

# Rezsi App

## Szoftveralkalmazás dokumentáció

Balogh Krisztián, Nagy István, Szilágyi Attila

5 0613 12 03 Szoftverfejlesztő és -tesztelő

2024.04.30

## Tartalomjegyzék

1. Szoftver célja	3
2. Fejlesztői dokumentáció	4
Deployment	4
A fejlesztéshez használt eszközök	4
Az alkalmazás felépítése	4
Könyvtárstruktúra	4
Backend	5
backend/routes	5
backend/routes/API	6
backend/routes/html	6
backend/routes/css	6
backend/routes/js	6
backend/app.js	6
backend/database.js	7
Frontend	9
frontend/css	9
frontend/html	9
frontend/images	10
frontend/js	10
Az alkalmazás backend oldali logikája	10
API végpontok	12
Az alkalmazás frontend oldali logikája	21
frontend/html	21
frontend/js	26
Adatbázis modell	42
3. Felhasználói dokumentáció	44
Alkalmazás célja	44
Rezsi App nyitóoldal	45
Bejelentkezés	45
Regisztráció	46
Kezdőlap	46
Gázóra állások	47
Villanyóra állások	50
Kijelentkezés	51
4. Tesztelési dokumentáció	51
Weboldalon megjelenő hibaüzenetek	52
Reszponzív nézetek	54
Fejlesztési lehetőségek	54
5. Irodalomjegyzék	55

## 1. Szoftver célja

A választott szoftver egy a hétköznapi életben sokak számára felmerülő igény kielégítését tűzte ki céljául, melynek egy kezdeményezését valósítottuk meg a Rezsi App-ban. Ez az igény pedig a rezsi költség számításának és nyilvántartásának alapjául szolgáló mérők követése, ezáltal a fogyasztóink optimalizálása, és a fogyasztási szokásaink megismerése, esetlegesen azok optimálisabba való változtatása.

A szoftver alkalmas a lakossági egyszerű villany és gáz mérők értékeinek követésére, melynek segítségével olyan kimutatások diagramok érhetőek el, amelyek alapján következtetéseket tudunk levonni a fűtési, hűtési és egyéb áram és gáz alapú fogyasztóink használatáról esetlegesen azok optimális fogyasztásáról, vagy a szokásainkról. Ezeket figyelembe véve tudunk ezeken változtatni, vagy eszközeinket optimalizálni.

## 2. Fejlesztői dokumentáció

### Deployment

Az alkalmazás használatához, egy böngészőre, egy node.js JavaScript futtató környezetre és egy MySQL szerverre van szükség, lévén a logikát kiszolgáló backend réteget egy node.js web server szolgálja ki, az adatokat pedig egy MySQL adatbázisban tároljuk.

### A fejlesztéshez használt eszközök

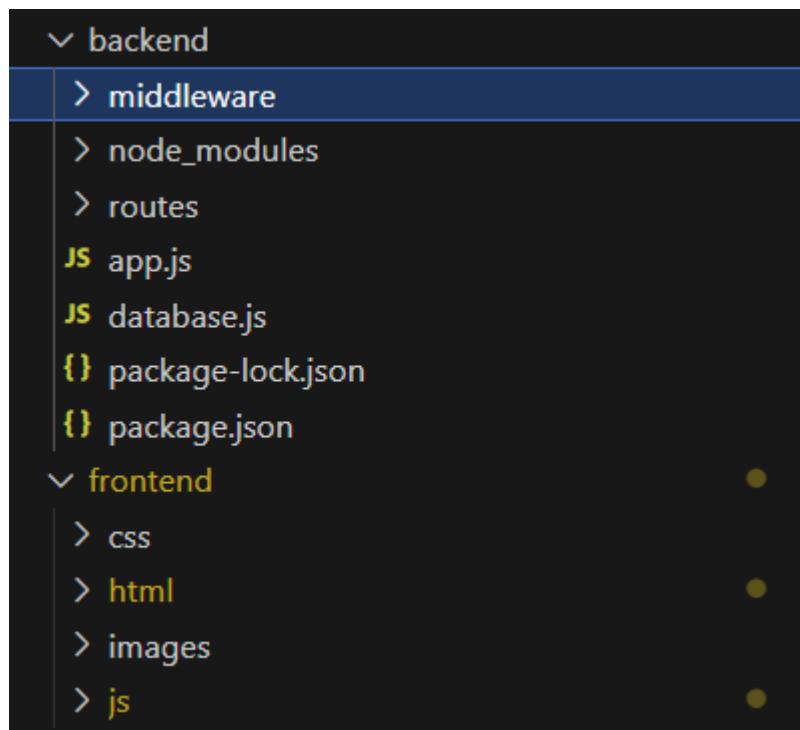
A kód része Visual Studio Code-ban került implementálásra annak is az 1.88.1-es verziójában.

Az adatbázist pedig MySQL Workbench 8 IDE segítségével építettük és programoztuk.

### Az alkalmazás felépítése

#### Könyvtárstruktúra

Az alkalmazás könyvtár struktúrája két fő egységre bomlik, a backend és a frontend könyvtárakra. Ezen belül pedig struktúráltan további alkönyvtárakra bomlik a javascript alkalmazás alapvető felépítési struktúrájának megfelelően.



## Backend

A backend tartalmazza az adatbázis felé a kapcsolatot, az API réteget és a frontend kiszolgálásához szükséges kódokat. Ezek segítségével a backend, adatot tud szolgáltatni a frontendnek a végpont hívásokon keresztül az adatbázisból, valamint adatot tud fogadni és elmenteni azt az adatbázisba.

Esetünkben az adatbázis is tartalmaz némi üzleti logikát, ami próbálja kiküszöbölni azt a tényt, hogy a mérések nem folyamatosak és sokszor több napon keresztül nincs input. Ezért triggerek segítségével, a mérőora állások alapján átlagokat tudunk számolni két rögzítési időpontból.

```

    < backend
      > middleware
      > node_modules
    < routes
      > API
      > bootstrap
      > css
      > html
      > js
    < JS index.js
    < JS app.js
    < JS database.js
    { } package-lock.json
    { } package.json
  
```

backend/routes

Itt találhatóak az alkalmazás útvonalainak kezelésére és irányítására szolgáló fájlok. Ezek a fájlok felelősek az alkalmazás különböző útvonalainak kezeléséért, valamint a klienstől érkező kérések feldolgozásáért és válaszolásáért. Ezek az Express.js keretrendszer segítségével vannak definiálva, minek köszönhetően szervezetté és könnyen kezelhetővé válik az alkalmazás.

### **backend/routes/API**

Itt találhatóak azok a végpontok amelyek az adatbázisból lekérdezik az adatot és szolgáltatják azt a frontend felé.

### **backend/routes/html**

A frontendet html fájait szolgáltatja a böngészőnek.

### **backend/routes/css**

A frontend pageknek szükséges formai és stílus beállításokat tartalmazó fájlokat tartalmazza. Esetünkben a szerver szolgáltatja őket ugyanúgy a routeson keresztül és nem statikusan adjuk meg a frontend oldalon.

### **backend/routes/js**

A frontend logikát JS szolgáltatja. minden frontend page-nek van egy saját fájlja, amit betöltéskor visszaadunk a böngészőnek.

### **backend/app.js**

A webserver maga. Ebben a fájlban lehet beállítani, hogy hol és milyen porton fusson a szerver, és még egyéb middlewareket is meg lehet adni, amelyek az adott alkalmazásnak szükségesek.

```
const express = require("express");
const cookieParser = require("cookie-parser");
const app = express();
const path = require("path");

const backendRoutes = require("./routes/index.js");
const frontendRoutes = path.join(__dirname, "..", "frontend");

const host = "127.0.0.1";
const port = 3000;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use("/frontend", express.static(frontendRoutes));
app.use("/", backendRoutes);

app.listen(port, host, () => {
  console.log(`A szerver itt fut: http://${host}:${port}`);
});
```

### backend/database.js

Az adatbázis kapcsolatot és a hozzáférési paramétereket tartalmazza. Itt építjük fel azt az újrahasznosítható connection objektumot és exportáljuk, amit minden API végpont használni fog, amelyik az adatbázishoz akar kapcsolódni.

```
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "127.0.0.1",
  port: 3306,
  database: "rezsi",
  user: "root",
  password: "",
});

connection.connect(function (err) {
  if (err) {
    console.log("Cannot connect to database!");
    return;
  }
});

module.exports = connection;
```

## Frontend

A frontend könyvtár tartalmazza a kliens oldali scripteket a felületeket felépítő html fájlokat a hozzájuk tartozó css stílus és forma beállítás fájlokkal és a felhasznált képek könyvtárát is a képekkel.

```

    < frontend
      < css
        # electricity.css
        # gas.css
        # home.css
        # login-and-register.css
      < html
        <> electricity.html
        <> gas.html
        <> home.html
        <> login.html
        <> register.html
      > images
    < js
      JS electricity.js
      JS gas.js
      JS home.js
      JS login.js
      JS logout.js
      JS register.js
  
```

### frontend/css

A klienst kiszolgáló típus beállításokat tartalmazza és formai megjelenítési beállításokat.

### frontend/html

A pagek felépítését leíró fájlok, strukturális és tartalmi elemekkel.

## frontend/images

Az alkalmazásban használt háttérképeket tartalmazza.

## frontend/js

Felhasználó és böngésző oldali logikákat és funkciókat tartalmazó fájlokat foglal magában.

### Az alkalmazás backend oldali logikája

Az alkalmazás fejlesztéséhez és futásához szükséges komponensek a **package.json** fájlban találhatóak, ami biztosítja az alkalmazás számára a függőségek beszerzését.

Ez a konfigurációs fájl teszi lehetővé azt is, hogy az alkalmazásról megmondjuk, hogy hogyan futtassuk és mi a fő fájl, amit el kell indítson.

Esetünkben:

```
"main": "app.js",

"dependencies": {
    "bcrypt": "^5.1.1",
    "bootstrap": "^5.3.3",
    "chart.js": "^4.4.2",
    "cookie-parser": "^1.4.6",
    "express": "^4.19.2",
    "moment": "^2.30.1",
    "multer": "^1.4.5-lts.1",
    "mysql2": "^3.9.4",
    "nodemon": "^3.0.3"
},
```

Az alkalmazás függőségei között amit itt megtalálható az egyik legfontosabb az az Express. Az alkalmazás erre az Express keretrendszerre van felépítve. Nagyon könnyen kezelhetővé teszi az útvonalak kezelését, az API hívásokat a kommunikációt, és korábban az app.js ben látott middlewarek sokaságát is kínálja, melyek még kezelhetőbbé teszik a rendszert.

Egy másik hasznos kiegészítő függőség a **path**, melynek segítségével egyszerűbb és átláthatóbb az útvonalak definiálása. Az útvonalak tekintetében az Express keretrendszer Router middleware-ja segítségével ezek az útvonalak és a hozzájuk tartozó műveletek jól csoportosíthatóak, ezzel is további könnyebbéget adva a fejlesztőnek, hogy jobban átlátssa a projektet.

A használata egy definíció és egy felhasználás utasításból áll, melyet az `app.js` fő fájlban tudunk megadni, hogy valóban használni is tudja az alkalmazás ezeket a deklarációkat.

`app.js`

```
const backendRoutes = require("./routes/index.js");
app.use("/", backendRoutes);
```

Az `index.js` fájlban pedig a használt fájlok útvonalai és használatuk van definiálva (pl:`home`):

```
const homeJSRoute = require("./js/home.js");
router.use("/", homeJSRoute);
```

Így az adott js fájlban definiált végpont hívásakor az ott megadott útvonalon lévő fájlt fogja visszaadni a backend.

`home.js`

```
const express = require("express");
const path = require("path");
const router = express.Router();

router.get("/home.js", function (req, res) {
  res.sendFile(
    path.join(__dirname, "...", "...", "...", "frontend", "js",
    "home.js")
  );
});

module.exports = router;
```

Ez a végpont vissza fogja adni a frontendnek a home pagehez tartozó logikát tartalmazó fájlt, amiben azok a hívások szerepelnek, amelyek ténylegesen ki tudják majd szolgálni a felületet adattal, az API végpont hívások használatával.

Ugyan így történik a HTML fájlok útvonal definiálása és azok visszaadása is a frontendnek, hogy be tudjanak tölteni amikor a böngészőben hívják őket.

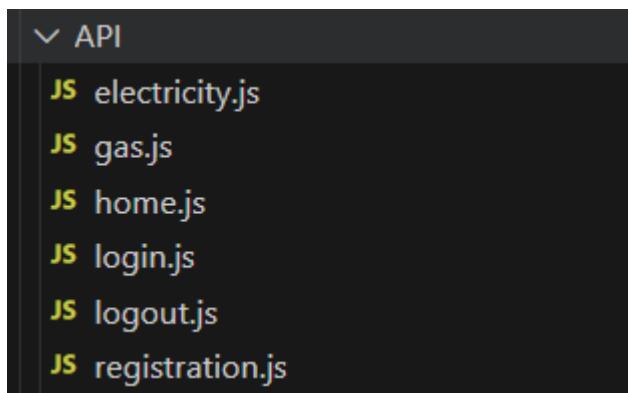
Az alkalmazás első oldala, ami betölt azt a routes modulban az egyszerű / végponthoz rendelt html fájl jelenti. Esetünkben a loginHTML.html oldal van ezen az útvonalon (<http://127.0.0.1:3000>) definiálva. Az alkalmazás betöltésekor így a login oldal fog betölteni.

```
router.get("/", function (req, res) {
  res.sendFile(
    path.join(__dirname, "../", "..", "..", "frontend", "html",
"login.html")
  );
});
```

Hasonlóképpen történik minden router által kezelt fájl kiszolgálása.

Erre alapozva megnézhetjük, hogy milyen fájlok is ezek, hogyan épül fel a logika és mit is csinál a backendünk.

## API végpontok



Az alkalmazás védett oldalakat tartalmaz, regisztrációt és bejelentkezést biztosít a felhasználóknak, hogy hozzáférjenek az alkalmazáshoz biztonságos módon.

Ezért szükség van a login, logout és registration fájlokban lévő végpontokra, hogy ezt a feladatot el tudjuk látni.

A regisztráció során email cím validációt és jelszó hash-t alkalmazunk.

```
router.post("/reg", function (req, res) {
  const { username, email, password } = req.body;
```

```
connection.query(
  "SELECT * FROM users WHERE email = ?",
  [email],
  (err, result) => {
    if (err) {
      return res.json("Hiba a regisztráció során!");
    }

    if (result.length > 0) {
      return res.status(400).json("Az email cím már foglalt");
    }
  }

  bcrypt.hash(password, saltRounds, (err, hash) => {
    if (err) {
      throw err;
    }

    connection.query(
      "INSERT INTO users(userID, email, username, password,
role) VALUES(NULL, ?, ?, ?, 1)",
      [email, username, hash],
      (err, result) => {
        if (err) {
          res.json("Hiba a regisztráció során!");
        }

        res.json("Sikeres regisztráció");
      }
    );
  });
});
```

Login indításkor e-mail cím alapján határozzuk meg a felhasználót és ellenőrizzük a jelszavát, és állítunk be egy cookie-t a válaszhoz. A cookie tartalmaz egy nevet, amelyen

keresztül elérhetőek az adatok, és a felhasználó adatait JSON-ban. A httpOnly:true pedig gondoskodik arról, hogy JS segítségével ne lehessen módosítani, csak HTTP kérésekkel legyen elérhető. A maxAge pedig a cookie élettartamát adja meg.

```
router.post("/login", function (req, res) {
  const { email, password } = req.body;

  connection.query(
    "SELECT * FROM users WHERE email = ?",
    [email],
    (err, result) => {
      if (err) {
        return res.status(500).json("Hiba történt a bejelentkezés során!");
      }

      if (
        result.length === 0 ||
        !bcrypt.compareSync(password, result[0].password)
      ) {
        return res.status(401).json("Hibás jelszó vagy felhasználónév!");
      }
    }
  );

  const user = {
    email: result[0].email,
    username: result[0].username,
    role: result[0].role,
  };

  res.cookie("userData", JSON.stringify(user), {
    httpOnly: true,
    maxAge: 1000 * 60 * 60 * 12,
  });
  return res.json({ success: true, user });
});
```

```
} );
```

A logint követő főoldal a Home oldal, ahol ami az alkalmazás fő oldala. A szerveroldali API végpontok a home oldalon lévő chartokat látják el adattal, ahol az éves fogyasztási adatok jelennek meg.

Ezek a végpontok minden a két típusú mérőre meg vannak írva.

```
router.get("/gasdashboard", async (req, res) => {
  try {
    connection.query(
      "SELECT      EXTRACT(YEAR      FROM      date)      AS      year,
      max(value)-min(value)  AS  value  FROM  gas  GROUP  BY  year",
      (err, result) => {
        if (err) {
          console.error("Error executing query:", err);
          res.status(500).json({ error: "Internal server error" });
        }
        return;
      }

      // Adatok előkészítése a Chart.js chrtthoz
      const gaugeData = {
        labels: result.map((row) => row.year),
        datasets: [
          {
            label: "Gázmérő érték(m3)",
            data: result.map((row) => row.value),
            backgroundColor: "rgba(52, 168, 83, 0.8)",
            borderColor: "rgba(32, 33, 36, 1)",
            borderWidth: 1,
          },
        ],
      };
      // Chart konfiguráció
      const gaugeOptions = {};
    }
  }
});
```

```

        // Visszaküldjük a JSON-t, amely tartalmazza a gauge
adatait és beállításait

        res.json({ gaugeData, gaugeOptions });
    }
);

} catch (err) {
    console.error("Error executing query:", err);
    res.status(500).json({ error: "Internal server error" });
}
);
}
);

```

a **/gasdashboard** végpont lekérdezi a gázmérőra értékét éves bontásban. Ezt az értéket pedig egy szériaként a **guageData**-ban visszaadja a **chartOptionsel** együtt, ahol a chartot érintő beállításokat lehetne már ezen a szinten is eszközölni, ha valami adatbázisból jönne, de ez még jelenleg ebben az esetben üres.

```
res.json({ gaugeData, gaugeOptions });
```

Az árammérőre hasonló a végpont csak a végpont és a lekérdezés más. Ott a **/electricitydashboard** a végpont, és az electricity-ből kérdezünk, le de a tárolási struktúra egyezőség miatt minden egyéb nagyon hasonló.

A Home pagenek még van egy feature funkciója, amit ebben a fájlban lévő végpontok látnak el.

Ez a **/dateCheckGas** és **/dateCheckElectricity** végpontok. A feladatuk, hogy adatot lekérdezve az adatbázisból megállaptsák, hogy külön-külön de mindenkorban az adatok mennyire frissek, és ha 10 napnál régebbiek, akkor egy arra utaló szöveggel térnek vissza a response-ban, hogy felhívják a figyelmet frissebb adat felvételére.

```

router.get("/dateCheckGas", (req, res) => {
    connection.query("SELECT MAX(date) as date FROM gas", (err,
result) => {
    if (err) {
        console.error("Error executing query:", err);
        res.status(500).json({ error: "Internal server error" });
        return;
    }
}
);

```

```

        // Adatelfekvőség figyeléshez számítás: legrégebbi adat
számítás

    const oldestDate = new Date(result[0]["date"]);
    const currentDate = new Date();
    const diffInDays = Math.floor(
        (currentDate - oldestDate) / (1000 * 60 * 60 * 24)
    );

    // Ellenőrizzük, hogy több mint 10 napja van-e az utolsó
dátumnak

    if (diffInDays > 10) {
        // Ha igen, küldjünk egy speciális üzenetet a toaster
számára

        res.status(200).json({
            message: "A legutóbbi gázmérő mérési adat már több mint 10
napos!",
        });
    } else {
        // res.json({ message: 'A gázmérő mérési adatok 10 napon
belüliek!' });
    }
}
);
}
);

```

A következő két végpontot tartalmazó API könyvtár fájl ugyanazt a feladatot látja el így csak a gázmérő esetén tárgyaljuk.

A két fájl a `gas.js` és az `electricity.js` fájlok.

Itt a “Gázóra állások” és “Villanyóra állások” oldalakat kiszolgáló végpontok találhatóak. A funkcionálitás, amit kiszolgálnak azok a gáz és az árammérők aktuális állásainak rögzítési lehetőségei, azok chartokban való megtekintése, valamint táblázatos formában való vizualizáció és adatmanipuláció.

Ehhez a

**/gas, /gasdata, /gasdata/:id, /gasdatachart** főbb végpontokat implementáltuk.

A `/gas` végpont az adatrögzítést oldja meg, ahol több az adatok helyességét biztosító ellenőrzést is elvégzünk a beküldött adaton.

```
router.post("/gas", function (req, res) {
```

```

// Ellenőrizzük, hogy a mDate és mValue nem üres string
const mDate = req.body.mDate.trim();
const mValue = req.body.mValue.trim();

if (!mDate || !mValue) {
    return res.status(400).json({ error: "Hiányzó dátum vagy érték" });
}

// Ellenőrizzük, hogy a mDate érvényes dátum-e
const dateRegex = /^(\d{4})-(\d{2})-(\d{2})$/;
if (!dateRegex.test(mDate)) {
    return res.status(400).json({ error: "Érvénytelen dátum formátum" });
}

// Ellenőrizzük, hogy a mValue egy szám
const value = parseFloat(mValue);
if (isNaN(value)) {
    return res.status(400).json({ error: "Az érték nem szám" });
}

// Ellenőrizzd, hogy a szám nem negatív
if (value < 0) {
    return res.status(400).json({ error: "Az érték nem lehet negatív" });
}

connection.query(
    "INSERT INTO rezsi.gas (id, date, value) VALUES (null, ?, ?)",
    [mDate, mValue],
    (err, result) => {
        if (err) {
            res.json(err);
        }
        res.json(result);
    }
)

```

```
) ;  
} ) ;
```

A **/gasdata** végpontok a rögzített adatokat, vagy csak egy szűrésnek megfelelő adatokat adják vissza. Utóbbi csak a dátumban és az értékben való keresést támogatja, a többi attribútum érték nem kereshető, csak megjeleníthető.

```
// Rögzített adatok lekérdezése  
router.get("/gasdata", function (req, res) {  
    connection.query(  
        "SELECT id, date, value, daily_average,intervall_length FROM  
gas ORDER BY date DESC",  
        (err, result) => {  
            if (err) {  
                return res.json(err);  
            }  
            res.json(result);  
        }  
    );  
});  
  
// Rögzített adatok keresése  
router.post("/gasdata", function (req, res) {  
    const searching = req.body.searching;  
    connection.query(  
        'SELECT id, date, value FROM gas WHERE date LIKE CONCAT("%",  
?, "%") OR value LIKE CONCAT("%", ?, "%") ',  
        [searching, searching],  
        (err, result) => {  
            if (err) {  
                res.json(err);  
            }  
            res.json(result);  
        }  
    );  
});
```

A **/gasdata/:id** végpontok az updateit és törlést támogatják.

```

// Rekord törlése
router.delete("/gasdata/:id", function (req, res) {
  const deleteID = parseInt(req.params.id);
  connection.query(
    "DELETE FROM gas WHERE id = ?",
    [deleteID],
    (err, result) => {
      if (err) {
        res.status(500).json(err);
      }
      res.status(200).json(result);
    }
  );
} );

// Rekord lekérdezése
router.get("/gasdata/:id", function (req, res) {
  const gasId = parseInt(req.params.id);
  connection.query(
    "SELECT date, value FROM gas WHERE id = ?",
    [gasId],
    (err, result) => {
      if (err) {
        res.status(500).json(err);
      }
      res.status(200).json(result);
    }
  );
} );

// Rekord módosítása
router.put("/gasdata/:id", function (req, res) {
  const gasId = parseInt(req.params.id);
  const { newDate, newValue } = req.body;
  connection.query(
    "UPDATE gas SET date = COALESCE(?, date), value = COALESCE(?, value) WHERE id = ?",
    [newDate, newValue, gasId]
  );
}
);

```

```
[newDate, newValue, gasId],
  (err, result) => {
    if (err) {
      res.status(500).json(err);
    }
    res.status(200).json(result);
  }
);
} );
} );
```

A **/gasdatachart** végpont, pedig a gázóra átlagértékek megjelenítését biztosító alap adatokért felel.

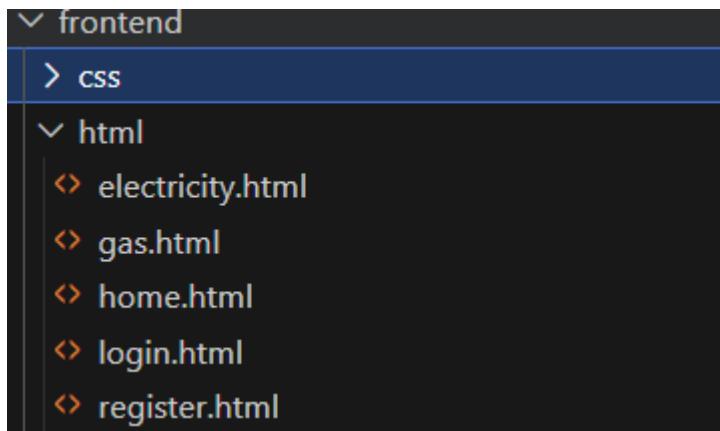
```
router.get("/gasdatachart", function (req, res) {
  connection.query("SELECT date, daily_average FROM gas", (err,
result) => {
  if (err) {
    return res.json(err);
  }
  res.json(result);
}) ;
} );
```

### Az alkalmazás frontend oldali logikája

A frontend a html fájlokból az öket formázó és stilizáló css fájlokból, a felhasznált képekből és a frontend oldali JS fájlokból azaz logikákból épül fel.

A főbb könyvtárak a html és JS ahol alapvetően felépül a frontend.

## frontend/html



A `register.html` fájlban definiáljuk a regisztrációs oldalt, amivel az alkalmazás landing page a login felület ad egy átnavigálási lehetőséget erre az oldalra, ahol be tudjuk regisztrálni a felhasználót.

```
<button type="button" class="btn" onclick="register()">>
```

A regisztrációs oldalon felhasználót, e-mail címet és jelszót bekérő komponensek kerültek fel és maga egy regisztrációt végző gomb, ami a hozzá tartozó JS fájlban lévő submit eseményt kezeli le.

```
<div class="input-container">
    <i class="fa fa-user icon"></i>
    <input
        class="input-field"
        type="text"
        placeholder="Felhasználónév"
        name="usrnm"
    />
</div>
<div class="input-container">
    <i class="fa fa-envelope icon"></i>
    <input
        class="input-field"
        type="email"
        placeholder="E-mail"
        name="email"
    />
</div>
<div class="input-container">
```

```
<i class="fa fa-key icon"></i>
<input
    class="input-field"
    type="password"
    placeholder="Jelszó"
    name="psw"
/>
</div>
```

```
<button type="submit" class="btn">Regisztráció</button>
```

A login.html hasonló elemeket tartalmaz a felhasználónév kivételével. Itt e-mail és jelszó komponensek kerültek felhasználásra. Mivel ez a kezdőképernyő így itt van a regisztrációra navigálás lehetősége, amit fentebb említettünk, valamint a bejelentkezést végző gomb, amit a hozzáartozó login.js fájlban lévő submit eseményben kezelünk le.

```
<button type="submit" class="btn">Bejelentkezés</button>
```

A további fájlok mint home.html, gas.html és electricity.html hordoznak egy közös kód részletet, mégpedig a navigációs bar-t tartalmazó header, és a footert. Ezeken az oldalakon a navigáció és a footer ahol a szerzők vannak feltüntetve egységes.

```
<header
    class="navbar navbar-expand bg-light navbar-light sticky-top shadow-sm">
    <div class="container-fluid">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link active" href="/home">
                    <i class="fa-solid fa-home"></i>
                    <span>Kezdőlap</span>
                </a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/gas">
                    <i class="fa-solid fa-mask-ventilator"></i>
                    <span>Gázóra állások</span>
                </a>
            </li>
        </ul>
    </div>
</header>
```

```

<li class="nav-item">
    <a class="nav-link" href="/electricity"
        ><i class="fa-solid fa-plug"></i>
        <span>Villanyóra állások</span></a>
    >
</li>
</ul>

        <button id="logout" onclick="logout()" class="button float-end">
            <i class="fa-solid fa-right-from-bracket"></i>
        </button>
    </div>
</header>

```

```

<footer class="container-fluid p-1 text-center sticky-bottom">
    Készítette: Balogh Krisztián, Nagy István, Szilágyi Attila
</footer>

```

A home.html az az oldal, ahova a sikeres bejelentkezést követően megérkezünk. Itt a közös részeken kívül kettő oszlop diagrammal találkozhatunk és egy toaster üzenettel. A chartok mérő típusonként mutatják éves bontásban a felhasznált mennyiséget, toaster pedig egy figyelmeztetést jelenít meg, amennyiben valamilyen mérési adat már 10 napnál régebbi, jelezve a felhasználónak, hogy a statisztikák pontossága miatt jó lenne adatot bevinni.

### Chartok beágyazása

```

<div class="container-fluid overflow-y-auto">
    <div id="canvasContainer">
        <canvas
            id="gasGauge"
            style="width: 100%; height: 45%; min-height: 400px"
        ></canvas>
        <canvas
            id="electricityGauge"
            style="width: 100%; height: 45%; min-height: 400px"
        ></canvas>
    </div>

```

```
</div>
```

## Toaster beágyazása

```
<script
src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
```

A gas.html és electricity.html oldalak itt is majdnem megegyező felépítésűek, lévén adatbázis és backen logika szerint is azok. Ezért csak a gas.html tartalmán mutatjuk be a frontend oldali html részben.

A gas.html ad lehetőséget a gázmérő óra állás rögzítésre beviteli mezők segítségével, amit egy formba ágyaztunk:

```
<form id="add-gas">
    <div class="mb-3 mt-3">
        <label class="form-label">Dátum:</label>
        <input
            type="date"
            class="form-control"
            id="mDate"
            placeholder="Dátum"
        />
    </div>
    <div class="mb-3">
        <label class="form-label">Mérő állása:</label>
        <input
            type="number"
            class="form-control"
            id="mValue"
            placeholder="Gázóra állás értéke"
        />
    </div>
    <button type="submit" class="button">Rögzít</button>
</form>
```

Itt a date mezőben automatikusan meghatározásra kerül az aktuális dátum. Erről a home.js fájlban gondoskodik egy logika, amit később bemutatunk.

Itt is található egy chart és egy táblázat: <div id="canvasContainer">

```
<canvas
    id="gasChart"
    style="width: 100%; height: 100%; min-height: 400px"
></canvas>

</div>

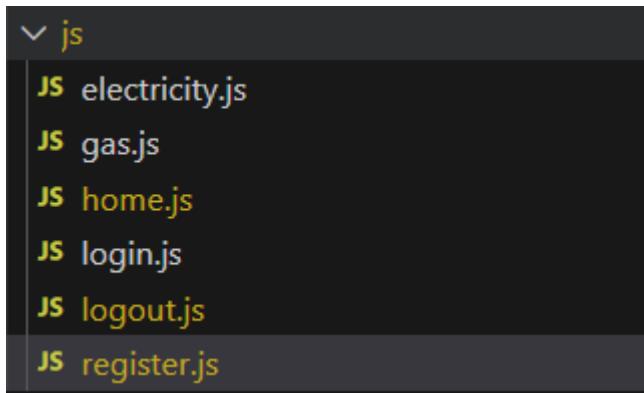
<form    class="d-flex    mt-3    mb-3"    role="search"
id="searchingForm">
    <input
        class="form-control"
        type="search"
        placeholder="Keresés"
        aria-label="Search"
        name="searching"
        id="searching"
    />
    <button class="btn" type="submit">
        <i class="fa-solid fa-magnifying-glass"></i>
    </button>
</form>

<div id="gasTable"></div>
</div>
```

ahol a chart évenként bontva naponként megjelenítve éven belül az aznapra eső fogyasztást. A táblázat pedig listázza a bevitt értékeket, lehetőséget biztosít keresésre és adatmódosításra, törlésre.

#### frontend/js

A JavaScript frontend logikai fájlok.



A `register.js` fájlban a `registerForm onsubmit` eseményében kiolvassuk a komponensek értékeit és azt a `/reg` POST végpont hívással átadjuk a backendnek. A beolvasáskor ellenőrizzük, hogy nem e üres valamelyik mező.

```
document.getElementById("registerForm").onsubmit = function(event) {
    event.preventDefault();

    const username = event.target.elements.usrnme.value;
    const email = event.target.elements.email.value;
    const password = event.target.elements.psw.value;

    if (!username && !email && !password) {
        alert("Minden mezőt ki kell tölteni!");
        return;
    }
    reg(username, email, password);
};

async function reg(username, email, password) {
    const res = await fetch("/reg", {
        method: "POST",
        headers: {
            "content-type": "application/json",
        },
        body: JSON.stringify({ username, email, password }),
    });

    const data = await res.json();
}
```

```
if (res) {  
    alert(JSON.stringify(data));  
}  
else {  
    alert(JSON.stringify(data));  
}  
}  
}
```

A `login.js` fájlban, ami a bejelentkezést támogatja, beolvassuk a komponensek értékeit (e-mail cím és jelszó) azt és egy POST `/login` végpont hívással átadjuk a backendnek. A `response`-t megvizsgáljuk, és eldöntjük, hogy jogosult-e a belépésre vagy sem, és amennyiben igen, akkor átnavigáljuk a `home` oldalra.

```
document.getElementById("loginForm").onsubmit = async function(event) {
    event.preventDefault();

    const email = event.target.elements.email.value;
    const password = event.target.elements.psw.value;

    const res = await fetch("/login", {
        method: "POST",
        headers: {
            "content-type": "application/json; charset=utf-8",
        },
        body: JSON.stringify({
            email: email,
            password: password,
        }),
    });
    const data = await res.json();

    if (data.success) {
        if (data.user.role === 1) {
            window.location.href = "/home";
        }
    } else {
        alert(JSON.stringify(data));
    }
}
```

```
}
```

```
};
```

A logout.js fájlból, a **/logout** végpont hívással jelezük, hogy a felhasználó kilép és visszานavigáljuk a login oldalra, de előtte a hívásnak köszönhetően töröljük a cookie-ját.

```
async function logout() {
    //console.log('lefutott');

    const res = await fetch("/logout", {
        method: "POST",
        headers: {
            "content-type": "application/json; charset=utf-8",
        },
    });

    const data = await res.json();
    if (data.success) {
        window.location.href = "/";
    } else {
        console.log(`Hiba a kijelentkezéskor: ${data.message}`);
    }
}
```

A home.js a home.html oldalnál említett chartokat és toasteres figyelmeztető funkciókat támogatja.

A gáz fogyasztási és áram fogyasztási chartok megegyezők így az egyiket mutatjuk csak be.

```
// Kezdőképernyő gázmérő érték éves bontásban
document.addEventListener("DOMContentLoaded", function () {
    fetch("/gasdashboard")
        .then((response) => response.json())
        .then((data) => {
            // Gauge adatai és beállításai
            const gaugeData = data.gaugeData;
            const gaugeOptions = data.gaugeOptions;
```

```
// A canvas elem kiválasztása
const canvas = document.getElementById("gasGauge");

// Gauge inicializálása
const ctx = canvas.getContext("2d");
new Chart(ctx, {
    type: "bar",
    data: gaugeData,
    options: {
        legend: {
            display: true,
            labels: {
                fontSize: 18,
            },
        },
        title: {
            display: true,
            text: "Gáz fogyasztás éves bontásban",
            fontSize: 20,
        },
        scales: {
            xAxes: [
                {
                    ticks: {
                        fontSize: 16,
                    },
                },
            ],
            yAxes: [
                {
                    ticks: {
                        fontSize: 16,
                    },
                },
            ],
        },
        responsive: false,
```

```

        } ,
    } ) ;
} )
    .catch((error) => console.error("Error fetching data:",
error));
} );
}

```

A **/gasdashboard** végpont hívással visszaadja a backend az éves adatokat a fogyasztásra nézve, és a chartnak a beállításait is, amennyiben már backend oldalon is vannak ilyen beállítások. Ezt követően pedig bizonyos opciós chart tulajdonságok mellett beállítjuk a chartot.

A toasteres figyelmeztetés is megegyező a két mérőtípus esetén ezért abból is csak a gázmérőn keresztül mutatjuk be a logikát.

```

// Ellenőrizzük mennyire friss az adat
document.addEventListener("DOMContentLoaded", function () {
    $.get("/dateCheckGas", function (response) {
        if (response.message) {
            Swal.fire({
                position: "top-end",
                icon: "warning",
                showConfirmButton: false,
                timer: 2500,
                text: response.message,
            });
        } else {
            console.log("Error reading date from database.");
        }
    }).fail(function (err) {
        console.error("Error calling API:", err);
    });
});
}

```

A **/dataCheckGas** végpont hívással a backend visszaad egy üzenetet. Ezt az üzenetet jeleníti meg a toaster “warning” módban, 2.5 másodpercig.

Az **/dateCheckElectricity** végpont hívást követően ez a megjelenítés 3 másodperccel késleltetve van, hogy egymást követően tudjuk megjeleníteni azt és ne írja egyik felül a másikat.

```
Swal.fire({
    position: "top-end",
    icon: "warning",
    showConfirmButton: false,
    timer: 2500,
    text: response.message,
}) ;
}, 3000);
```

A `gas.js` és `electricity.js` szintén megegyező csak a mérőkben és az ōket leíró változó és végpont név hívásokban térnek el, így ebben az esetben is csak a gázmérő esetén mutatjuk be a logikát.

A `gas.js` fő feladata a `gas.html`-ben szereplő adatbevitel valamint a `chart` és táblázat kiszolgálása.

Az adatok betöltéséért és minden friss megjelenítéséért két metódus felel a `fetchGasData()` és a `fetchChartData()`. ezeket egyszerre hívjuk a cél érdekében már betöltéskor:

```
document.addEventListener("DOMContentLoaded", function () {
    fetchGasData();
    fetchChartData();
});
```

Az adatbevitel a **/gas** végpont POST hívásával történik az `onsubmit` eseményre. Itt a dátum és érték mezők értékeit küldjük a backendnek.

```
// Dátum és érték kiolvasás és küldés a backendnek
document.getElementById("add-gas").onsubmit = async function(event) {
    event.preventDefault();
    const mDate = event.target.elements.mDate.value;
    const mValue = event.target.elements.mValue.value;

    const res = await fetch("/gas", {
        method: "POST",
        headers: {
```

```

    "content-type": "application/json",
  },
  body: JSON.stringify({ mDate, mValue }),
);
if (res.ok) {
  event.target.elements.mValue.value = null;
  fetchGasData(); // Táblázat frissítések
  fetchChartData(); // Chart frissítése
}
};

```

A dátum esetén az automatikus aktuális kitöltés is betöltéskor megy végbe itt:

```

// Aktuális dátum előtöltés
document.addEventListener("DOMContentLoaded", function () {
  const currentDate = new Date().toISOString().slice(0, 10);
  document.getElementById("mDate").value = currentDate;
});

```

A sikeres végpont hívást követően, pedig újratöljük az adatokat, hogy a frissen bevitt adat is szerepeljen a charton és a táblázatba.

Az adatokat a táblázatban két funkció gomb segítségével módosíthatjuk meg. Az egyik egy dialógust hoz fel és előtölти az adatot, hogy azt szerkesztve visszamentsük az adatbázisba. A másik gombbal pedig törölhetjük az adatot.

A Táblázat felépítése:

```

function displayGasData(data) {
  const gasTableDiv = document.getElementById("gasTable");
  const table = document.createElement("table");

  // Fej sorok
  table.className = "table table-bordered table-striped";
  const headerRow = table.insertRow();
  headerRow.className = "table-dark";
  const dateHeader = headerRow.insertCell();
  dateHeader.textContent = "Dátum";
  const dailyHeader = headerRow.insertCell();

```

```

dailyHeader.textContent = "Érték";
const dailyAverageHeader = headerRow.insertCell();
dailyAverageHeader.textContent = "Napi átlag";
const periodHeader = headerRow.insertCell();
periodHeader.textContent = "Eltelet napok";
const buttonHeader = headerRow.insertCell();
buttonHeader.textContent = "Műveletek";
buttonHeader.className = "col-sm-1";

// Adatsorok
data.forEach((entry) => {
    const row = table.insertRow();
    const dateCell = row.insertCell();
    dateCell.textContent = entry.date.slice(0, 10);
    const dailyCell = row.insertCell();
    dailyCell.textContent = entry.value;
    const dailyAverageCell = row.insertCell();
    dailyAverageCell.textContent = entry.daily_average;
    const periodCell = row.insertCell();
    periodCell.textContent = entry.intervall_length;
    const buttonCell = row.insertCell();

    // Funkciógombok
    buttonCell.innerHTML = `<div>
        <button class="btn btn-outline-secondary"
type="button" onclick="deleteGasData(${entry.id})"><i
class="fa-solid fa-trash fa-xs"></i></button>
        <button class="btn btn-outline-secondary"
type="button" onclick="editGasData(${entry.id})"><i
class="fa-solid fa-edit fa-xs"></i></button>
    </div>`;
});

gasTableDiv.innerHTML = table.outerHTML;
}

```

és feltöltése adatokkal:

```
async function fetchGasData() {
  const res = await fetch("/gasdata");
  const gasData = await res.json();

  displayGasData(gasData);
}
```

ami a **/gasdata** végpont hívással megy vége. Az itt kapott response pedig tartalmazza a dátum, érték, napi átlag és eltelt napok értékeit.

A táblázatba pedig minden sorhoz egy innerHTML beágyazás segítségével fűzzük hozzá a funkcióból.

A szerkesztés gomb eseménye egy felugró ablak, ahol feltöltjük az adott sor értékeivel a mezőket

```
async function editGasData(idGas) {
  const res = await fetch(`/gasdata/${idGas}`);
  const data = await res.json();
  console.log("Módosítandó adatok: ", data);

  const editGasForm = document.getElementById("editGas");

  const modal = new bootstrap.Modal(editGasForm);

  editGasForm.setAttribute("data-id", idGas);

  const editDate = data[0].date.slice(0, 10);
  document.getElementById("editDate").value = editDate;
  document.getElementById("editValue").value = data[0].value;

  modal.show();
}
```

és visszamentjük azt a **/gasdata/\${id}** PUT típusú hívással.

```
async function updateGasData() {
  const editForm = document.getElementById("editGas");
  const id = editForm.getAttribute("data-id");
  const newDate = document.getElementById("editDate").value;
  console.log("Új dátum: ", newDate);
```

```

const newValue = document.getElementById("editValue").value;
console.log("Új mérőállás: ", newValue);

const res = await fetch(`/gasdata/${id}`, {
  method: "PUT",
  headers: {
    "content-type": "application/json",
  },
  body: JSON.stringify({ newDate, newValue }),
}) ;

if (res.ok) {
  alert("Sikeres módosítás!");
  fetchGasData();
  fetchChartData();
} else {
  console.log(res);
  alert("Hiba a módosítás során!");
}
}

```

A törlés gomb eseményre pedig a `/gasdata/${id}` rekord DELETE method hívása fogja kitörölni a rekordot Ezt követően itt is frissítjük az adatokat, így a chartról és táblázatból is el fog tűnni az adott rekord.

```

// Adatrekord törlése
async function deleteGasData(idGas) {
  const confirmed = confirm("Biztosan szeretné törölni a rekordot?");

  if (!confirmed) {
    return;
  }

  const res = await fetch(`/gasdata/${idGas}`, {
    method: "DELETE",
  });
}

```

```
//const gasData = await res.json();

if (res.ok) {
    fetchGasData();
    fetchChartData();
}
}
```

A `gas.js` legnagyobb metódusa a chart kiszolgálása. Az itt adódó extra feladatok abból fakadnak, hogy az adatok bevitelre nem meghatározott ciklikusságú valamint a `chart.js` komponensek rendezése nem terjed ki minden esetre.

A ciklikusságot tekintve ugyebár, ha minden nap berögzítésre kerülne egy és csakis egy adat akkor nagyon egyszerű dolgunk lenne, de mivel két bevitel között beláthatatlan idő telik el így gondoskodni kell a szakadási pontokról, hogy a vizualizáció egy trendet tudjon mutatni és ne csak adatpontokat.

Ezért a hiányzó napokat meg kell határozni.

Előbb felolvassuk a meglévő tárolt adatokat, amit a `/gasdatachart` végpont ad nekünk vissza.

```
async function fetchChartData() {
    const res = await fetch("/gasdatachart");
    const gasSeries = await res.json();
```

ezeket előbb rendezzük:

```
// Rendezzük az adatokat időrendi sorrendbe
gasSeries.sort((a, b) => new Date(a.date) - new Date(b.date));
```

majd kigyűjtjük a char számára szükséges formában év, hónap nap szerint:

```
for (let gasItem of gasSeries) {
    const date = new Date(gasItem.date);
    const year = date.getFullYear();
    const month = date.getMonth() + 1; // Hónapok 0-tól
számolódnak, ezért adunk hozzá 1-et
    const day = date.getDate();
    const key = `${month.toString().padStart(2, "0")}-${day
        .toString()
        .padStart(2, "0")}`; // Hónap-nap, kiegészítve 0-val a
rendezés miatt
```

```
// Hiányzó adatok kiegészítése a rendezés és szakadási pontok
miatt
if (!gasData[key]) {
    gasData[key] = {};
}

gasData[key][year] = gasItem.daily_average;
}
```

és kiegészítjük azt, undefined értékekkel a key-ekhez, hogy minden hónap minden napjának legyen meg a helye, hogy később adatokat tudunk beszúrni azokhoz a key-ekhez is. Ezt azért csináljunk mert a chart napokban vizualizál.

Ezért le kell generáljuk az év összes napját:

```
// Az év összes napjának generálása
const startDate = new Date(gasSeries[0].date);
const endDate = new Date(gasSeries[gasSeries.length - 1].date);
const allDates = generateAllDates(startDate, endDate);

// Az év összes dátumának generálása
function generateAllDates(startDate, endDate) {
    const dates = [];
    let currentDate = new Date(startDate);
    while (currentDate <= endDate) {
        dates.push(new Date(currentDate));
        currentDate.setDate(currentDate.getDate() + 1);
    }
    return dates;
}
```

Elkezdjük összeállítani az adatokat a chartnak megfelelően:

```
// Adatok összeállítása a hiányzó napokra
for (const date of allDates) {
    const year = date.getFullYear();
    const month = date.getMonth() + 1;
    const day = date.getDate();
    const key = `${month.toString().padStart(2, "0")}-${day
        .toString()}
```

```

    .padStart(2, "0") }`;

    if (!gasData[key]) {
        gasData[key] = {};
    }

    // Ha az adott dátumhoz nincs adat, akkor nullát állítunk be
    if (!gasData[key][year]) {
        gasData[key][year] = null;
    }
}

const gasLabels = Object.keys(gasData).sort(); // Rendezzük a
hónap-nap párokat
const years = Array.from(
    new Set(gasSeries.map((item) => new
Date(item.date).getFullYear())))
    .sort();

```

Mivel a chart az x tengelyen rosszul rendez növekvő sorban, ha van 1 és a 10 elem is, mivel 1 után a 10 jön, mert a 10 is 1-essel kezdődik, így még az egy számjegyű elemeket ki kell egészíteni egy 0-val.

Ezt követően, egy új tömbben, ahol már a fogyasztási értékeket tároljuk, az adott év hónap nap párhoz.

Végig iterálunk a gasLabel tömbön, amiben rendezett hónap-nap párok vannak, és ha van értékünk hozzá akkor azzal töltjük fel különben nullával, így mivel minden napra lesz egy adatpontunk még ha nulla is az értéke, de chart ki tudja majd rajzolni és kitölteni a két adattal rendelkező pont között egy vélt görbével egy trendet mutatva.

```

// Adatsorok létrehozása minden évhez
for (const year of years) {
    const gasValues = [];

    for (const gasLabel of gasLabels) {
        // Ellenőrizzük, hogy az adott évhez tartozó hónap-nap
párhoz van-e érték
        if (gasData[gasLabel][year] !== undefined) {

```

```

        gasValues.push(gasData[gasLabel][year]);
    } else {
        gasValues.push(null); // Ha nincs érték, nullát adunk
hozzá
    }
}

```

Időközben meghatározzuk melyik az utolsó széria, mert csak azt akarjuk a charton majd megjeleníteni a többi hidden lesz. Ezt a chartnak egy beállítása tudja majd kezelni.

```

const isLastDataset = year === years[years.length - 1]; // Az
utolsó adatsor ellenőrzése (Ez azért kell, hogy a többi szériát
inaktivá tegyük automatikusan)

```

Összeállítjuk a chart számára a forrást:

```

datasets.push({
    label: year,
    borderColor: barColors.pop(),
    fill: false,
    data: gasValues,
    hidden: !isLastDataset,
});

```

és felépítjük a chartot:

```

// A chart feléptése és adatbetöltés
new Chart("gasChart", {
    type: "line",
    data: {
        labels: gasLabels,
        datasets: datasets,
    },
    options: {
        legend: { display: true },
        title: {
            display: true,
            text: "Gáz fogyasztás",
            fontSize: 18,
        }
    }
})

```

```

        } ,
        scales: {
            xAxes: [
                {
                    ticks: {
                        fontSize: 14,
                    },
                },
            ],
            yAxes: [
                {
                    ticks: {
                        fontSize: 14,
                    },
                },
            ],
        },
        spanGaps: true,
    },
} );
}
}

```

Itt pár főbb beállítás a chart típusa: Line, a forrás és labelek átadása:

```

data: {
    labels: gasLabels,
    datasets: datasets,
},

```

Végezetül a nem megjelenített szériák miatt a chart legend eseményeket is kell kezelni, hogy a többi év adatait is meg tudjuk jeleníteni egyszerre, így összehasonlításokat tudunk végezni:

```

// Legendában lévő címkékre kattintás eseménykezelője
document.getElementById("gasChart").onclick = function (evt) {
    const activePoints = myChart.getElementsAtEventForMode(
        evt,
        "point",
        myChart.options
    )
}

```

```
) ;  
if (activePoints.length > 0) {  
    const datasetIndex = activePoints[0].datasetIndex;  
    const dataset = myChart.data.datasets[datasetIndex];  
    dataset.hidden = !dataset.hidden; // Adatsor elrejtése vagy  
    megjelenítése  
    myChart.update(); // Chart frissítése  
}  
};
```

## Adatbázis modell

Az adatbázis egyszerű felépítésű. A bejelentkezés, védett lapok és regisztráció támogatása végett létrehoztuk a login és users táblát, amiben az ehhez szükséges adatokat tároljuk. Valamint a mérőrák adatainak a tárolására a gas és electricity táblákat.

A gas és electricity táblák megegyezők, mert ugyan azokat az adatokat tároljuk róluk: dátum és érték. Ezek a táblák viszont kiegészülnek két fontos számolt értékkel, amit triggerek biztosítanak számunkra.

**Daily\_average** - átlagérték, amit abból számolunk ki, hogy a rögzített érték és annak dátuma, valamint az azt megelőző dátumú érték között mennyi a különbség. Az értékkülönbséget osztva a napok számával megkapjuk a vélt átlagot. Lévén ez egy szakadási pont, ha nincs minden napra adat így ezt mi próbáljuk biztosítani szoftveresen.

**Intervall\_length** - egyszerűen csak az eltelt napok száma, hogy nagyjából azért következtetni tudjunk az átlag pontosságára, hiszen, ha az eltelt napok száma túl magas, akkor nagyobb valószínűséggel pontatlan az adat, ha pl egy gázfogyasztást nézünk.

```

CREATE DEFINER='root'@'localhost` TRIGGER `gas_BEFORE_INSERT` BEFORE INSERT
ON `gas` FOR EACH ROW BEGIN
    DECLARE prev_value INT;
    DECLARE prev_date DATE;

    SELECT value, date INTO prev_value, prev_date
    FROM gas
    WHERE date < NEW.date
    ORDER BY date DESC
    LIMIT 1;

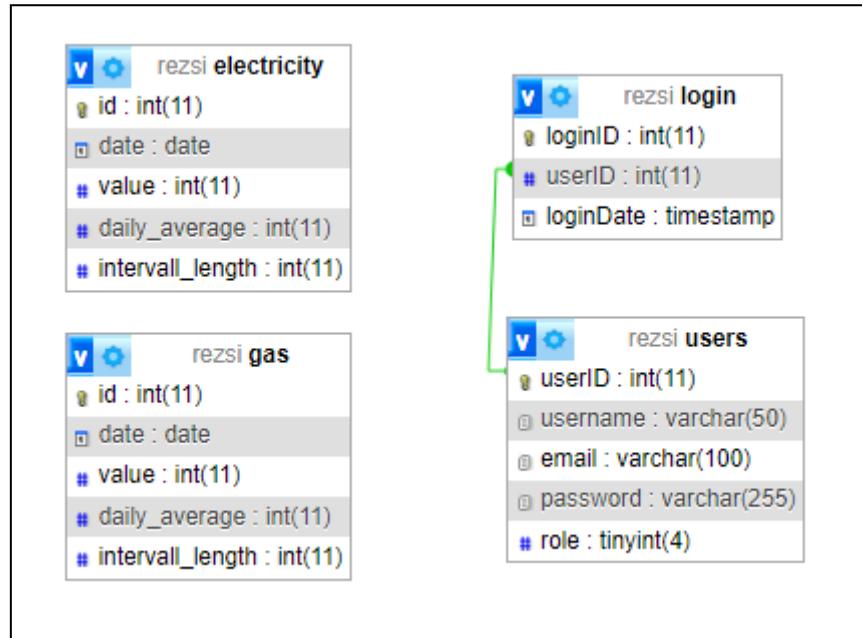
    IF prev_value IS NOT NULL THEN
        SET NEW.daily_average = (NEW.value - prev_value) / DATEDIFF(NEW.date,
prev_date);
    END IF;
END

CREATE DEFINER='root'@'localhost` TRIGGER `gas_BEFORE_INSERT_1` BEFORE INSERT
ON `gas` FOR EACH ROW BEGIN
    DECLARE prev_date DATE;

    -- Keresd meg az utolsó beszúrt rekord dátumát
    SELECT MAX(date) INTO prev_date FROM gas;

```

```
-- Számold ki az intervallum hosszát napokban, ha van előző rekord  
IF prev_date IS NOT NULL THEN  
    SET NEW.intervall_length = DATEDIFF(NEW.date, prev_date);  
END IF;  
END
```



### 3. Felhasználói dokumentáció

#### Alkalmazás célja

Az általunk fejlesztett Rezsi App egy webes alkalmazás, mely böngészőben futtatva alkalmas villany- és gázóra állások feltöltésére, módosítására, törlésére. A bevitt adatokból éves fogyasztási diagramokat képzünk. Az alkalmazás fontos funkciója a chartok, melyeken az éves fogyasztási görbék tetszőlegesen kapcsolhatók és egymásra illeszthetőek, ezzel segítve a felhasználókat az energiafogyasztásuk optimalizálásában.

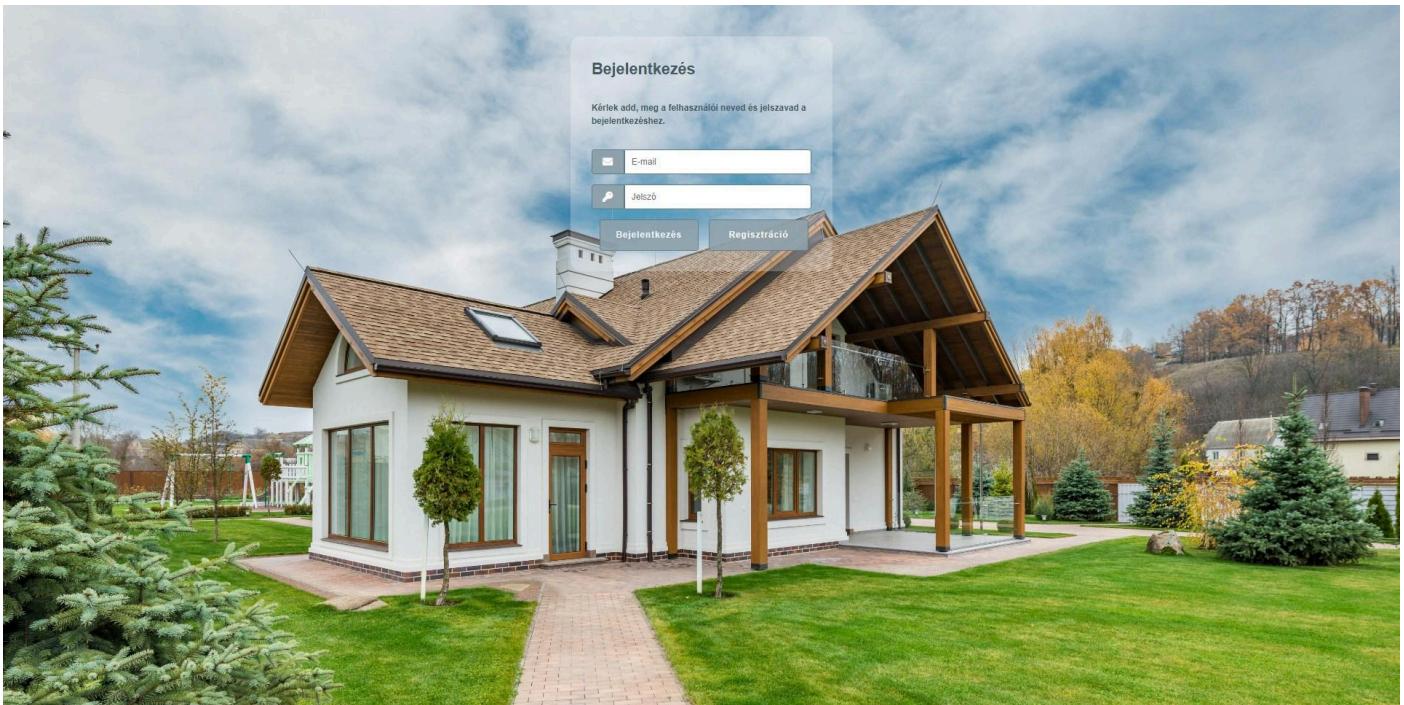
A teljes weboldal reszponzív, támogatva a telefonos felületről történő óraállások gyors bevitelét.

Az oldal használata regisztrációhoz kötött, anélkül csak a bejelentkezés és a regisztráció érhető el.

Bejelentkezést követően az alábbi oldalakon böngészhetünk:

- Kezdőlap
- Gázóra állások
- Villanyóra állások

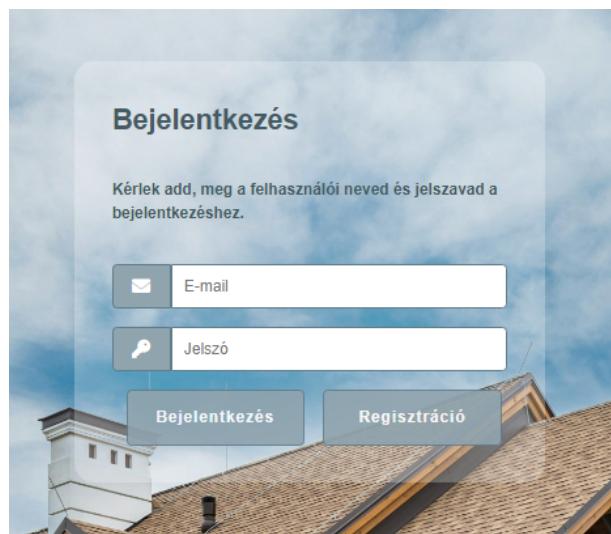
## Rezsi App nyitóoldal



### Bejelentkezés

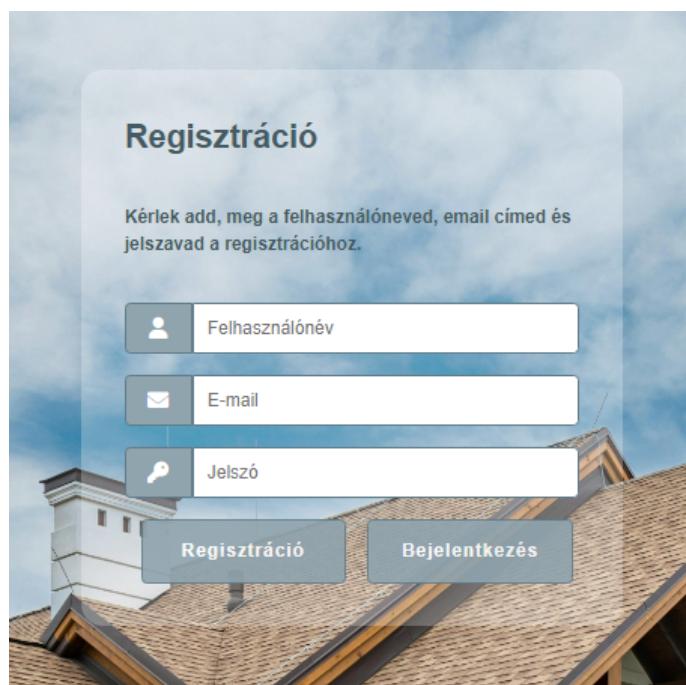
A felhasználók azonosítása e-mail cím és jelszó párossal történik. Bejelentkezés gombra kattintva a Kezdőoldalra jutunk, ezzel egyidőben egy cookie kerül tárolásra, ennek célja a védett oldalak (Kezdőlap, Gáz-, Villanyóra állások) csakis bejelentkezést követően legyenek elérhetőek. A cookie lejáratú idővel rendelkezik, ha lejárt a felhasználónak újra be kell jelentkeznie, egyébként "Hozzáférés megtagadva" üzenetet írunk ki.

Új felhasználóként a jobb oldalon található Regisztráció gombra kattintva jutunk el a regisztrációs felületre.



## Regisztráció

Regisztrációhoz 3 kötelező adatot kell megadni: felhasználónév, e-mail cím és jelszó. Regisztráció gombra adatbázisban ellenőrizzük az e-mail cím foglaltságát, amennyiben foglalt akkor másikat kell választani. Jelszó erősségre jelenleg nincsenek megkötések, egyszerű jelszavak is megadhatóak, ezzel együtt a jelszavak biztonságos tárolására a bcrypt jelszó algoritmusát használjuk. Ez fontos a biztonság szempontjából, így illetéktelen adatbázis hozzáférés esetén a támadók nem tudják visszafejteni az adatbázisban tárolt jelszavakat.



## Kezdőlap

A felület felső részén található a menüsor. Az oldal betöltésekor lefut egy ellenőrzés ha 10 napnál régebben rögzítettünk utoljára mérőállást és erről egy információs ablakban tájékoztatjuk a felhasználót.



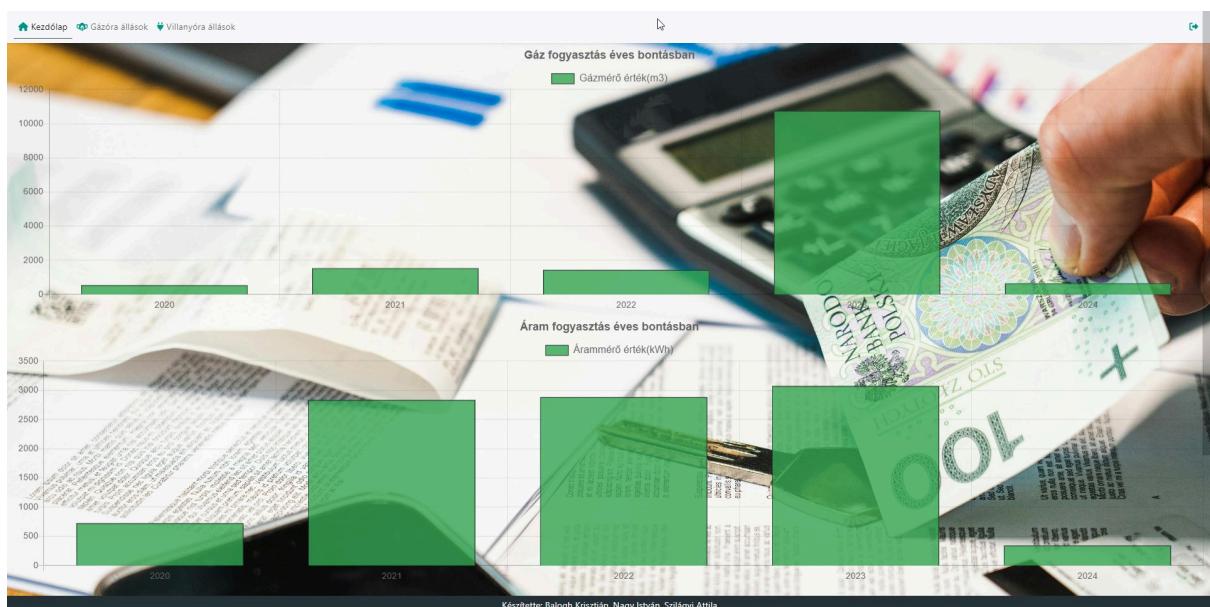


A legutóbbi gázmérő mérési adat már több mint 10 napos!



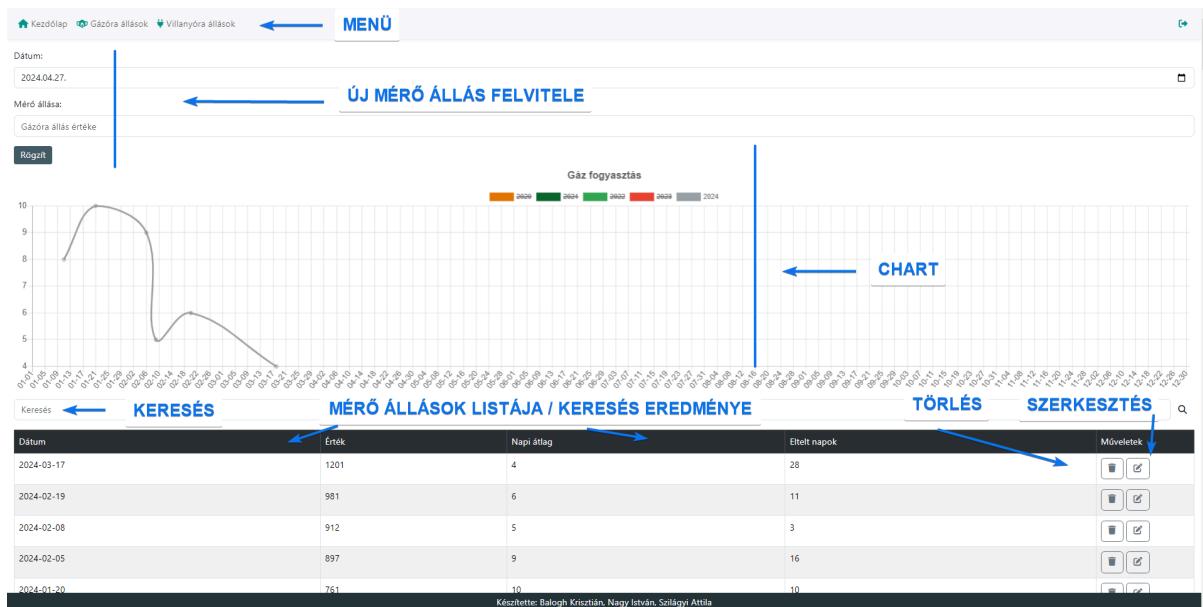
A legutóbbi árammérő mérési adat már több mint 10 napos!

A menüsor alatt látható a Gáz és Villanyóra fogyasztás éves összesítése. Évenkénti bontásban oszlopos formában láthatjuk a szumma értéket, így jól látható ha az időszakban csökkent vagy nőtt az összfogyasztás.



## Gázóra állások

A gázóra állások oldal felépítését az alábbi képen részletezem:



A menüsor alatt találjuk a gázóra állások beviteli mezőit: Dátum és Mérő állása, minden kötelező. Rögzítés gombra kattintva ellenőrizzük a dátum érvényességét, az óraállás nem lehet negatív, csak számok beírása engedélyezett.

Kezdőlap Gázóra állások Villanyóra állások

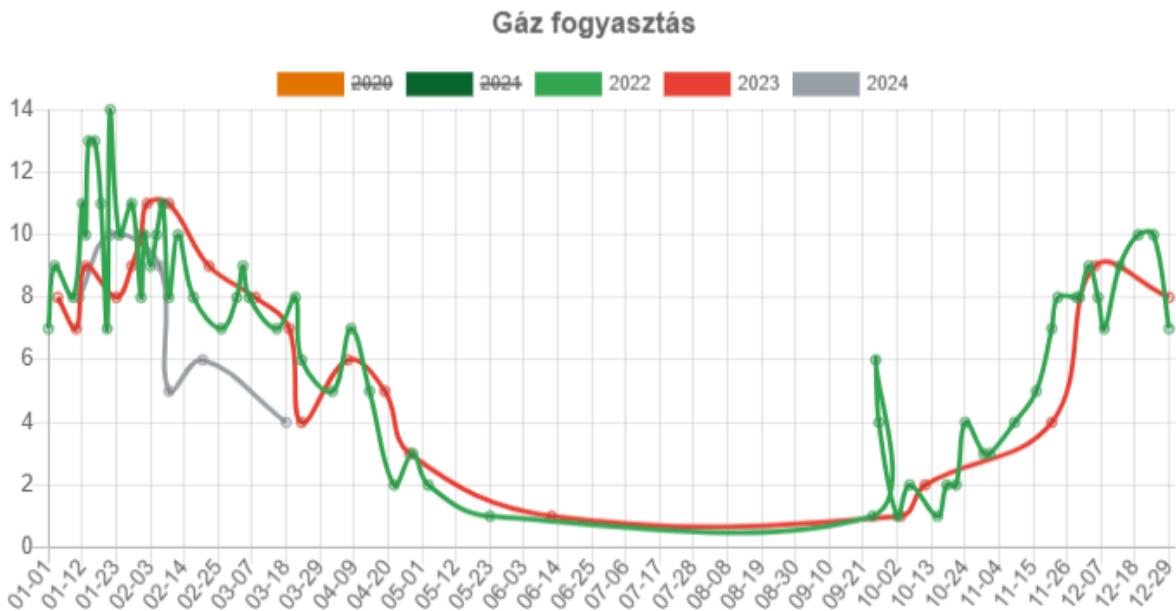
Dátum: 2024.04.27.

Mérő állása:

Gázóra állás értéke

Rögzít

Gáz fogyasztás chart betöltéskor az aktuális év napi átlag fogyasztását mutatja. A jelmagyarázatban látható évszámokra kattintással be lehet kapcsolni az adott év megjelenítését. Fontos, hogy a chart nem a bevitt mérőállásokat, hanem azokból számolt napi átlagfogyasztást mutatja.



A keresés funkcióval bevitel mezőjébe írt adat a mérőállás dátum és érték mezőjében egyaránt keres, a keresés eredményét az alatt lévő táblázatba írjuk ki. Ha a kereső mező üres, akkor az összes mérőállás kilistázzuk.

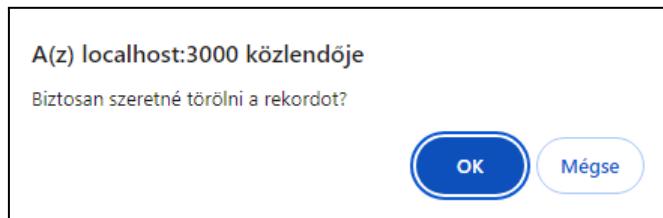
🔍

Az oldal betöltésekor a keresés mező alatti táblázatba listázzuk az összes óraállást. A táblázat oszlopai:

- Dátum: óraállás dátuma
- Érték: felhasználó által rögzített óraállás
- Napi átlag: a legutolsó mérőállás óta eltelt napok arányában számolt átlag fogyasztás
- Eltelt napok: az előző fogyasztási adat felvitele óta eltelt napok száma
- Műveletek:
  - Törlés
  - Szerkesztés

Dátum	Érték	Napi átlag	Eltel napok	Műveletek
2024-04-26	1210	0	40	
2024-03-17	1201	4	28	
2024-02-19	981	6	11	
2024-02-08	912	5	3	
2024-02-05	897	9	16	
2024-01-20	761	10	10	

Törlés gombra kattintáskor a böngésző a törlés megerősítését kéri a felhasználótól.



A szerkesztés gombra kattintva egy felugró ablakban módosíthatjuk a mérőállás dátumát és értékét.

The dialog has the following fields:  
Mérőállás módosítása  
Dátum: 2024.04.26.  
Új mérőállás: 1210  
Rögzít

## Villanyóra állások

A villanyóra állások oldal felépítése és működése megegyezik a Gázóra állások oldallal.

## Kijelentkezés

A kijelentkezés gomb a menüsor jobb szélén található ikonra kattintással indítható el. Hatására megtörténik a felhasználóhoz létrehozott cookie törlése, így ezután csak új bejelentkezés után tudjuk használni a védett oldalakat (Kezdőlap, Gáz-, Villanyóra állások).



## 4. Tesztelési dokumentáció

Webes alkalmazásunkat 3 böngészőben teszteltük. Az oldalak felépítését, kinézetét illetően minimális eltérések voltak. Az oldal funkcióit, műveleteit minden böngésző ugyanúgy dolgozza fel, ugyanazokat az üzeneteket adják vissza.

Tesztelt böngészők:

- Microsoft Edge | Verzió: 124
- Opera | Verzió: 109
- Chrome | Verzió: 124

Részletes teszteset lista a Teszteredmények dokumentációja mellékletben található.

Manuális teszteléssel végrehajtott teszteset száma: 27 darab

Sikeress: 21 darab

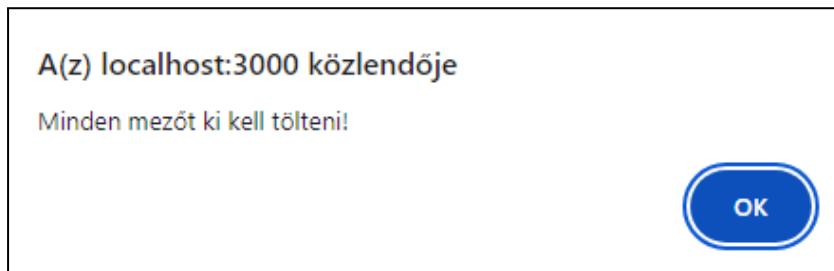
Sikertelen: 6 darab

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4												
5	<b>Teszteredmények dokumentációja</b>											
6	<b>Aalkalmazás neve</b>	Rezsi App	<b>Teszter</b>	Szilágyi Attila								
7	<b>Prioritás</b>	Magas	<b>Teszteset reviewer</b>	Balogh Krisztián, Nagy István								
8	<b>Tesztelés típusa</b>	Manuális tesztelés	<b>Teszteset verzió</b>	1.5								
9	<b>Tesztelés célja</b>	Form és funkcionális hibakeresés	<b>Teszteseti időszak</b>	2024.04.22-2024.04.29								
10	<b>Teszteset azonosító</b>	<b>Teszteset leírása</b>	<b>Előfeltétel</b>	<b>Teszteset lépései</b>	<b>Tesz adat</b>	<b>Elvárta eredmény</b>	<b>Aktuális eredmény</b>	<b>Státusz</b>	<b>Hiba prioritása</b>	<b>Jegyzet</b>		
22	TC_Gaz_03	Gázóra állás rogzítése	Gázóra állások oldala navigálás	1. Helyes dátum és pozitív szám a mérőlábas mezőben -> Rögzítés gombra kattintás	1. 2024.04.27/1220	Mérőlábas bekerül az adatbázisba chart azon belül frissül. Kereső frissítés után elbúsztat azonnal frissítés és tartalmazza az új mérőlálist	Elvárta eredménynek megfelelő működés	<span style="color: green;">Sikeress</span>	-			
23	TC_Gaz_04	Gázfogyasztás chart szériák látthatóságának kapcsolása	Gázóra állások oldala navigálás és korábbi évekre visszamenőleg felvitt adatok	1. Korábbi évek feliratára kattintással be és kilépésre kattintás	Adatbázisban korábban eltárolva	A szériák megjelennek és elűnnék, az összes széria egysidőben is látható	Elvárta eredménynek megfelelő működés	<span style="color: green;">Sikeress</span>	-			
24	TC_Gaz_05	Keresés	Van rögzítve gáz fogadástári adat	1. Keresés mezőbe írás után enter vagy kerések ikonra kattintás	1. 1210 2. 2024.04.27	1 rekord megjelenítése a táblázatban	Elvárta eredménynek megfelelő működés	<span style="color: green;">Sikeress</span>	-			
25	TC_Gaz_06	Keresés eredménynek törlése, összes adat újból megjelenítése	Keresett adat szerepel a mérőlábsok táblázatban	1. Keresés mezőbe írás után enter vagy kerések ikonra kattintás 2. Keresés feltétel törlése után enter vagy kerések ikonra kattintás	1. 1210 2. üres	Az oldal megnyitásakor betöltött táblálmú táblázat megjelenítése	Az oldal megnyitásakor betöltött táblálmú táblázat megjelenítése	<span style="color: red;">Sikertelen</span>	<span style="color: blue;">Normal</span>	<p>F5 gomb lenyomása nélkül kell visszakapni a helyes rendezést és minden oszlop adattal feltölve legyen.</p>		

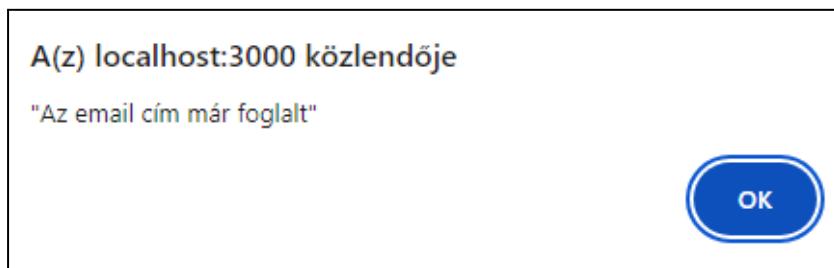
## Weboldalon megjelenő hibaüzenetek

### Regisztráció

Új felhasználó regisztrálásakor ha a 3 kötelező mező közül valamelyiket üresen hagyjuk.



Új felhasználó regisztrálásakor ha korábban már felhasznált e-mail címmel szeretnénk regisztrálni.



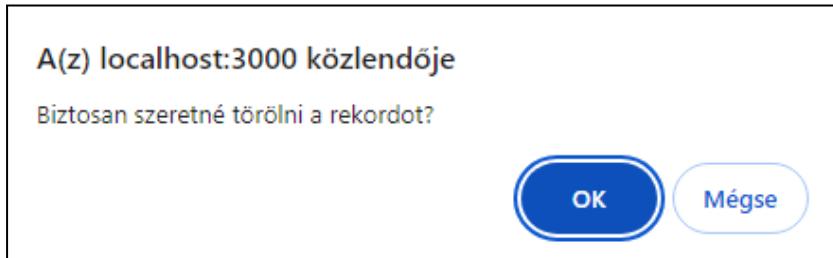
### Bejelentkezés

Ha a bejelentkezés során üresen vagy tévesen adunk meg e-mail címet vagy jelszót.



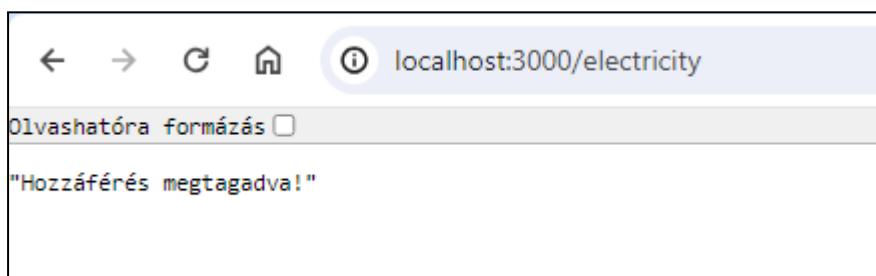
## Gázóra állások / Villanyóra állások oldal

Ha egy meglévő rekordot törölni szeretnénk, akkor egy megerősítő üzenetet követően lehet törölni.

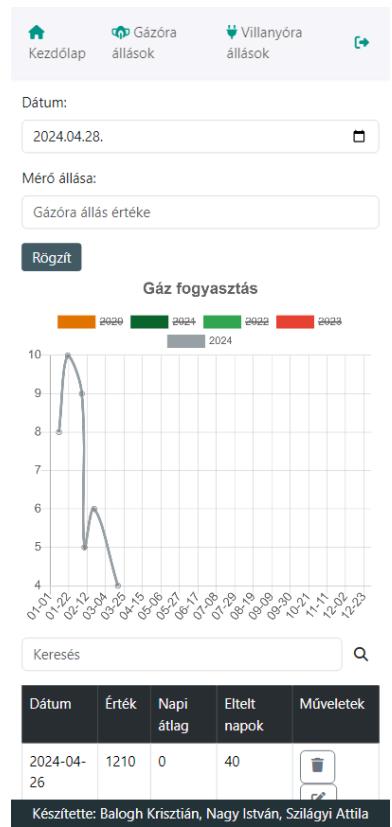
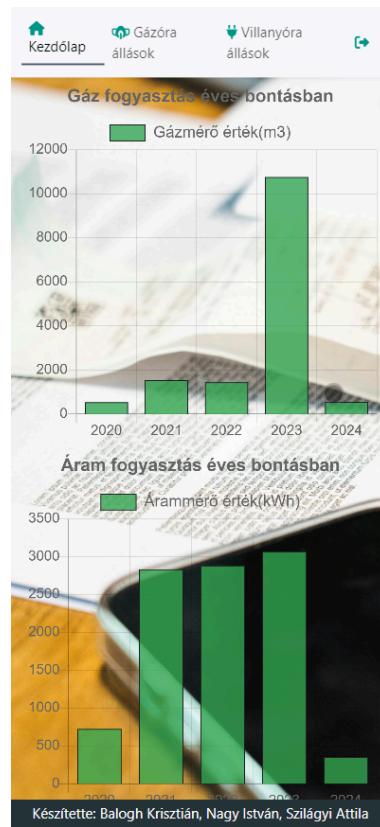
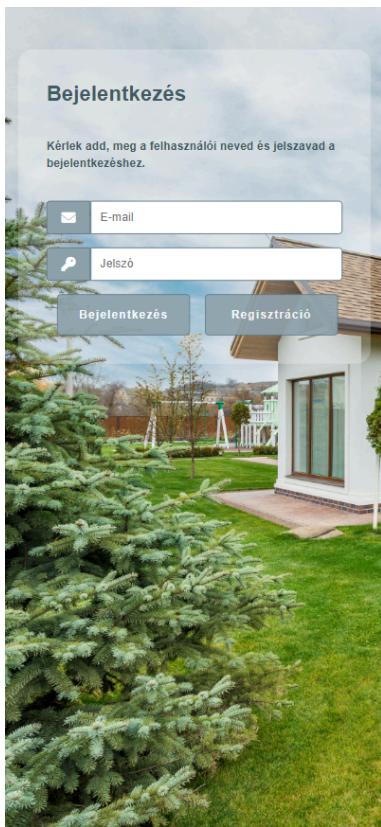


## Védett oldalak

Ha a védett oldalakat böngésző előzményekből szeretnénk megnyitni, de előtte nem történt belépés vagy már kijelentkeztünk a Kilépés gombbal.



## Reszponzív nézetek



## Fejlesztési lehetőségek

- mérőállások adattáblák UserID-val bővítése és a felhasználói felületen csak a bejelentkezett felhasználó adatainak megjelenítése
- mérőállások felvitele
  - előfordulhat, hogy egy felhasználónak aki 1 e-mail címmel regisztrál, több mérőrája is van, akár több ingatlanja, így több gázórája, amit egy account alatt kezelhet
  - emellett más közmű mérők is felvihetőek lehetnek, pl.: vízóra
- mérőállások cseréjének kezelése: időnként a mérőrákat cserélik és a mérőállás 0 értékkel indul, ez a charton megjelenő adatokat eltorzíthatja
- védett oldalak szövegének bővítése: a leggyakrabban előforduló jelenség ha a cookie lejárt és újra be kell lépni, ebben az esetben automatikusan átirányíthatjuk a bejelentkezésre
- elfelejtett jelszó funkció bevezetése
- keresés funkció bővítése dátum tól-ig és mérőállás értéke tól-ig

## 5. Irodalomjegyzék

- <https://fontawesome.com/>
- <https://www.w3schools.com>
- <https://www.chartjs.org/>
- <https://www.pexels.com/hu-hu/>