Ez a dokumentáció a **Quick Serve** backend részét írja le, amely lehetővé teszi a felhasználók számára a regisztrációt, bejelentkezést, rendeléskezelést és felhasználói adatok lekérdezését. Az alkalmazás Node.js és Express keretrendszert használ, valamint MySQL adatbázist a tartalom tárolásához.

**Főbb Funkciók**

1. **Regisztráció**
   * A felhasználóknak szükségük van egy fiókra, hogy rendeléseket adhassanak le.
   * **Működés**: A felhasználó megadja adatait (név, telefonszám, lakcím, email, jelszó, osztály). A jelszót bcrypt segítségével hash-eljük, így biztonságosan tárolható az adatbázisban.
2. **Bejelentkezés**
   * A felhasználóknak be kell jelentkezniük a fiókjukhoz való hozzáféréshez.
   * **Működés**: A felhasználó név és jelszó megadása után az adatbázisból lekérjük az adatokat. A jelszó hash-ellenőrzése után JWT (JSON Web Token) generálódik, amely azonosítja a felhasználót a további kérések során.
3. **Felhasználói adatok lekérdezése**
   * A felhasználóknak szükségük van a profiljukhoz és rendeléseikhez való hozzáférésre.
   * **Működés**: Bejelentkezett felhasználók számára elérhetők a profilinformációk és a korábbi rendelések.
4. **Rendeléskezelés**
   * A felhasználóknak lehetőségük van rendeléseik leadására és állapotuk nyomon követésére.
   * **Működés**:
     + **Rendelés elküldése**: A felhasználó által megadott termékek adatainak rögzítése az adatbázisban.
     + **Rendelés részleteinek lekérdezése**: A felhasználók megnézhetik rendeléseik részleteit.
     + **Rendelés státuszának módosítása**: Az adminisztrátorok frissíthetik a rendelés állapotát.

**Adatbázis Tervezés**

Az alkalmazás MySQL adatbázist használ, amely a következő táblákat tartalmazza:

* **Vásárlók**: Felhasználói információk tárolása.
* **Eladók**: Az eladók adatai.
* **Megrendelések**: Rendelések rögzítése.
* **Rendelés\_állapot**: A rendelés állapotának nyomon követése.
* **Termékek**: A büfé kínálatának adatai.

**Kérés Módszerek és Végpontok, pl:**

* **POST /register**: Regisztrációs adatok elküldése.
* // Regisztrációs végpont
* app.post("/register", async (req, res) => {
* const { Név, Telefonszám, Lakcím, Email, Jelszó, Osztály } = req.body; // Kéréstől érkező adatok
* const hashedPassword = await bcrypt.hash(Jelszó, 10); // Jelszó hash-elése
* const sql = "INSERT INTO Vásárlók (Név, Telefonszám, Lakcím, Email, Jelszó, Osztály) VALUES (?, ?, ?, ?, ?, ?)"; // SQL lekérdezés
* // Adatok betöltése az adatbázisba
* db.query(sql, [Név, Telefonszám, Lakcím, Email, hashedPassword, Osztály], (err, result) => {
* if (err) return res.status(500).json({ error: "Hiba a regisztrációnál" }); // Hiba esetén válasz
* res.json({ message: "Sikeres regisztráció!" }); // Sikeres válasz
* });
* });
* **POST /login**: Bejelentkezési adatok elküldése.
* // Bejelentkezési végpont
* app.post("/login", (req, res) => {
* const { Név, Jelszó } = req.body; // Kéréstől érkező adatok
* const sql = "SELECT \* FROM vásárlók WHERE Név = ?"; // SQL lekérdezés
* // Felhasználói adatok keresése
* db.query(sql, [Név], async (err, results) => {
* if (err || results.length === 0) return res.status(400).json({ error: "Hibás email vagy jelszó" }); // Hiba kezelése
* const user = results[0]; // Felhasználó adatok
* const isMatch = await bcrypt.compare(Jelszó, user.Jelszó); // Jelszó ellenőrzése
* if (!isMatch) return res.status(400).json({ error: "Hibás email vagy jelszó" }); // Hiba kezelése
* // JWT token létrehozása
* const token = jwt.sign({ Vásárló\_ID: user.Vásárló\_ID, Név: user.Név }, "secret", { expiresIn: "1h" });
* res.json({ token, user: { name: user.Név } }); // Válasz tokennel és felhasználóval
* });
* });
* **GET /user**: Bejelentkezett felhasználó adatok lekérdezése.
* // Felhasználói adatok lekérdezése
* app.get("/user", (req, res) => {
* const token = req.headers.authorization; // Token kérésektől
* if (!token) return res.status(401).json({ message: "Nincs bejelentkezve" }); // Hiba kezelése
* // Token ellenőrzése
* jwt.verify(token, "secret", (err, decoded) => {
* if (err) return res.status(403).json({ message: "Érvénytelen token" }); // Hiba kezelése
* res.json({ Név: decoded.Név, Id: decoded.Vásárló\_ID }); // Sikeres válasz
* });
* });
* **GET /admin/orders**: Rendelések listázása adminisztrátor számára.
* // Rendelések lekérdezése
* app.get("/admin/orders", (req, res) => {
* const query = "SELECT megrendelések.Megrendelés\_ID, megrendelések.Dátum, megrendelések.Összeg FROM megrendelések"; // SQL lekérdezés
* db.query(query, (err, results) => {
* if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése
* res.json(results); // Sikeres válasz
* });
* });
* **PUT /admin/sendstatus/:rendelesid**: Rendelés státuszának frissítése.
* // Rendelés státuszának módosítása
* app.put("/admin/sendstatus/:rendelesid", (req, res) => {
* const { rendelesid } = req.params; // Rendelés azonosító kérésektől
* const { Státusz } = req.body; // Státusz kéréstől
* const query = "UPDATE rendelés\_állapot SET Státusz=? Where rendelés\_állapot.Megrendelés\_ID=?;"; // SQL lekérdezés
* db.query(query, [Státusz, rendelesid], (err, results) => {
* if (err) console.log(err); // Hiba kiírása
* res.json({ message: "A státuszt sikeresen frissítettük", Státusz: Státusz }); // Sikeres válasz
* });
* });
* **GET /admin/orderdetails/userinf/:rendelesid**: Rendelés részleteinek lekérdezése.

// Rendelés részleteinek lekérdezése

app.get("/admin/orderdetails/userinf/:rendelesid", (req, res) => {

  const { rendelesid } = req.params; // Rendelés azonosító kérésektől

  const query = "SELECT vásárlók.név, vásárlók.email, vásárlók.telefonszám, vásárlók.Osztály, megrendelések.Megrendelés\_ID FROM vásárlók INNER JOIN megrendelések ON vásárlók.Vásárló\_ID = megrendelések.Vásárló\_ID WHERE megrendelések.Megrendelés\_ID=?"; // SQL lekérdezés

  db.query(query, [rendelesid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json(results); // Sikeres válasz

  });

});

* **GET /admin/orderdetails/products/:rendelesid**: Rendeléshez tartozó termékek lekérdezése.

// Rendeléshez tartozó termékek lekérdezése

app.get('/admin/orderdetails/products/:rendelesid', (req, res) => {

  const { rendelesid } = req.params; // Rendelés azonosító kérésektől

  const query = "SELECT termékek.terméknév, termékek.kategória, termékek.típus, termékek.egységár, megrendelések.cikkszám, megrendelések.mennyiség, SUM(termékek.egységár\*megrendelések.mennyiség) As Összeg, megrendelések.dátum FROM megrendelések INNER JOIN termékek ON megrendelések.cikkszám=termékek.cikkszám WHERE megrendelések.Megrendelés\_ID=? GROUP By termékek.cikkszám;"; // SQL lekérdezés

  db.query(query, [rendelesid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json(results); // Sikeres válasz

  });

});

* **PUT /admin/** **sendstatus/:rendelesid**: Rendelés státuszának módosítása.

// Rendelés státuszának módosítása

app.put("/admin/sendstatus/:rendelesid", (req, res) => {

  const { rendelesid } = req.params; // Rendelés azonosító kérésektől

  const { Státusz } = req.body; // Státusz kéréstől

  const query = "UPDATE rendelés\_állapot SET Státusz=? Where rendelés\_állapot.Megrendelés\_ID=?;"; // SQL lekérdezés

  db.query(query, [Státusz, rendelesid], (err, results) => {

    if (err) console.log(err); // Hiba kiírása

    res.json({ message: "A státuszt sikeresen frissítettük", Státusz: Státusz }); // Sikeres válasz

  });

});

* **GET /** **userinfo/:vasarloid**: Bejelentkezett felhasználó adatainak lekérdezése.

// Bejelentkezett felhasználó adatainak lekérdezése

app.get("/userinfo/:vasarloid", (req, res) => {

  const { vasarloid } = req.params; // Vásárló azonosító kérésektől

  const query = "SELECT vásárlók.név, vásárlók.osztály, vásárlók.email, vásárlók.telefonszám, vásárlók.lakcím FROM vásárlók WHERE vásárlók.Vásárló\_ID=?"; // SQL lekérdezés

  db.query(query, [vasarloid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json(results); // Sikeres válasz

  });

});

* **PUT /** **userinfo/changepassword/:vasarloid**: Jelszó módosítása.

// Jelszó módosítása

app.put("/userinfo/changepassword/:vasarloid", (req, res) => {

  const { vasarloid } = req.params; // Vásárló azonosító kérésektől

  const { Jelszó } = req.body; // Új jelszó kéréstől

  const query = "UPDATE vásárlók SET Jelszó=? Where vásárlók.Vásárló\_ID=?;"; // SQL lekérdezés

  db.query(query, [Jelszó, vasarloid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json({ message: "A jelszót sikeresen frissítettük" }); // Sikeres válasz

  });

});

* **GET /** **userorders/:vasarloid**: Vásárló rendeléseinek lekérdezése.

// Vásárló rendeléseinek lekérdezése

app.get("/userorders/:vasarloid", (req, res) => {

  const { vasarloid } = req.params; // Vásárló azonosító kérésektől

  const query = "SELECT megrendelések.Megrendelés\_ID, megrendelések.Dátum, SUM(termékek.egységár\*megrendelések.mennyiség) As Összeg FROM megrendelések INNER JOIN vásárlók ON vásárlók.Vásárló\_ID=megrendelések.Vásárló\_ID INNER JOIN termékek ON termékek.cikkszám=megrendelések.Cikkszám WHERE vásárlók.Vásárló\_ID=? GROUP BY megrendelések.Megrendelés\_ID"; // SQL lekérdezés

  db.query(query, [vasarloid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json(results); // Sikeres válasz

  });

});

* **GET /** **userorders/details/:rendelesid/:vasarloid**: Rendelés részleteinek lekérdezése.

// Rendelés részleteinek lekérdezése

app.get("/userorders/details/:rendelesid/:vasarloid", (req, res) => {

  const { rendelesid } = req.params; // Rendelés azonosító kérésektől

  const { vasarloid } = req.params; // Vásárló azonosító kérésektől

  const query = "SELECT megrendelések.Megrendelés\_ID, termékek.terméknév, termékek.kategória, termékek.típus, termékek.egységár, megrendelések.cikkszám, megrendelések.mennyiség, SUM(termékek.egységár\*megrendelések.mennyiség) As Összeg, megrendelések.dátum FROM megrendelések INNER JOIN termékek ON megrendelések.cikkszám=termékek.cikkszám INNER JOIN vásárlók ON vásárlók.Vásárló\_ID=megrendelések.Vásárló\_ID WHERE megrendelések.Megrendelés\_ID=? and vásárlók.Vásárló\_ID=? GROUP BY termékek.terméknév"; // SQL lekérdezés

  db.query(query, [rendelesid, vasarloid], (err, results) => {

    if (err) return res.status(500).json({ error: "Hiba az adatok lekérdezésekor" }); // Hiba kezelése

    res.json(results); // Sikeres válasz

  });

});

* **POST /** **sendorder**: Rendelés elküldése.

// Rendelés elküldése

app.post("/sendorder", (req, res) => {

  const termekek = req.body; // Kéréstől érkező termékek

  const sql1 = "INSERT INTO megrendelések(Megrendelés\_ID, Vásárló\_ID, Cikkszám, Mennyiség, Szünet) VALUES ?"; // SQL lekérdezés

  const sql2 = "INSERT INTO rendelés\_állapot(Megrendelés\_ID, Státusz) VALUES (?, 'Rendelés elküldve')"; // SQL lekérdezés

  // Termékek tömbje SQL lekérdezéshez

  const values = termekek.map(t => [t.Megrendelés\_ID, t.Vásárló\_ID, t.Cikkszám, t.Mennyiség, t.Szünet]);

  // Rendelés rögzítése

  db.query(sql1, [values], (err, results) => {

    if (err) {

      console.error("SQL hiba az első lekérdezésnél:", err); // Hiba kiírása

      return res.status(500).json({ error: "Hiba a rendelés rögzítésekor" }); // Hiba válasz

    }

    // Rendelés állapotának rögzítése

    db.query(sql2, [termekek[0].Megrendelés\_ID], (err2, results2) => {

      if (err2) {

        console.error("SQL hiba a rendelés állapot rögzítésekor:", err2); // Hiba kiírása

        return res.status(500).json({ error: "Hiba a rendelés állapot rögzítésekor" }); // Hiba válasz

      }

      res.json({ message: "A rendelést és az állapotot sikeresen rögzítettük!" }); // Sikeres válasz

    });

  });

});

**Biztonsági Szempontok**

* **Jelszó Hash-elés**: A jelszavakat bcrypt segítségével hash-eljük a tárolás előtt.
* **JWT Token**: A felhasználók azonosítására és a biztonságos hozzáférés biztosítására használjuk.