentés azonosítóját. -->
        <h5 class="card-title">Bejelentés #{{ b.id }}</h5>

        <!-- Itt jelenítem meg a bejelentéshez tartozó adatokat. -->
        <p><strong>Lakó ID:</strong> {{ b.lakoId }}</p>
        <p><strong>Típus:</strong> {{ b.tipus }}</p>
        <p><strong>Leírás:</strong> {{ b.leiras }}</p>

        <!-- Itt indítom el a szerkesztést vagy törlést a kiválasztott bejelentésre. -->
        <button class="btn btn-primary me-2" (click)="szerkeszt(b)">Módosítás</button>
        <button class="btn btn-danger" (click)="torol(b.id)">Törlés</button>

      </div>
    </div>

  </div>
</div>
```

```
</div>
```

```
/* Itt adok felső margót a teljes konténernek. */
.container {
  margin-top: 20px;
}

/* Itt alakítom át a sorokat rácsos elrendezéssé 3 oszloppal. */
.row {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}

/* Itt állítom át a rácsot 2 oszloposra közepes kijelzőméret alatt. */
@media (max-width: 992px) {
  .row {
    grid-template-columns: repeat(2, 1fr);
  }
}

/* Itt teszem egyoszlopossá a kártyákat kisebb kijelzőkön. */
@media (max-width: 768px) {
  .row {
    grid-template-columns: 1fr;
  }
}

/* Itt állítom be a kártyák alapstílusát, árnyékát és átmeneteit. */
.card {
  background: #f8f9fa;
  border-radius: 12px;
  border: 1px solid #ddd;
  box-shadow: 0 2px 8px rgba(0,0,0,0.08);
  transition: transform 0.15s ease, box-shadow 0.15s ease;
  height: 100%;
}

/* Itt adom meg a hover animációt, ami kiemeli a kártyát. */
.card:hover {
  transform: translateY(-4px);
  box-shadow: 0 6px 20px rgba(0,0,0,0.15);
}

/* Itt szabályozom a kártyák belső margóját. */
.card-body {
```

```
  padding: 18px;
}

/* Itt formázom a kártyacímeket megjelenés és olvashatóság miatt. */
.card-title {
  font-weight: 600;
  font-size: 1.2rem;
  margin-bottom: 12px;
  color: #333;
}

/* Itt állítom be az inputmezők stílusát az űrlapokon. */
form .form-control {
  border-radius: 8px;
  padding: 10px;
  font-size: 1rem;
}

/* Itt állítom be az űrlap címkéinek megjelenését. */
label {
  font-weight: 500;
  margin-bottom: 4px;
}

/* Itt egységesítem a gombok alakját és méretét. */
button {
  border-radius: 8px !important;
  padding: 8px 16px;
}

/* Itt formázom a siker (zöld) gombot. */
.btn-success {
  background-color: #28a745;
  border-color: #28a745;
  font-weight: 500;
}

/* Itt adom meg a hover állapot színét a zöld gombnak. */
.btn-success:hover {
  background-color: #218838;
}

/* Itt formázom az elsődleges (kék) gombot. */
.btn-primary {
  background-color: #007bff;
  border-color: #007bff;
  font-weight: 500;
}
```

```
/* Itt állítom be a kék gomb hover színét. */
.btn-primary:hover {
  background-color: #0069d9;
}

/* Itt formázom a veszély (piros) gombot. */
.btn-danger {
  background-color: #dc3545;
  border-color: #dc3545;
  font-weight: 500;
}

/* Itt változtatom meg a piros gomb színét hover esetén. */
.btn-danger:hover {
  background-color: #c82333;
}

/* Itt adom meg a másodlagos gomb általános stílusát. */
.btn-secondary {
  font-weight: 500;
  border-radius: 8px !important;
}

/* Itt állítom be az alsó margót a .mb-3 elemeknél. */
.mb-3 {
  margin-bottom: 1.2rem !important;
}

/* Itt növelem meg a felső margót az .mt-4 osztálynál. */
.mt-4 {
  margin-top: 1.5rem !important;
}
```

```
// Itt betöltöm az Express modult, hogy létrehozhassam a bejelentéseket kezelő útvonalakat.
const express = require("express");
const router = express.Router();

// Itt kapcsolódom az adatbázishoz az SQL lekérdezések végrehajtásához.
const kapcsolat = require("./kapcsolat");


// Itt kezelem a bejelentések listázását GET metódussal.
router.get("/", (req, res) => {

    // Itt állítom össze az SQL lekérdezést a bejelentések kilistázásához.
    const sql = `
        SELECT id, lakoId, tipus, leiras
        FROM bejelentesek
```

```
    `;

    // Itt futtatom le a lekérdezést, és visszaküldöm az eredményt.
    kapcsolat.query(sql, (err, result) => {
        if (err) {
            console.error("Hiba a bejelentések lekérése közben:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }
        res.json(result);
    });
});


// Itt kezelem az új bejelentés felvételét POST kérésből.
router.post("/", (req, res) => {

    // Itt veszem ki a bejelentéshez szükséges adatokat a kérésből.
    const { lakoId, tipus, leiras } = req.body;

    // Itt ellenőrzöm, hogy minden kötelező mező ki van-e töltve.
    if (!lakoId || !tipus || !leiras) {
        return res.status(400).json({ error: "Hiányzó kötelező adatok!" });
    }

    // Itt készítem elő az SQL parancsot az új bejelentés rögzítésére.
    const sql = `
        INSERT INTO bejelentesek (lakoId, tipus, leiras)
        VALUES (?, ?, ?)
    `;

    // Itt futtatom le az INSERT lekérdezést.
    kapcsolat.query(sql, [lakoId, tipus, leiras], (err, result) => {
        if (err) {
            console.error("Adatbázis hiba:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }

        // Itt küldöm vissza a sikeres felvétel válaszát és az új rekord ID-ját.
        res.json({
            message: "Bejelentés sikeresen felvéve!",
            id: result.insertId
        });
    });
});


// Itt kezelem egy meglévő bejelentés adatainak frissítését PUT metódussal.
router.put("/:id", (req, res) => {

    // Itt veszem ki az URL-ből az id-t.
```

```
    const id = req.params.id;

    // Itt szedem ki a frissített adatokat a kérésből.
    const { lakoId, tipus, leiras } = req.body;

    // Itt állítom össze az SQL módosító parancsot.
    const sql = `
        UPDATE bejelentesek
        SET lakoId = ?, tipus = ?, leiras = ?
        WHERE id = ?
    `;

    // Itt futtatom le a módosítást az adatbázisban.
    kapcsolat.query(sql, [lakoId, tipus, leiras, id], (err, result) => {
        if (err) {
            console.error("Hiba módosítás közben:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }

        res.json({ message: "Bejelentés módosítva!" });
    });
});


// Itt kezelem egy bejelentés törlését DELETE metódussal.
router.delete("/:id", (req, res) => {

    // Itt veszem ki a törlendő bejelentés azonosítóját.
    const id = req.params.id;

    // Itt készítem elő az SQL törlő parancsot.
    const sql = "DELETE FROM bejelentesek WHERE id = ?";

    // Itt futtatom le a törlést.
    kapcsolat.query(sql, [id], (err, result) => {
        if (err) {
            console.error("Hiba törlés közben:", err);
            return res.status(500).json({ error: "Adatbázis hiba!" });
        }

        res.json({ message: "Bejelentés törölve!" });
    });
});

// Itt exportálom az útvonalakat, hogy a szerver is használhassa őket.
module.exports = router;
```

server.js fájl létrehozása

A server.js fájlban hoztam létre az Express alapú szervert, amely a teljes backend működését biztosítja. Bekapcsoltam a CORS-t és a JSON feldolgozást annak érdekében, hogy az Angular frontend zökkenőmentesen kommunikálhasson a szerverrel. Ezután regisztráltam az összes útvonalkezelőt, így külön kezeltem a lakók, bejelentések, bejelentkezés és regisztráció API-jait.

```
// Itt betöltöm az Express modult, hogy létrehozhassam a szerver alkalmazást.
const express = require("express");
// Itt engedélyezem a CORS-t, hogy az Angular alkalmazás kommunikálni tudjon a backenddel.
const cors = require("cors");

const app = express();
const PORT = 3000;

// Itt bekapcsolom a CORS-t, hogy külső domainekről is tudjak adatot fogadni.
app.use(cors());
// Itt állítom be, hogy a bejövő JSON adatokat automatikusan felismerje és feldolgozza a
szerver.
app.use(express.json());


// Itt csatolom be a lakókat kezelő útvonalakat.
const lakasRoutes = require("./lakok");
app.use("/api/lakok", lakasRoutes);

// Itt töltöm be a bejelentéseket kezelő útvonalakat.
const bejelentesekRoutes = require("./bejelentesek");
app.use("/api/bejelentesek", bejelentesekRoutes);

// Itt adom hozzá a regisztrációs funkció backend útvonalait.
const regisztracioRoutes = require("./regisztracio");
app.use("/api/regisztracio", regisztracioRoutes);

// Itt csatolom be a bejelentkezést kezelő útvonalakat.
const bejelentkezesRoutes = require("./bejelentkezes");
app.use("/api/bejelentkezes", bejelentkezesRoutes);

// Itt indítom el a szervert a megadott porton.
app.listen(PORT, () => {
    console.log(`Szerver fut: http://localhost:${PORT}`);
});
```

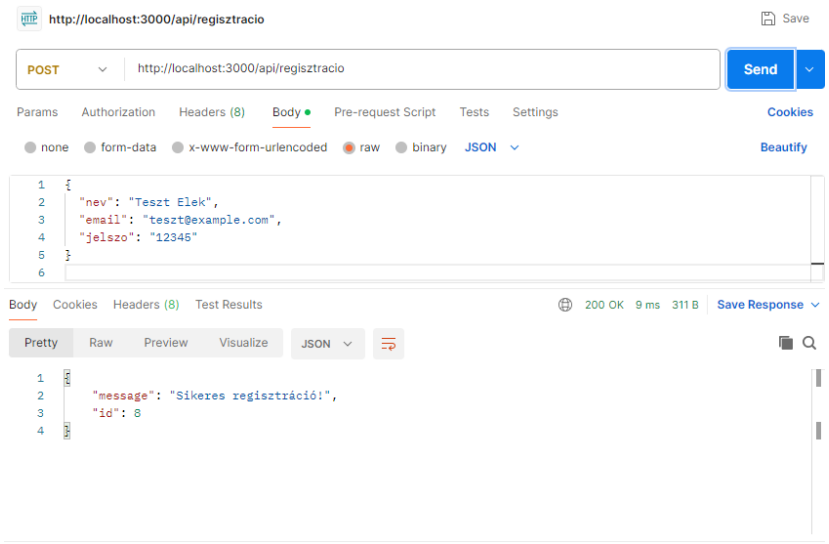
app.router.ts fájl létrehozása

Az Angular útvonalkezelőjét úgy állítottam be, hogy minden fontos komponens külön útvonalon legyen elérhető. Létrehoztam a főoldal, a lakók, a bejelentések, a regisztráció és a belépés útvonalait. Az üres útvonalat automatikusan átirányítottam a főoldalra, ezzel biztosítva az alkalmazás logikus indulását.

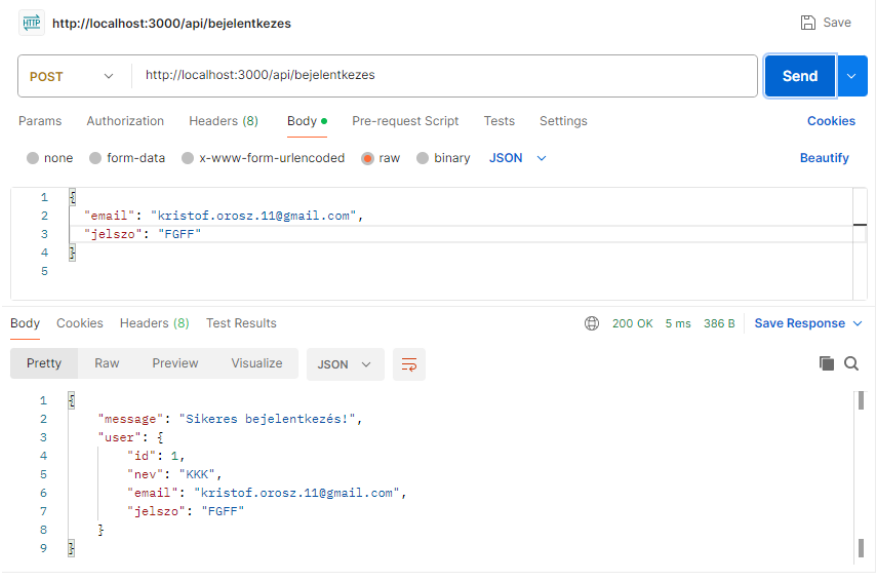
```
// Itt importálom az Angular útvonalkezelőt, hogy létrehozhassam az alkalmazás
navigációját.
import { Routes } from '@angular/router';

// Itt betöltöm a főoldal komponensét, hogy útvonalat hozzak hozzá.
import { FooldalComponent} from './fooldal/fooldal.component';
// Itt importálom a bejelentések kezelő komponensét.
import { BejelentesekComponent } from './bejelentesek/bejelentesek.component';
// Itt csatolom be a regisztráció nézetét.
import { RegisztracioComponent } from './regisztracio/regisztracio.component';
// Itt töltöm be a belépési oldalt.
import { BelepesComponent } from './belepes/belepes.component';
// Itt importálom a lakók kezelésére szolgáló komponenst.
import { LakasComponent } from './lakas/lakas.component';


// Itt hozom létre az alkalmazás teljes útvonal-listáját.
export const routes: Routes = [

  // Itt átirányítom az üres útvonalat a főoldalra.
  { path: '', redirectTo: 'fooldal', pathMatch: 'full' },

  // Itt jelenítem meg a belépési oldalt.
  { path: 'belepes', component: BelepesComponent},

  // Itt jelenik meg a főoldal komponense.
  { path: 'fooldal', component: FooldalComponent},

  // Itt érem el a lakók kezelésére szolgáló oldalt.
  { path: 'lakas', component: LakasComponent},

  // Itt jelenítem meg a bejelentések oldalát.
  { path: 'bejelentesek', component: BejelentesekComponent},

  // Itt érem el a regisztráció nézetet.
  { path: 'regisztracio', component: RegisztracioComponent }
];
```

Postman tesztelés

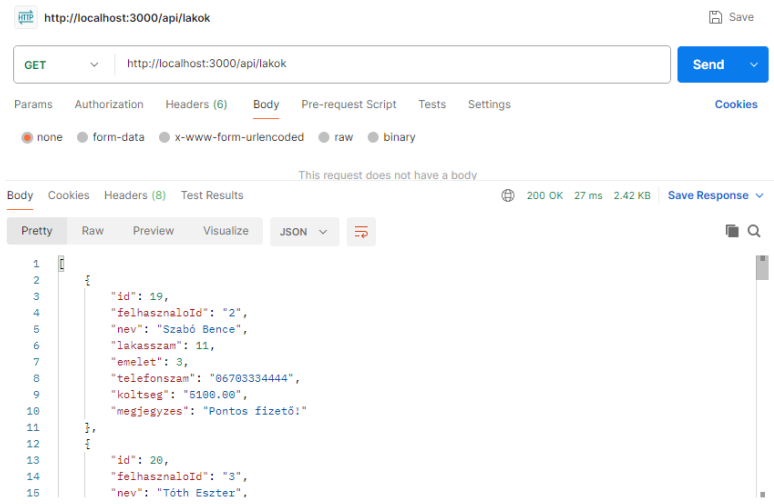
A regisztrációs végpont működését Postman segítségével teszteltem. Helyes bemeneti adatokkal a rendszer sikeresen elmentette az új felhasználót, és visszaadta a generált azonosítót.



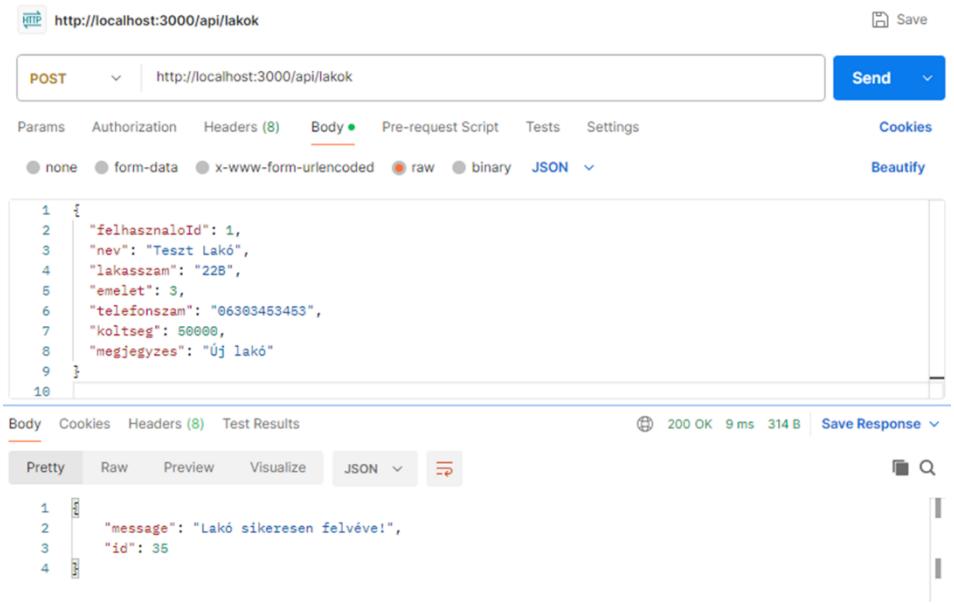
A bejelentkezési funkciót valós, helyes felhasználói adatokkal teszteltem. A szerver sikeresen beazonosította a felhasználót és visszaküldte a hozzá tartozó adatokat.



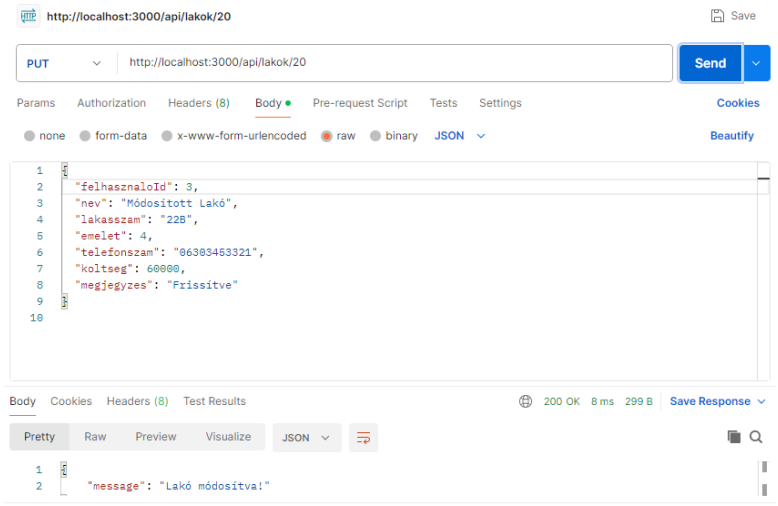
A lakók listázását végző GET végpontot is teszteltem Postmanben. A kérés elküldése után a szerver helyesen visszaküldte az adatbázisban tárolt összes lakó adatait JSON formátumban.



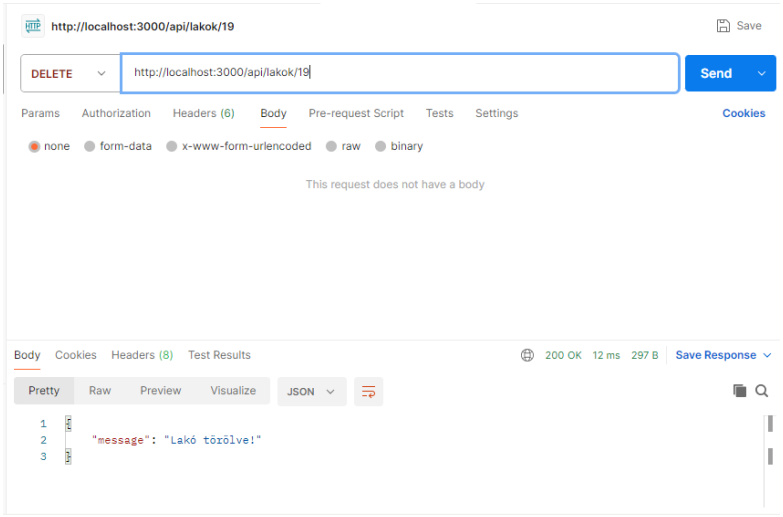
A lakó felvételére szolgáló végpontot sikeresen teszteltem. A rendszer a megadott adatok alapján elmentette az új lakót, és a válaszban visszaküldte az adatbázisban létrejött új azonosítót.



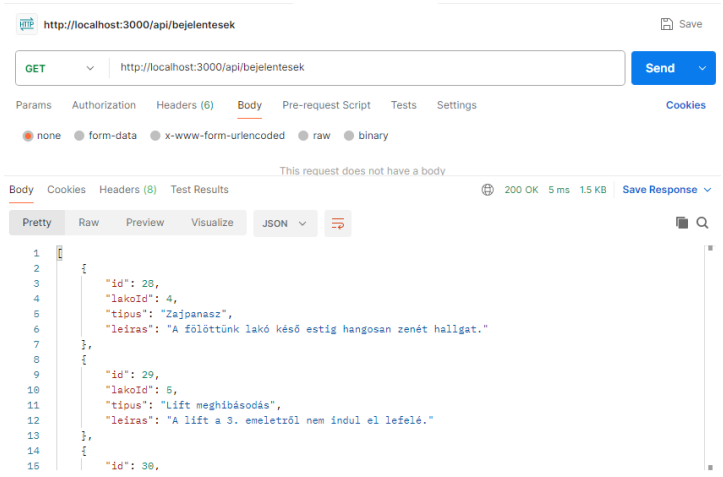
A lakók módosítását célzó tesztet helyesen kitöltött adatokkal futtattam le. A szerver megfelelően frissítette a megadott lakó adatait, és visszajelezte a módosítás sikerességét.



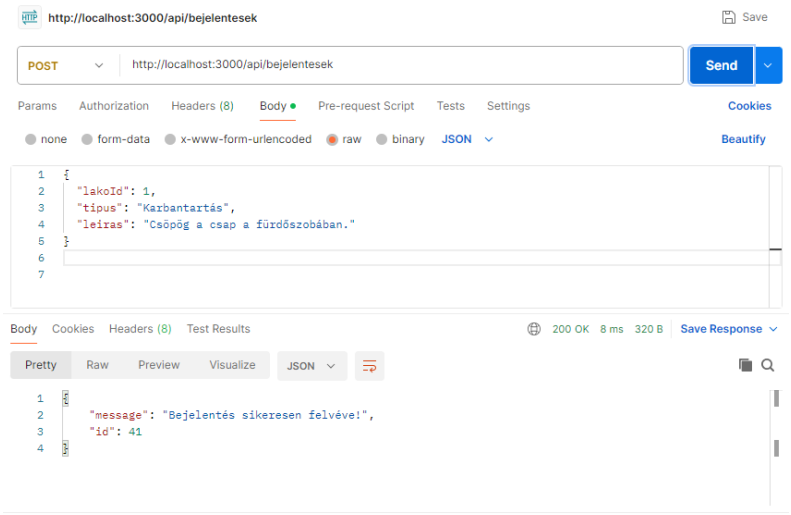
A törlési művelet Postmanben végzett tesztje sikeresen lefutott. A megadott azonosító alapján a rendszer eltávolította a lakót az adatbázisból, majd visszaigazolta a törlés végrehajtását.



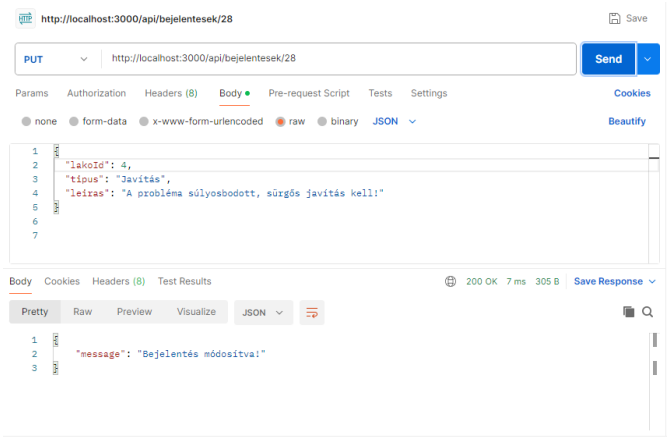
A bejelentések lekérdezését végző GET végpontot szintén teszteltem. A szerver hibamentesen visszaadta az összes bejelentést az adatbázisból, áttekinthető JSON formátumban.



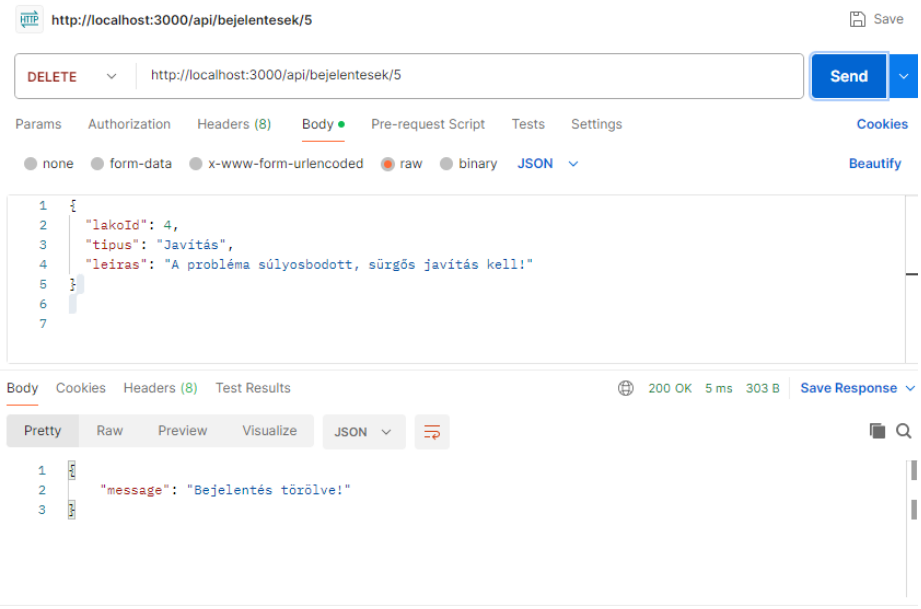
A bejelentések felvételére szolgáló végpont tesztje helyesen futott le. A beküldött adatok alapján a rendszer sikeresen rögzítette a bejelentést, és visszaküldte az új rekord azonosítóját.



A bejelentések módosítását helyes adatokkal teszteltem, és a szerver sikeresen frissítette az adott bejelentés adatait.



A bejelentések törlésének tesztje szintén sikeresen lefutott. A megadott id alapján a rendszer eltávolította a bejelentést, és visszaigazolta a műveletet.



Jogosultságellenőrzés

A rendszer a felhasználó bejelentkezési állapotát az Angular keretrendszer segítségével vizsgálja. A navigációs sávban található menüpontok megjelenése dinamikusan, a felhasznalo szolgáltatásban elérhető isBejelentkezve() függvény visszatérési értéke alapján történik. Amennyiben a felhasználó nincs bejelentkezve, akkor a rendszer kizárólag a bejelentkezési és regisztrációs hivatkozásokat jeleníti meg. Ezeket a menüpontokat a kódban a *ngIf="!felhasznalo.isBejelentkezve()" feltétel biztosítja, amely gondoskodik arról, hogy a felhasználó csak a bejelentkezést megelőző funkciókhoz férhessen hozzá. Bejelentkezett állapotban a felhasznalo.isBejelentkezve() igaz értéke miatt már a jogosultságot igénylő menüpontok jelennek meg, mint például a Lakások és a Bejelentések oldalak. Ebben az esetben a rendszer megjeleníti a kijelentkezés lehetőségét is, amely a felhasznalo.kijelentkeztet() metódust hívja meg.

```
<!-- A felső navigációs sáv fő konténere -->
<nav class="felsosav">

  <!-- A bal oldali menürész, ahol a logó fixen megjelenik -->
  <div class="menu-bal">
    <span class="logo">Társasházkezelő</span>
  </div>

  <!-- A jobb oldali menü, ahol a jogosultság alapján változó elemek jelennek meg -->
  <div class="menu-jobb">
```

```
    <!-- A "Bejelentkezés" menüpont csak akkor jelenik meg, ha a felhasználó nincs
bejelentkezve -->
    <a routerLink="/belepes" *ngIf="!felhasznalo.isBejelentkezve()">Bejelentkezés</a>

    <!-- A "Regisztráció" menüpont szintén csak kijelentkezett állapotban látható -->
    <a routerLink="/regisztracio" *ngIf="!felhasznalo.isBejelentkezve()">Regisztráció</a>

    <!-- A "Lakások" oldal elérése csak bejelentkezett felhasználóknak engedélyezett -->
    <a routerLink="/lakas" *ngIf="felhasznalo.isBejelentkezve()">Lakások</a>

    <!-- A "Bejelentések" oldal is csak akkor jelenik meg, ha a felhasználó be van
jelentkezve -->
    <a routerLink="/bejelentesek" *ngIf="felhasznalo.isBejelentkezve()">Bejelentések</a>

    <!-- A kijelentkezés gomb csak bejelentkezett állapotban látható, és meghívja a
kijelentkeztetési metódust -->
    <a href="#" (click)="felhasznal
```