

Kisvárdai SZC Fehérgyarmati Petőfi Sándor Technikum

Szoftverfejlesztő és -tesztelő szak



AZ ISKOLAI BÜFÉ MŰKÖDSÉNEK OPTIMALIZÁLÁSA  
WEBALAPÚ PLATFORM FEJLESZTÉSÉVEL ÉS  
IMPLEMENTÁLÁSÁVAL

Konzulens:

Lakatos Sándor

Berki Balázs

Készítette:

Markulinec Viktória

Molnár Péter

FEHÉRGYARMAT, 2025

# Tartalomjegyzék

<b>Bevezetés.....</b>	<b>3</b>
1. Miért készült ez a projekt? .....	3
2. Célkitűzés .....	3
3. Célközönség .....	3
<b>Projektleírás .....</b>	<b>4</b>
1. Funkcionalitás .....	4
2. Felhasználói folyamat .....	4
3. Előnyök .....	5
<b>Technológiai háttér .....</b>	<b>6</b>
Használt technológiák .....	6
a) Node.js .....	6
b) MySQL .....	10
c) HTML, CSS, Bootstrap .....	13
e) React Native .....	17
f) Jest .....	18
<b>Fejlesztési folyamat .....</b>	<b>19</b>
1. Idővonal .....	19
2. Kihívások és megoldások .....	21
<b>Tesztelés .....</b>	<b>21</b>
1. Tesztelési módszerek .....	21
2. Eredmények .....	23
<b>Jövőkép.....</b>	<b>25</b>
<b>A Quick Serve vásárlói oldala: .....</b>	<b>25</b>
<b>A Quick Serve eladói oldala: .....</b>	<b>28</b>
<b>Irodalomjegyzék.....</b>	<b>30</b>
<b>Ábrajegyzék .....</b>	<b>30</b>

# Bevezetés

## 1. Miért készült ez a projekt?

A **Quick Serve** projekt ötlete abból a mindennapi problémából fakadt, amelyet az iskolai büfék működése során tapasztaltunk saját és diáktársaink példájából. Ugyanis a szünetekben a diákok hosszú percekig töltnek sorban állással, miközben az eladók kapkodva próbálják kiszolgálni a hirtelen érkező tömeget. Előfordulhat olyan eset is, amikor egy diák nem kerül az adott szünetben sorra, így kénytelen a következőben újra sorba állnia.

Ez nemcsak időpazarlást jelent a diákok számára, hanem stresszt és nehézségeket okoz a büfében dolgozóknak is. Ebből a problémából ismertük fel, hogy egy digitális megoldással ezt a folyamatot hatékonyabbá, gyorsabbá és kényelmesebbé tehetjük mindkét fél számára. Így született meg a Quick Serve ötlete, amely egy modern, webalapú platformot kínál az iskolai büfé működésének optimalizálására.

## 2. Célkitűzés

A **Quick Serve** célja, hogy egyszerűsítse és felgyorsítsa az iskolai büfé rendelési és kiszolgálási folyamatait. Szerettük volna elérni, hogy a diákok előre leadhassák rendeléseiket egy könnyen használható online felületen keresztül, ezzel megszüntetve a szünetekben kialakuló sorokat és torlódásokat. További célunk volt, hogy az eladó számára átláthatóbbá és hatékonyabbá tegyük a napi teendőket, például a rendelések kezelését és a készletgazdálkodást. A rendszer tervezése során arra törekedtünk, hogy mind a diákok, mind az eladó számára jobb élményt nyújtsunk, miközben csökkentjük a hibalehetőségeket és növeljük az elégedettséget.

## 3. Célközönség

A **Quick Serve** elsősorban az iskolai diákokat és a büfében dolgozó eladót célozza meg. A diákok számára a platform egy kényelmes és gyors megoldást kínál, amely illeszkedik a fiatalabb korosztály digitális szokásaihoz, például lehetővé téve számukra, hogy a nap bármely pontján, akár órák között is rendeljenek. Az eladó számára pedig egy olyan eszközt biztosít, amely segít a rendelések rendszerezésében, a kiszolgálás ütemezésében és a készletek optimalizálásában. A rendszer diákbarát dizájnja és az eladó számára kialakított átlátható felület egyaránt azt szolgálja, hogy mindkét csoport hatékonyan tudja használni a platformot.

# Projektleírás

## 1. Funkcionalitás

A **Quick Serve** egy webalapú platform, amelyet az iskolai büfé rendelési és kiszolgálási folyamatainak optimalizálására fejlesztettünk ki. A rendszer lehetővé teszi a diákok számára, hogy előre, online megrendeljék a büfé termékeit, megadva az átvétel kívánt időpontját (egy konkrét szünetet) és a fizetési módot. A diákok kedvenc termékeiket megjelölhetik a gyorsabb rendelés érdekében, sőt, kuponokat is használhatnak, amelyeket az eladók hozhatnak létre kedvezmények biztosítására. Ezeket addig lehet felhasználni, ameddig az eladó a saját felületén ki nem törli.

Az eladók egy átlátható felületen kezelhetik a beérkező rendeléseket, frissíthetik azok státuszát (pl. "Elkészült"), és e-mailes értesítésekkel informálhatják a diákokat. A Quick Serve így egy hatékony, mindkét fél számára előnyös megoldást kínál az iskolai büfék mindennapi működéséhez.

## 2. Felhasználói folyamat

A **Quick Serve** használata egyszerű és átlátható folyamatot követ mind a diákok, mind az eladó számára. A diák először regisztrál az oldalon általános adatainak megadásával, majd bejelentkezik, hogy hozzáférjen a rendelési felülethez. Ezután böngészhet a büfé kínálatában, bizonyos termékeket kedvencként is megjelölhet, ami az oldal elején listázódik ki, a kívánt termékeket pedig kosárba helyezheti, emellett a kosár tartalmát is módosíthatja, ahol az összesített összeg ennek megfelelően frissül.

A rendelés véglegesítésekor kiválasztja az átvételi időpontot és a fizetési módot, ha van aktiválja a kuponkódot, ami után az adott érték levonásra kerül az összegből, így láthatja mennyibe kerül az eredeti rendelése és azt is, hogy a kupon felhasználása után mennyire csökkent, majd leadja a rendelést, ha már nem szeretne módosítani rajta, erről azonnali értesítést kap az oldalon és e-mailben is. A korábbi rendelései pedig bármikor visszanézhetők a profiljában, ahol láthatja milyen státuszban áll a rendelése.

Az eladók bejelentkezés után megtekinthetik a beérkező rendeléseket, frissíthetik azok státuszát, és kuponokat hozhatnak létre és törölhetnek. A rendszer pedig egy egyedi rendelési azonosítóval és automatikus adatellenőrzéssel biztosítja a zökkenőmentes működést.

Emellett van egy mobilalkalmazásunk is, ahol a regisztrált felhasználók egy bejelentkezés követően megtekinthetik a leadott rendeléseiket kilistázva, amelyeket a részletes adataikkal is megtekinthetnek.

### 3. Előnyök

A **Quick Serve** számos előnnyel jár a diákok és az eladók számára egyaránt:

- **Időmegtakarítás:** A diákok előre rendelhetnek, így a szünetekben nem kell sorban állniuk, az eladó pedig hatékonyabban ütemezheti a kiszolgálást.
- **Rövidebb várakozási idő:** A pontos átvételi időkkal megszűnik a büfé előtti torlódás, és a diákok azonnal átvehetik rendeléseiket.
- **Jobb kiszolgálás:** Az eladó átláthatóan követheti a rendeléseket, optimalizálhatja a készleteket, és pontosan teljesítheti a kéréseket.
- **Kényelem:** A diákok szabadon választhatják ki, melyik szünetben szeretnék átvenni a rendelést.
- **Átláthatóság:** A rendelések nyomon követhetők, az eladó pedig azonnal jelezheti a változásokat.
- **Kevesebb hibalehetőség:** Az előre rögzített rendelések csökkentik a félreértéseket, növelve a kiszolgálás megbízhatóságát.
- **Növekvő elégedettség:** A gyors és rugalmas folyamat jobb élményt nyújt mindkét félnek.
- **Okosabb készletgazdálkodás:** Az eladók idővel látják, mely termékekből érkezik sok rendelés, és melyeket kerülnek a diákok. Ez alapján pontosabban rendelhetnek utánpótlást, elkerülve, hogy népszerűtlen termékekből felesleg halmozódjon fel, amit később kidobni kell, ha lejár a szavatossága.
- **Diákbarát dizájn:** Az oldal színes, átlátható felülete kifejezetten a fiatalabb korosztály számára készült, hogy gyorsan eligazodjanak rajta, akár először használják, akár rutinos rendelők. De az eladók számára is törekedtünk az átlátható megjelenésben.

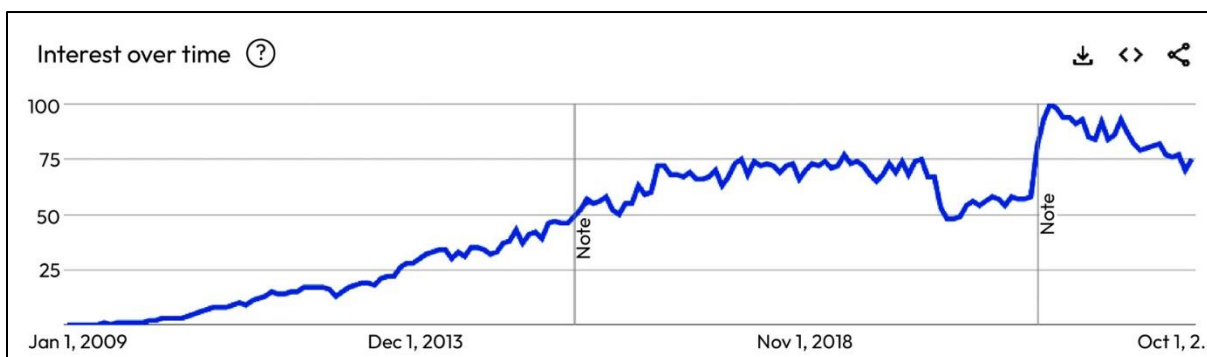
# Technológiai háttér

## Használt technológiák

### a) Node.js

A **Quick Serve** fejlesztéséhez olyan technológiákat választottunk, amelyek biztosítják a rendszer stabilitását, skálázhatóságát és valós idejű funkcionalitását. A backend alapja a **Node.js**, ami egy aszinkron, eseményvezérelt JavaScript környezet, amely kiválóan kezeli a párhuzamos kéréseket, így ideális egy több felhasználót kiszolgáló platformhoz. Azért választottuk, mert gyors és egyszerűen használható szerveroldali alkalmazásokhoz.

Az 1.1 ábrán látható, hogy a Node.js népszerűsége az idők során hogyan emelkedett, és napjainkban is folyamatosan, dinamikusan nő.



1.1.ábra

A Node.js egy **nyílt forráskódú**, több platformon futó JavaScript futtatókörnyezet, amely a V8 JavaScript-motorra épül – ugyanarra a motorra, amely például a Google Chrome-ot és a Microsoft Edge-et is hajtja. Ez egy könnyű és gyors futtatókörnyezet, amely egyszálú architektúrával működik, így nem szükséges minden egyes kéréshez új szál indítani, szemben más elterjedt technológiákkal, mint például a PHP. Ez jelentős előnyt biztosít, mivel alacsony memóriahasználat mellett kiemelkedően magas teljesítményt nyújt.

Jól működik adatbázisokkal, mint a MySQL, és sok csomag (pl. Express, bcrypt) könnyen integrálható vele. **Aszinkron**, tehát nem várja meg, hogy egy művelet (pl. adatbázis lekérdezés) befejeződjön, hanem közben más feladatokat is végez. A Node npm csomagkezelője segítségével külső könyvtárakat is felhasználtunk, mint például az Express, MySQL, bcrypt, JWT és CORS.

A Node.js **2009-es megjelenése** igazi áttörést hozott a webfejlesztés világában, amit Ryan Dahl, a technológia megalkotója egy akkoriban rendhagyónak számító egyszálú architektúrával ért el. A Node.js egy rendkívül gazdag ökoszisztémával büszkélkedhet, amely csomagokból, JavaScript-könyvtárakból és a közösség által létrehozott erőforrásokból áll, így bármilyen típusú alkalmazás könnyedén elkészíthető vele.

Többplatformos természetű, vagyis a modern piacon elterjedt bármely operációs rendszeren és architektúrán futtatható, nem kizárólag webalkalmazások fejlesztésére alkalmas: egyszerű parancssori eszközöktől kezdve egészen olyan komplex asztali programokig, mint a Slack vagy a Visual Studio Code, sokféle alkalmazás készíthető vele.

A Node.js a **JavaScriptre épül**, amely napjaink egyik legelterjedtebb programozási nyelve, így a fejlesztők milliói számára már ismerős, ami egyszerű tanulási görbét biztosít. Jeff Atwood híres mondása, az Atwood törvénye szerint: „Minden olyan alkalmazás, amely megírható JavaScriptben, idővel JavaScriptben is fog elkészülni.”

- **express:** Ez a szerver alapja, ezzel kezeljük a HTTP kéréseket (pl. GET, POST). Az Express a JavaScript egyik legnépszerűbb webes keretrendszere, amely évek óta, de facto szabványként van jelen. Ez egy rendkívül minimalista keretrendszer, könnyen megtanulható, és nagy rugalmasságot biztosít a webalkalmazások fejlesztésében. Minimalizmusa azt jelenti, hogy szükség esetén harmadik féltől származó könyvtárakat kell hozzáadni, de egy aktív közösséggel rendelkezik, amelynek köszönhetően rengeteg könyvtár érhető el a gyakori problémák megoldására.

```
app.get("/user", (req, res) => {  
  const token = req.headers.authorization;  
  if (!token) return res.status(401).json({ message: "Nincs bejelentkezve" });  
  
  jwt.verify(token, "secret", (err, decoded) => {  
    if (err) return res.status(403).json({ message: "Érvénytelen token" });  
    res.json(decoded.Nev);  
  });  
});
```

1.2.ábra

- **mysql2**: Az adatbázishoz csatlakozik, hogy adatokat tudjunk tárolni és lekérdezni.

```
const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "quickservice",
});

db.connect(err => {
  if (err) throw err;
  console.log("Sikeres adatbázis kapcsolódás");
});
```

1.3.ábra

A **mysql2** modul segítségével az adatbázis kapcsolat olyan adatait, mint: a szerver Ip címe, az adatbázis neve, és az adatbázis hozzáférésehez szükséges felhasználói fiók adatait egy változóban tároljuk, így bármikor könnyedén hozzáférhetünk, amikor az api végpontokban lekérdezéseket hajtunk végre. A **connect** metódussal ellenőrizzük, hogy valóban létrejött-e a megfelelő kapcsolat.

- **bcryptjs**: A jelszavakat titkosítja, hogy ne sima szöveggént tároljuk őket.

```
const hashedPassword = await bcrypt.hash(Jelszo, 10);
```

1.4.ábra

```
const isMatch = await bcrypt.compare(Jelszo, vesarlo.Jelszo);
```

1.5.ábra

A **bcryptjs** regisztrációkor képes a felhasználó által megadott jelszót hash formába átalakítani, így az adatbázisban már a titkosított verzió kerül rögzítésre. A **hash**: egy olyan folyamat vagy függvény eredménye, amely egy bemenetet (pl. szöveget, adatot) egy fix hosszúságú karaktersorozattá alakít. Ezt gyakran használják adatbázisokban, jelszókezelésben vagy adatellenőrzésben, gyors és nehéz visszafejteni. A vásárló autentikációjakor a modul képes visszafejteni az eltárolt hash-elt jelszót és összehasonlítani a bejelentkezéskor megadott jelszóval. Az **isMatch** változónak adjuk át true vagy false értékként az összehasonlítás eredményét.



- **jsonwebtoken:** Tokeneket generál a bejelentkezéshez, hogy a szerver tudja, ki van bejelentkezve.

```
const token = jwt.sign({ Vasarlo_ID: vasarlo.Vasarlo_ID,  
  Nev: vasarlo.Nev ,Email:vasarlo.Email},"secret", { expiresIn: "1h" });
```

### 1.6.ábra

A **jwt.sign** metódus létrehoz egy tokenet ha a felhasználó sikeresen megadta a nevét és jelszavát. A tokenben eltároljuk a vásárló azonosítóját, nevét és email címét. A token 1 óráig érvényes ezután a felhasználónak újra be kell jelentkeznie.

```
jwt.verify(token, "secret", (err, decoded) => {  
  if (err) return res.status(403).json({ message: "Érvénytelen token" });
```

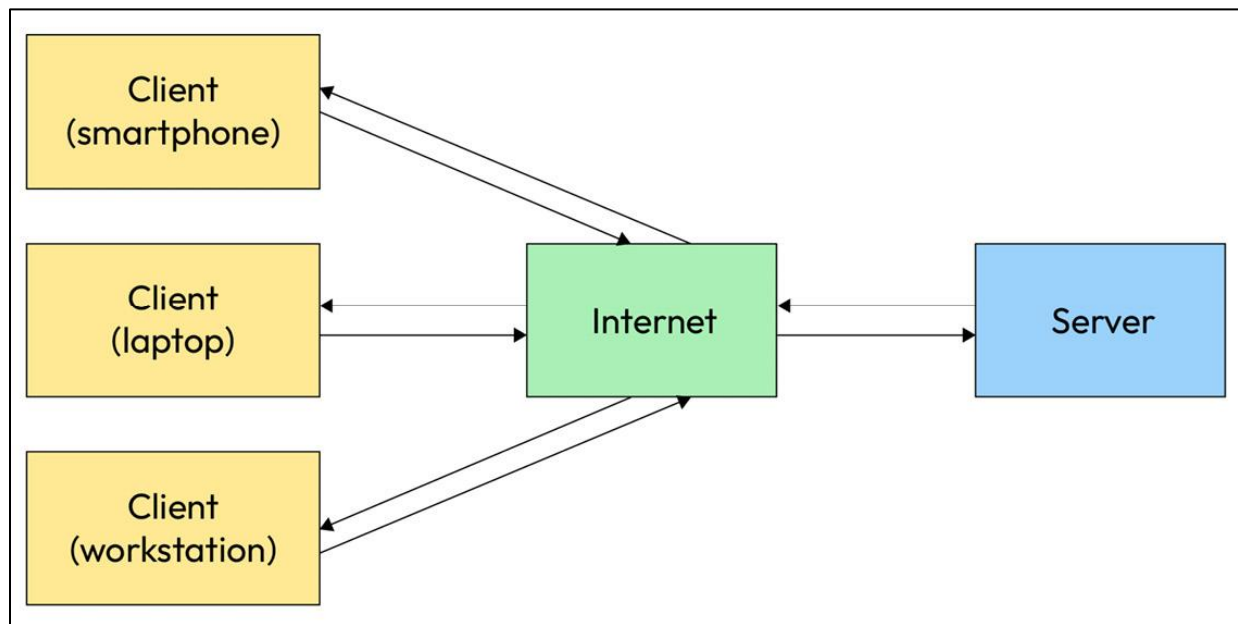
### 1.7.ábra

A **jwt.verify** a token érvényességét határozza meg. Érvénytelenség esetén hibaüzenettel tér vissza és nem engedi, hogy SQL lekérés végbe mehessen. A decoded változóval elérhetjük a tokenben titkosított adatokat.

- **cors:** Lehetővé teszi, hogy a frontend (pl. egy másik domainről) kommunikáljon a szerverrel.

A **szerver és a kliens közötti kommunikáció** az HTTP protokollon keresztül zajlik, ahol a kliens kéréseket küld a szervernek, a szerver pedig válaszol ezekre a kérésekre.

- **Szerver:** A szerver egy olyan számítógép, amelyen az alkalmazás fut, és amely az adatbázissal kapcsolatos lekérdezéseket, valamint sok más feladatot végez. Ezt a szervert gyakran háttérprogramnak is nevezik.
- **Kliens:** A kliens az a szoftver, amelyet a végfelhasználó helyi gépén futtat, például webes alkalmazások esetén. A felhasználó a webböngészőt használja, hogy végrehajtsa a szoftverben található kódokat (HTML, CSS, JS stb.). Ez a tipikus kérés/válasz ciklus.



1.8.ábra

A leggyakoribb **kérési módszerek** a GET, POST, PUT és DELETE: a GET erőforrás lekérésére, a POST erőforrás létrehozására, a PUT erőforrás frissítésére, a DELETE pedig törlésre szolgál.

A **REST** (Representational State Transfer) egy építészeti stílus, amelyet API-k építésére használnak, ahol minden erőforrást egyedi URL azonosít, és a kliens HTTP-módszerekkel végezhet rajta műveleteket.

A **JavaScript**, amelyre a Node.js épül, egy erős és rugalmas programozási nyelv, amit a front-end és back-end fejlesztés mellett mobilalkalmazásokban, asztali számítógépeken, sőt az IoT (Internet of Things) területén is használnak. Több mint 25 éve létezik, alapjait Brendan Eich fektette le 1995-ben a Netscape Communications Corporation-nél, eredetileg Mokka néven, amit később LiveScriptre, majd végül JavaScriptre kereszteltek át.

## b) MySQL

Az adatbázis-kezeléshez **MySQL**-t használtunk, amelyet phpMyAdmin segítségével menedzselünk a XAMPP környezetben, lehetővé téve a rendelések, felhasználók és termékadatok strukturált tárolását és gyors lekérdezését. Az **SQL** (Structured Query Language) egy szabványos lekérdező nyelv relációs adatbázisok kezelésére.

A **relációs adatmodell** táblákon alapul, ahol az adatok sorokban és oszlopokban tárolódnak, és kulcsokkal – például **elsődleges kulcsokkal** (primary keys) és **idegen kulcsokkal** (foreign keys) – kapcsolhatók össze. Ez a struktúra biztosítja az adatok konzisztenciáját és integritását, miközben lehetővé teszi komplex lekérdezések futtatását, például JOIN műveletekkel több tábla adatainak összekapcsolására.

A **MySQL** választása különösen előnyös volt, mivel nyílt forráskódú, széles körben támogatott, és hatékonyan kezeli a nagy adathalmazokat is. A MySQL egy gyors, hatékony és könnyen használható adatbázis-kezelő rendszer, amely minden olyan funkciót kínál, amire egy weboldalnak szüksége lehet az adatok tárolásához és böngészők számára történő megjelenítéséhez.

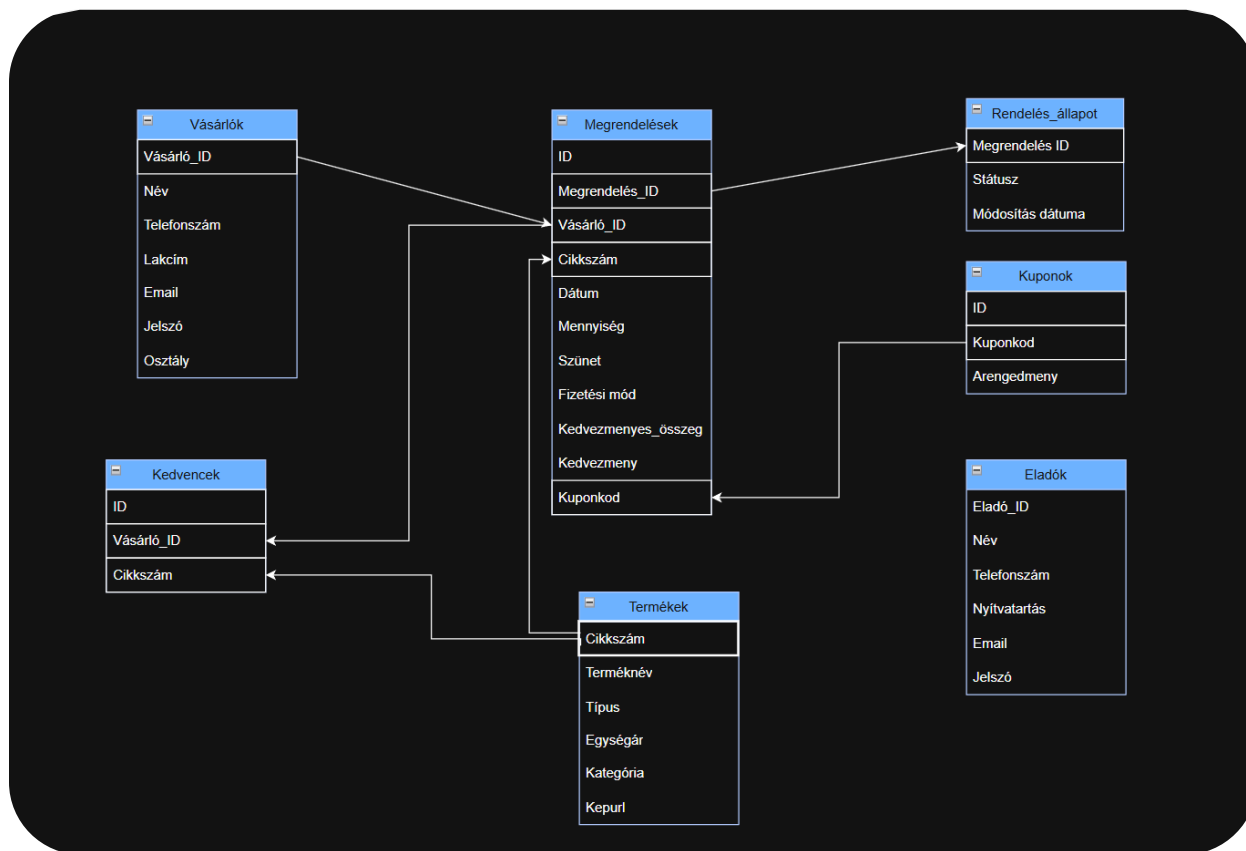
Az 1990-es évek közepén fejlesztették ki, és mára egy kiforrott, megbízható technológiává vált, amely angolszerű parancsokat használ. A **phpMyAdmin** grafikus felülete megkönnyítette az adatbázis adminisztrációját, például új táblák létrehozását, indexek kezelését vagy adatimportálást/exportálást. A **XAMPP** környezet pedig egy integrált fejlesztési platformot biztosított, amelyben az Apache webservert, a MySQL adatbázist és a PHP programozási nyelvet zökkenőmentesen működött együtt.

Ez ideális választás volt egy **helyi tesztkörnyezet** kialakításához is. A MySQL szerver többszálú, így egyszerre több kliens is csatlakozhat hozzá, és minden ügyfél több adatbázist is használhat párhuzamosan. Emellett a MySQL telepítési mérete viszonylag kicsi, különösen más, nagyobb adatbázis-rendszerek hatalmas lemezterületéhez képest, ami további előnyt jelentett a használatában.

A MySQL adatbázis több táblát is tartalmazhat, amelyek mindegyike rekordokat vagy sorokat tárol. Minden sor különböző oszlopokkal vagy mezőkkel rendelkezik, amelyek az adatokat tartalmazzák.

MYSQL PARANCSONK	
CREATE	Létrehoz egy adatbázist, táblát vagy indexet
DELETE	Sor törlése a táblázatból
INSERT	Adatok beszúrása
UPDATE	Meglévő rekord frissítése

1.9.ábra



1.10. ábra

Az adatbázis több táblát tartalmaz, amelyek között kapcsolatok (idegen kulcsok) biztosítják az adatok összekapcsolását.

- A **Vásárlók tábla** például alapvető ügyfeladatokat tárol (név, telefonszám, lakcím, email, jelszó, osztály), és a Megrendelések táblával van összekapcsolva a Vásárló\_ID mezőn keresztül, amely a vásárlókhoz rendeli a rendeléseket.
- A **Megrendelések tábla** a rendelések részleteit pl:(státusz, módosítás dátuma) tartalmazza, amely több más táblával is kapcsolatban áll.
- A **Termékek tábla** a termékek adatait (név, típus, egységár, kategória, kép URL) rögzíti, és a Megrendelések táblával a Cikkszám mezőn keresztül kapcsolódik.
- Az **Eladók tábla** az eladók adatait (név, telefonszám, nyitvatartás, email, jelszó) tárolja Hasonló attribútumokkal rendelkezik a „Vásárló” táblához. Elsődleges kulcsa az „Eladó ID”.
- A **Kedvencek tábla** a vásárlók által kedvelt termékeket rögzíti, összekapcsolva a Vásárlók és a Termékek táblákat.
- **Rendelés állapot:** Feladata, hogy eltárolja a Megrendelések státuszait

### Néhány fontosabb attribútum:

1. **Vásárlók tábla attribútuma:** Vásárló ID: Az tábla egyedi azonosítója (primary key) és idegen kulcsa, amely egyértelműen azonosít minden vásárlót. Szám (Integer).
2. **Rendelés\_állapot attribútuma:** Státusz: A rendelés aktuális állapota (pl. „Elfogadva”, „Teljesítve”, „Átvehető”) (VARCHAR).
3. **Megrendelések tábla attribútuma:** Megrendelés ID: Egyedi azonosító a rendeléshez (primary key).
4. **Termékek tábla attribútuma:** Cikkszám: A termék egyedi azonosítója (primary key) Szám (Integer).

### c) HTML, CSS, Bootstrap

A frontend HTML jelölőnyelv, CSS stílusleírónyelv és Bootstrap keretrendszer kombinációjára épül. A frontend HTML, CSS és Bootstrap kombinációja lehetővé teszi, hogy a weboldalak gyorsan és hatékonyan létrejöjjenek, miközben reszponzív és felhasználóbarát dizájnt biztosítanak.

A **HTML – vagyis a Hypertext Markup Language** – az oldal alapvető struktúráját és tartalmát biztosítja, amely lehetővé teszi a hiperszövegek használatát, vagyis olyan szövegeket, amelyek hiperhivatkozásokat tartalmaznak, így a felhasználók kattintással navigálhatnak az oldalakon belül vagy más oldalakra.

Jelenleg az 5-ös verzió, az úgynevezett **HTML5** az aktuális, amely 2008-ban jelent meg, 2014-ben frissült jelentősen, és 2023-ban is folyamatos fejlesztés alatt áll, új funkciókkal bővülve. Egy érvényes HTML-dokumentum alapvető elemei közé tartozik a dokumentumtípus-deklaráció, a html, head és body elemek, valamint a head-ben egy title elem.

A **CSS – vagyis a Cascading Style Sheets** – segítségével a dizájn és a vizuális megjelenés finomhangolható, például színek, tipográfia, elrendezés és animációk révén. Ez a nyelv a HTML tartalom formázására szolgál, és három módon alkalmazható: külső fájlként, a HTML-dokumentumba ágyazva vagy egyedi címkékben. A külső CSS-fájl használata a legelterjedtebb, mert egyszerre több oldal stílusát szabályozza, így egyszerűbb a módosítás.

A **Bootstrap** egy olyan népszerű CSS keretrendszer, amely előre definiált stílusokat és komponenseket biztosít, ezzel gyorsítva a fejlesztési folyamatot és biztosítva, hogy az oldal minden eszközön jól nézzen ki, legyen szó asztali számítógépről, táblagépről vagy mobiltelefonról. A Bootstrapet 2011-ben Mark Otto és Jacob Thornton, a Twitter fejlesztői hozták létre, hogy konzisztenciát és karbantarthatóságot biztosítsanak a kódban.

**Reszponzív grid rendszere** 12 oszlopra osztja a képernyőt, amelyek dinamikusan alkalmazkodnak a képernyőmérethez, és négy osztályelőtagot (col, col-sm, col-md, col-lg) kínál a különböző eszközméretek kezelésére. A beépített UI komponensei (például gombok, navigációs sávok és űrlapok) leegyszerűsítik a weboldalak kialakítását, miközben biztosítják a zökkenőmentes felhasználói élményt.

```
<div class="col-lg-8 col-md-10 col-12">
```

1.11. ábra

#### d) Angular

A kliensoldali logikát **Angular** keretrendszerrel valósítottuk meg, amely dinamikus, SPA (Single Page Application), ami azt jelenti, hogy az oldal nem töltődik újra, hanem csak a szükséges részek frissülnek, így gyorsabb. Egyoldalas alkalmazásként kezeli a valós idejű interakciókat, például a rendelési státuszok frissítését. A korábban elkészített oldalainkat kisebb részekre, vagyis komponensekre bontja, például navigációs sáv vagy kosár komponens. Ezek megkönnyítik a kód átláthatóságát és könnyebbé teszik a HTML és JavaScript kezelését. Egy komponens egy olyan osztály, amelynek tulajdonságai és metódusai vannak, amelyeket a View-ban használnak fel, így a nézet interakcióba léphet a komponensekkel.

Ezen kívül az Angular **kétirányú adatbindingot** alkalmaz, ami azt jelenti, hogy az alkalmazásunkban történt változtatások automatikusan frissítik a felhasználói felületet, és fordítva, így az interakciók dinamikusak és zökkenőmentesek. Az adatkötés lehet egyirányú (One-way Data Binding), amely vagy a komponensből a DOM-ba, vagy a DOM-ból a komponensbe irányul, illetve kétirányú (Two-way Data Binding), amely mindkét irányú kommunikációt lehetővé tesz, például az [(ngModel)]="nev" használatával. Emellett támogatja a rezponzív dizájnt, amelynek segítségével az alkalmazás bármilyen képernyőmérethez alkalmazkodik.

Az Angularral készített alkalmazások bármilyen eszközre célozhatók, például mobiltelefonokra, táblagépekre és asztali számítógépekre.

A fejlesztés során fontos szerepet kap a **modulok és szolgáltatások** szervezése. Az Angular moduláris felépítése lehetővé teszi, hogy az alkalmazásunk különböző részeit könnyedén bővítsük és karbantartsuk. Az Angular az AngularJS utódja, amelyet teljesen újraírtak, javított függőséginjektálással, dinamikus betöltéssel és egyszerűbb útvonalvezetéssel. Architektúrája sablonokból, metaadatokból és direktívákból is áll. Ez a struktúra nemcsak a kód átláthatóságát növeli, hanem segít abban is, hogy az alkalmazás skálázható és fenntartható maradjon hosszú távon.

A **direktíva** utasításokat vagy iránymutatásokat jelent egy sablon megjelenítéséhez. A strukturális direktívák manipulálják a DOM elemeket, és azok struktúráját módosítják DOM elemek hozzáadásával, eltávolításával vagy cseréjével. Például:

```
<ul>
<li *ngFor="let termek of termekek">
  {{ termek.Cikkszam }}
</li>
</ul>
```

**Néhány gyakran használt strukturális direktíva a következő:**

- **ngFor**: Ismétlődő direktíva, amelyet általában egy elemlista ciklikus megjelenítésére használnak.
- **ngIf**: DOM elemeket mutat vagy rejt el egy kifejezés kiértékelésének eredményétől függően; az eredmény igaz vagy hamis lehet.

A **szolgáltatások** (Services) felhasználó által definiált osztályok, amelyeket problémák megoldására használnak. Az Angular azt javasolja, hogy a komponensekben csak sablonra specifikus kódok legyenek. Egy komponens feladata az Angular alkalmazásban a felhasználói felület/élmény (UI/UX) gazdagítása, míg az üzleti logikát a Service-ek kezelik.

A **TypeScript** a JavaScript egy kiterjesztett változata, egy nyílt forráskódú nyelv, amelyet a Microsoft alkotott meg, és amelyet az Angular is használ. Az „any” nevű adattípus rugalmas,

bármilyen értéket képes tárolni. A TypeScript fordítási hibát jelez, ha egy karakterláncot próbálunk egy egész szám típusú változóhoz rendelni. Ahhoz, hogy elkerüljük a típusellenőrzést, az „any” típust használtuk. Példa:

```
public Termek: any[]=[];
```

Ebben az esetben egy „any” típusú tömböt hoztunk létre, amely számokat, szövegeket és logikai értékeket is tartalmazhat.

Az Angularban az **ngClass direktíva** lehetővé teszi CSS osztályok hozzáadását vagy eltávolítását egy kifejezés alapján, de az osztálybekötés egy másik praktikus megoldás ugyanezen célra. Példa:

```
[ngClass]="{'btn-danger': isKedvenc(termek), 'btn-outline-danger': !isKedvenc(termek)}"
```

Az Angular alkalmazások alapegységei a **komponensek**, és minden alkalmazáshoz szükség van legalább egy gyökérkomponensre, hogy működőképes legyen. Minden komponensnek megvan a maga életciklusa: létrejön, figyeli a változásokat, majd megszűnik. Az Angular hook metódusokat kínál, amelyekkel a fejlesztők beavatkozhatnak ebbe az életciklusba. Ezeket a TypeScript interfészek biztosítják, amelyeket a komponens osztályok opcionálisan megvalósíthatnak:

```
ngOnInit(): void {  
  this.loadtermek();  
  this.loadKedvencek();  
}
```

**ngOnInit:** Ez a metódus az első ngOnChanges után, az adatbekötött és bemeneti tulajdonságok inicializálása után fut le egyszer, és a direktívák életciklusához is kapcsolódik.

A **navigáció** alapvető szerepet tölt be minden weboldalon vagy alkalmazásban. Az útvonalak meghatározása mellett segít a felhasználóknak az oldalak közötti mozgásban, a funkciók



használatában, és támogatja a keresőoptimalizálást is. Az útvonalak definíciói a következő elemeket tartalmazhatják:

- **path:** Az a webcím, amely a böngésző címsávjában megjelenik.
- **component:** Az a komponens, amely a nézetet és a logikát biztosítja.
- **redirectTo (opcionális):** Az az URL, ahová a felhasználót átirányítjuk.
- **pathMatch (opcionális):** Meghatározza, hogy az útvonal hogyan illeszkedjen az URL-hez; értéke lehet full vagy prefix.

Például a vásárló rendeléseihez tartozó komponens útvonala:

```
{path: 'rendeleseim', component: RendeleseimComponent},
```

A **router-outlet** direktíva jelzi, hogy hol jelenjenek meg a betöltött nézetek. A (pipes) pedig a sablonokban lévő adatok formázására szolgálnak a megjelenítés előtt. Például:

```
{{ Rendesadatok[0].datum | date: 'yyyy-MM-dd HH:mm' }}
```

A pipes a „|” szimbólummal és a pipe névvel definiálhatók.

## e) React Native

A **React Native** egy nyílt forráskódú keretrendszer, amelyet natív mobilalkalmazások készítésére hoztak létre, és a Facebook egyik csapata fejlesztett. Az egyik legjobb dolog a React Native-ben, hogy olyan ismert webes technológiákat alkalmaz, mint a JavaScript, a CSS és az HTML, miközben az elkészült alkalmazás teljes mértékben natív élményt nyújt.

Az **Expo** egy nyílt forráskódú eszközkészlet, amelyet a React Native köré terveztek, hogy megkönnyítse az iOS és Android alkalmazások fejlesztését. Az Expo segítségével villámgyorsan elkezdhetjük a mobilalkalmazásunk fejlesztését. A kód írása mellett letöltöttük az Expo alkalmazást a Google Play Áruházból Androidra. Ez az app lehetővé tette, hogy a fejlesztés során teszteljük és kipróbáljuk az alkalmazást. Az Expo Developer Tools hozzáférést biztosít a Metro bundler naplóihoz, valamint gyorsbillentyűket kínál különböző lehetőségekhez, például az alkalmazás futtatásához iOS vagy Android szimulátorokon. Emellett létrehoz egy QR-kódot is, amit az Expo Go alkalmazással beolvastva azonnal tesztelhetjük a projektet.

A React Native-ben az alapvető elemeket a **react-native könyvtárból** hívjuk be. Más keresztplatformos megoldásokkal ellentétben a React Native nem webes nézetekre támaszkodik a JavaScript kód megjelenítéséhez az eszközön, hanem a JavaScriptben megírt felületet natív kezelőfelületi elemekké alakítja. A működése két szálon zajlik: a JavaScript szálon fut a kód, míg a natív szál (vagy UI szál) kezeli az eszközinterakciókat, például a felhasználói bevitelt és a képernyők megjelenítését.

Mióta 2015-ben **nyílt forráskódúvá** tették, egy óriási és folyamatosan bővülő közösség jött létre, amely rengeteg kiegészítő csomagot készít a legkülönbözőbb igényekre és feladatokra. Ez a React Native egyik legnagyobb erőssége más hasonló keresztplatformos technológiákkal szemben.

## f) Jest

A **Jest** egy széles körben elismert JavaScript tesztelési keretrendszer, amelyet eredetileg a Meta (korábban Facebook) hozott létre, elsősorban React alkalmazások tesztelésére, de bármilyen JavaScript projekthez kiválóan alkalmazható. Az évek során gazdag funkcionálitása és megbízható teljesítménye miatt a JavaScript-kód tesztelésének egyik legnépszerűbb eszközévé vált. Egyszerű kezelhetősége és hatékonysága révén a fejlesztők körében is nagy népszerűsége lett, ezért mi is úgy gondoltuk, hogy ideális és hatékony választás lesz a projektünk tesztelési igényeire.

A Node-ban szimulált böngészőkörnyezet használatának **előnyei** jelentősek: a tesztek gyorsabban futnak, a kódlefedettség egyszerűen mérhető, ráadásul ugyanazok a tesztek konzisztens eredményeket hoznak, függetlenül attól, hogy egy fejlesztői gépen vagy egy folyamatos integrációs szerveren futtatjuk őket. Ez a rugalmasság és megbízhatóság tovább növeli a Jest értékét a projektünk szempontjából.

**Ez a technológiai kombináció gyors, megbízható és felhasználóbarát rendszert eredményezett.**

# Fejlesztési folyamat

## 1. Idővonal

A felvetett problémára megoldást kínáló program elkészítésére hét hónap állt rendelkezésünkre. Miután megszületett az ötlet és a projekt neve, a munka első lépéseként alapos tervezésbe kezdtünk. Elgondolkodtunk, milyen irányba induljunk el. Két lehetőség merült fel: natív asztali alkalmazás vagy webalkalmazás. A natív alkalmazás hátránya az volt, hogy a .NET keretrendszer megkötései miatt kizárólag Windows környezetben működött volna. Ezzel szemben a webalkalmazás szinte minden platformon zökkenőmentesen használható. Ezért az utóbbi mellett döntöttünk.

A következő héten meghatároztuk azokat az alapvető funkciókat, amelyeket mindenképp szerettünk volna beépíteni a programba. Ezek alapján elkészítettük az első látványtervet, amelyek az eladói és vásárlói felhasználói felületek elképzelt kinézetét mutatták be. A projekt korai szakaszában külön dolgoztunk, hogy minél több ötletet gyűjtsünk össze. Mindketten előálltunk saját javaslatainkkal, amelyeket később összevetettünk, és a legjobb megoldásokat kiválasztottuk az éles verzióhoz.

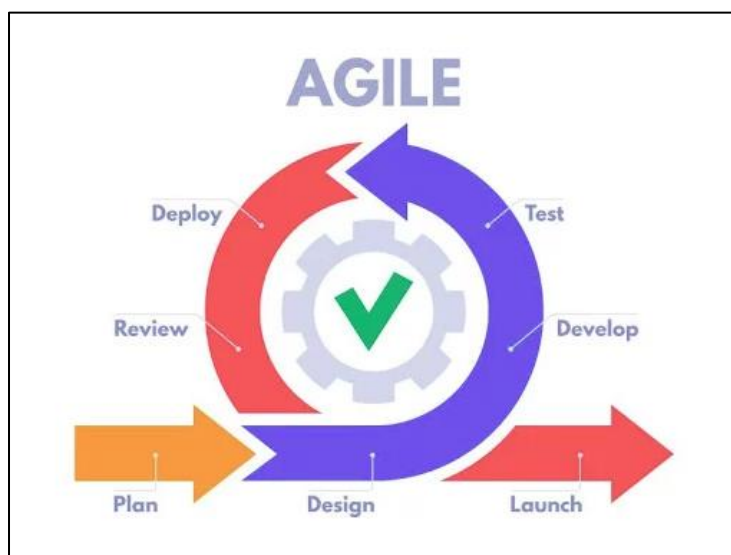
A terveink mintája alapján külön-külön létrehoztunk egy statikus weboldalt. Ezt követően összefűztük őket, megtartva a legjobban sikerült elemeket. Következő lépésként a backend tervezésére koncentráltunk. Elkészítettük az ER-diagramot, átgondoltuk az egyedek közötti kapcsolatokat, és létrehoztuk az adatbázis sémáját néhány tesztadattal. Fiktív lekérdezésekkel ellenőriztük, hogy a táblák közötti átjárás megfelelően működik-e, és minden adat elérhető.

Ezután adatgyűjtésbe kezdtünk. Termékeket válogattunk össze a hozzájuk tartozó fotókkal. A képeket az Imgur szolgáltatásra töltöttük fel, az adatbázisban pedig a linkjeiket tároltuk. Miután befejeztük ezt a lépést, és ellenőriztük a táblák közötti kapcsolatokat, létrehoztuk az éles adatbázist a valós adatokkal, amelyeket az oldalon megjelenítünk. További tesztekkel megbizonyosodtunk róla, hogy minden az elvárásoknak megfelelően működik.

Ezután a frontend és a backend összekapcsolása következett, majd a tervezett funkciók megvalósítása. Ez a folyamat a rendelkezésre álló idő jelentős részét igénybe vette. Ugyanakkor ennek köszönhetően fedeztünk fel olyan hiányosságokat, mint a kuponkezelés, az email-visszajelzés vagy a termékek kedvencnek jelölése. Ezeket hasznos kiegészítéseknek találtuk, így

beépítettük őket a projektbe. Az elkészült munkát többször teszteltük és ellenőriztük. Ezt követően elkészítettük a mobilalkalmazást is, amit újabb ellenőrzések követtek.

A csapatmunkánk során az **agilis módszertatok** felhasználásával dolgoztunk. Ez egy olyan munkaszervezési és gyakorlati megközelítés, amely az agilis alapelvekre építve, iteratív módon valósítja meg a szoftverfejlesztést vagy más projekteket. Az iterációk során a fejlesztés és a tesztelés párhuzamosan zajlik, miközben állandó kapcsolat áll fenn a termék felhasználóival. Ez lehetővé teszi a gyors visszajelzést és a változó ügyféligények felismerését. Minden iteráció végén a fejlesztés egy működőképes részterméket, vagyis egy használható funkcionalitást eredményez.



1.12.ábra

Az egymás közti kommunikációtartásra és a megbeszélésekre használtuk fel a **Google Meet**-et (valós idejű megbeszélésekre alkalmas platform), ahol megoszthattuk egymás képernyőjét is, ami megkönnyítette a projekttel való haladásunkat. Emellett a legfontosabb csoportmunka eszközünknek a **GitHub** nevezhető. Ez egy olyan webalapú platform, ami lehetővé teszi a fejlesztők számára, hogy kódot tároljanak, kezeljenek és megosszanak másokkal. Alapja a Git verziókezelő rendszer, ami segít nyomon követni a kódban történt változtatásokat, és lehetővé teszi több fejlesztő együttműködését egy projekten.

A **GitHub Desktop** pedig egy ingyenes, grafikus felületű alkalmazás, amely leegyszerűsíti a GitHubon tárolt projektek kezelését. Segít klónozni tárolókat, követni változtatásokat, commitokat készíteni és szinkronizálni a kódot, parancssor használata nélkül.

## 2. Kihívások és megoldások

Az alapos és megfontolt tervezés ellenére a munka során számos akadályba ütköztünk, melyen sikeresen túljutottunk. A legnagyobb kihívást a frontend és a backend összekapcsolása jelentette, hiszen az addig elkészült statikus adatokat megjelenítő angular oldalakat dinamikussá kellett átalakítanunk a már korábban taglalt direktívák segítségével.

Ez azt jelentette, hogy az oldalunk képes bármilyen beérkező adat megfelelő megjelenítésére és az adatbázisban történt változások esetén mindig frissíteni azt, amit a felhasználó lát.

Sok gondot okozott emellett a frontend és backend közötti kommunikáció is, hiszen rengetegszer futottunk abba a hibába, hogy a frontend egyáltalán nem vagy hiányos információkat kap a Node szerverünktől. A legapróbb hibák is ahhoz vezettek, hogy nem jött létre a megfelelő kapcsolat ezért nagyon ügyelnünk kellett.

Felkellett készítenünk programunkat olyan esetekre is például, hogy mi történjen, ha a felhasználó nincs bejelentkezve ilyen esetben elkellet érjünk azt, hogy ne tudjon rendelni vagy például ne férjen hozzá a rendeléseihez és még számos felhasználói interakciót ne végezhesen, hiszen ezen funkciók mind a felhasználó autentikációjához kötöttek.

## Tesztelés

### 1. Tesztelési módszerek

A teszteléshez a **Jest keretrendszert** használtuk. A tesztek **mockoltuk**, hogy ne legyen szükség valós adatbáziskapcsolatra és a tesztek izoláltan tudjanak működni így az adatbázisban nem keletkezik változás.

Használtuk a **Supertest** nevű modult. A SuperTest egy népszerű Node.js könyvtár, amelyet HTTP kérések tesztelésére használnak, különösen Express.js alkalmazások esetében.

Lehetővé teszi, hogy **egyszerűen és olvashatóan írassunk** integrációs tesztek a **REST API-khoz** vagy más HTTP-alapú szolgáltatásokhoz.

Segítségével a válaszok státuszkódját, fejléceit, tartalmát stb. könnyen ellenőrizhetjük. Támogatja a Promise-okat és az async/await szintaxist is.

A **mock** egy szoftverfejlesztési és tesztelési koncepció, amelynek lényege, hogy egy objektumot, függvényt vagy modult helyettesítenek egy kontrollált, előre meghatározott viselkedésű változattal. Ezt általában **egységtesztelés** (unit testing) során alkalmazzák, hogy izolálják a tesztelt kódot a külső függőségektől, például adatbázisoktól, API-któl vagy más bonyolult rendszerektől.

A **describe**: A Jest egy csoportosító függvénye, amely összefogja a kapcsolódó tesztekét. Az it: Egy konkrét tesztet definiál. A szöveg leírja, mit várunk el pl: "törölnie kellene a kupont érvényes tokennel".

Az **expect eljárás** a Jest tesztkeretrendszer egyik alapvető része, amelyet arra használnak, hogy ellenőrizzük egy teszt során a várt eredményeket. Magyarul gyakran "**elvárás**"-nak vagy "**állítás**"-nak fordítják, mert azt definiálja, hogy mit várunk el egy adott kódrészlettől vagy annak kimenetétől.

Az **expect** egy függvény, amely egy **értéket vesz át**, és különböző "**matcher**" (illesztő) metódusokkal kombinálva lehetővé teszi, hogy pontosan megvizsgáljuk, megfelel-e az érték az elvárásainknak.

## 2. Eredmények

```
(node:20700) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a u
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS src/app/kosar/kosar.component.spec.ts
  KosarComponent
    ✓ Létre kellene jönnie (51 ms)
    ngOnInit
      ✓ Be kellene töltenie a kosár elemeit és a felhasználói profilt (3 ms)
    csokkentés
      ✓ Meg kellene hívnia a decreaseQuantity-t a kosarService-en (2 ms)
    noveles
      ✓ Meg kellene hívnia az increaseQuantity-t a kosarService-en (2 ms)
    termektorles
      ✓ Meg kellene hívnia a removeItem-et a kosarService-en (1 ms)
    getteljesar
      ✓ Vissza kellene adnia a teljes árat a kosarService-től (2 ms)
    getnettoar
      ✓ Vissza kellene adnia a nettó árat a kosarService-től (1 ms)
    getado
      ✓ Vissza kellene adnia az ÁFA összegét a kosarService-től (2 ms)

(node:18780) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a u
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS src/app/rendelesreszletes/rendelesreszletes.component.spec.ts
  RendelesreszletesComponent
    ✓ Létre kellene jönnie a komponensnek (40 ms)
    getteljesosszeg
      ✓ Le kellene kérnie a rendelés részleteit és ki kell számítani a nettó és adóösszegeket
      ✓ Null-ra kellene állítani a nettó és adóösszegeket, ha az Osszeg 0 vagy undefined (2 ms)
      ✓ Kezelnie kellene az undefined Osszeg-et a válaszban (2 ms)

(node:4588) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a u
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS src/app/SERVICES/vasarlo-auth.service.spec.ts
  VasarloAuthService
    register
      ✓ POST kérést kellene küldenie a regisztrációhoz. (18 ms)
    login
      ✓ POST kérést kellene küldenie a bejelentkezéshez. (2 ms)
    logout
      ✓ El kellene távolítani a token-t a localStorage-ból (2 ms)
    getvasarlo
      ✓ Null-t kellene visszaadnia, ha nincs jelen token (1 ms)
      ✓ GET kérést kellene küldenie a felhasználó lekérdezéséhez, ha a token jelen van (5 ms)

Test Suites: 3 passed, 3 total
Tests: 17 passed, 17 total
Snapshots: 0 total
Time: 3.245 s
```

1.13.ábra



```

POST /sendorder endpoint
  Autentikáció ellenőrzése
    ✓ 401-es hibakódot kell visszakapnia, ha nincs token megadva (21 ms)
    ✓ 403-as hibakódot kell visszakapnia, ha érvénytelen token van megadva (2 ms)
  Sikeres rendelés rögzítése
    ✓ 200-as kódot kell visszakapnia, ha érvényes token és adatok vannak megadva (3 ms)
  Hibakezelés
    ✓ 500-as hibakódot kell visszakapnia, ha az adatbázis-lekérdezés a rendelés rögzítésekor sikertelen (2 ms)
    ✓ 500-as hibakódot kell visszakapnia, ha az adatbázis-lekérdezés az állapot rögzítésekor sikertelen (2 ms)

console.log
  MySQL Connected...

    at log (database.js:12:11)

PASS ./kupontorles.test.js
DELETE /admin/kuponok/:kuponid
  ✓ törölnie kellene a kupont érvényes tokennel (15 ms)
  ✓ 401-es hibakódot kellene visszaadnia, ha nincs token megadva (11 ms)
  ✓ 403-as hibakódot kellene visszaadnia, ha a token érvénytelen (2 ms)
  ✓ 500-as hibakódot kellene visszaadnia, ha az adatbázis-lekérdezés sikertelen (2 ms)

console.log
  MySQL Connected...

    at log (database.js:12:11)

PASS ./termek.test.js
GET /products
  ✓ termékeket kellene visszakapnia, ha az adatbázis-lekérdezés sikeres (8 ms)
  ✓ 500-as hibakódot kellene visszaadnia, ha az adatbázis-lekérdezés sikertelen (3 ms)

console.log
  TesztUser

    at log (server.js:51:13)

console.log
  RosszUser

    at log (server.js:51:13)

console.log
  TesztUser

    at log (server.js:51:13)

PASS ./login.test.js
POST /login
  ✓ sikeres bejelentkezésnek kellene lennie helyes adatokkal (12 ms)
  ✓ 400-as hibakóddal kellene visszatérnie ha a felhasználónév helytelen (3 ms)
  ✓ 400-as hibakódot kellene visszakapnia, ha a jelszó helytelen (2 ms)

console.log
  MySQL Connected...

    at log (database.js:12:11)

PASS ./user.test.js
GET /user
  ✓ 401 hibakóddal kellene visszatérjen ha nem adtak meg token (5 ms)
  ✓ 403 hibakóddal kellene visszatérnie a a token érvénytelen (2 ms)
  ✓ a felhasználóval kellene visszatérnie ha a token érvényes (1 ms)

Test Suites: 5 passed, 5 total
Tests: 17 passed, 17 total
Snapshots: 0 total
Time: 1.068 s
Ran all test suites.

```

## 1.14.ábra



# Jövőkép

A projektünkkel kapcsolatban természetesen a jövőképre nézve is vannak módosítási ötleteink, amelyeket szívesen megvalósítanánk, hogy egy sokoldalúbb és kedveltebb webalkalmazást nyújthassunk mind a diákok és az eladók számára. Ilyen ötletek közé tartozik a:

- **Hűségprogram bevezetése:** A vásárlók pontokat gyűjthetnének minden rendelés után, amelyeket később kedvezményekre vagy ingyenes termékekre válthatnak be.
- **Közösségi funkciók:** „Top kedvencek” lista, ahol látszik, mit rendelnek a legtöbben.
- **Automatizált készletkezelés és előrejelzés az eladóknak:** A rendszer jelezhetné az eladóknak, ha egy termék kifogyóban van.
- **Fizetési módok bővítése:** Apple Pay / Google Pay integráció.

## A Quick Serve vásárlói oldala:



Üdvözljük a QuickServe-nél!

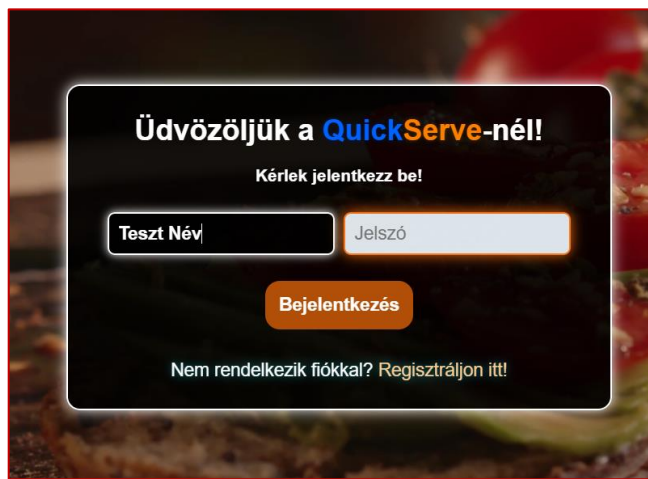
Kérlek regisztrálj!

Felhasználónév	Email cím
Jelszó	Telefonszám
Lakcím	9

Regisztrálás

Már van fiókod? Jelentkezz be itt!

1.15.ábra



Üdvözljük a QuickServe-nél!

Kérlek jelentkezz be!

Teszt Név	Jelszó
-----------	--------

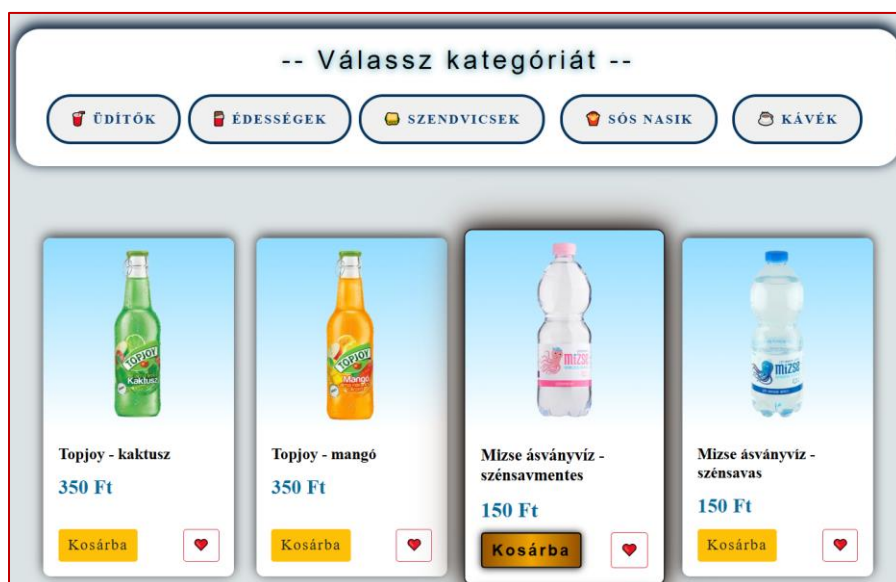
Bejelentkezés

Nem rendelkezik fiókkal? Regisztráljon itt!

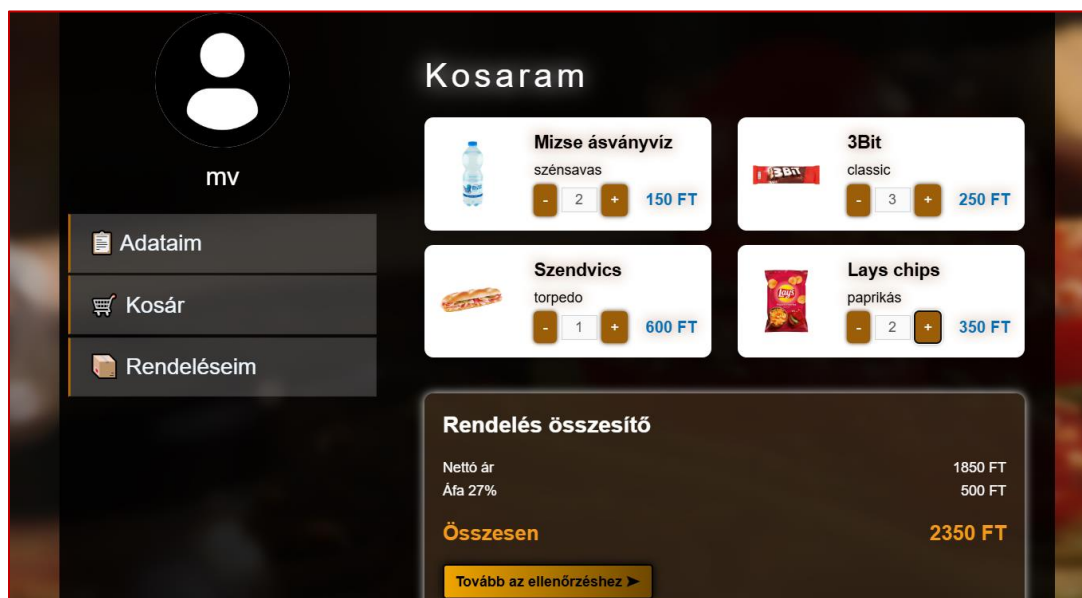
1.16.ábra



1.17.ábra



1.18.ábra



1.19.ábra

Melyik szünetbe kéred?

Kérek válassz egy szünetet


Kuponkód (opcionális)

Add meg a kuponkódot

Ellenőrzés

FIZETÉSI MÓD:


☒ Készpénz
☐ Bankkártya



Mizse ásványvíz - szénsavas

Mennyiség: 2

150 FT



3Bit - classic

Mennyiség: 3

250 FT

Rendelés összesítő

Nettó ár

827 FT

Áfa 27%

223 FT

Összesen

1050 FT

1.20.ábra



Rendelés visszaigazolás

Kedves mvl

Köszönjük, hogy a QuickServe-t választottad! Az alábbiakban találod a rendelésed részleteit:

Rendelésszám: #20250323148

Termék: Mizse ásványvíz (szénsavas), 3Bit (classic)

Mennyiség: 5

Összeg: 1050 Ft

Rendelés dátuma: 2025-03-23 20:55

Státusz: Rendelés elküldve

Fizetési mód: Készpénz

Átvétel várható időpontja: 4. szünet

Ha bármilyen kérdésed van, keress minket bátran!

Üdvözlettel,

A QuickServe csapata

1.21.ábra



Rendelési állapot

Kedves Vásárlónk!

A(z) 20250323024 rendelésed státusza megváltozott.

Jelenlegi állapota: Átvehető

Köszönjük, hogy minket választottál!

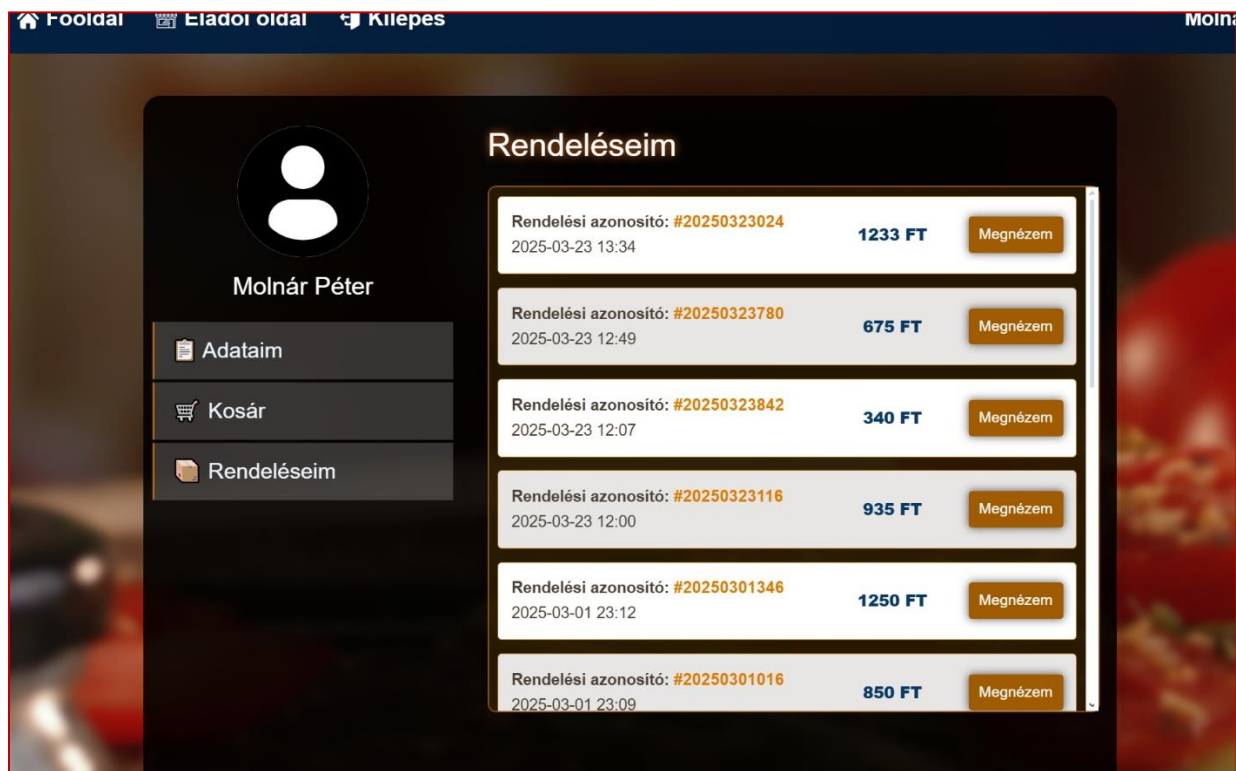
Rendelés Megtekintése

Üdvözlettel: QuickServe csapata

Ha kérdésed van, keress minket bátran!

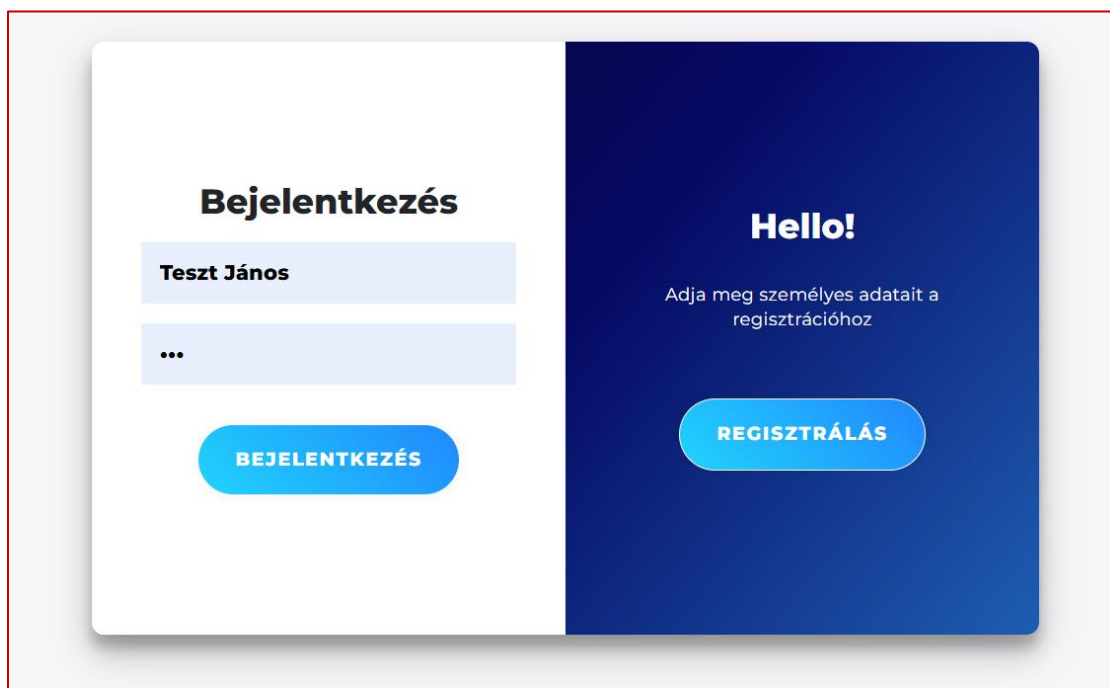
1.22.ábra

27

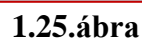


1.23.ábra

A **Quick Serve** eladói oldala:



1.24.ábra



## Irodalomjegyzék

1. Akshat Paul, Abhishek Nalwaya. *React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android applications*. O'REILLY, 2019. E-BOOK.
2. Alexander Benedikt Kutting. *Professional React Native*. O'REILLY, 2022. E-BOOK.
3. DuBois, Paul. *MySQL*. O'REILLY, 2008. E-BOOK.
4. Gascón, Ulises. *Node.js for Beginners*. O'REILLY, 2024. E-BOOK.
5. Hart-Davis, Guy. *Teach Yourself VISUALLY HTML and CSS, 2nd Edition*. O'REILLY, 2023. E-BOOK.
6. Henick, Ben. *HTML & CSS: The Good Parts*. O'REILLY, 2010. E-BOOK.
7. Mathieu Nayrolles, Rajesh Gunasundaram, Sridhar Rao. *Expert Angular*. O'REILLY, 2017. E-BOOK.
8. McFedries, Paul. *HTML & CSS Essentials For Dummies*. O'REILLY, 2024. E-BOOK.
9. Mikhail Sakhniuk, Adam Boduch. *React and React Native – Fifth Edition*. O'REILLY, 2024. E-BOOK.
10. Nixon, Robin. *Learning PHP, MySQL & JavaScript, 7th Edition*. O'REILLY, 2025. E-BOOK.
11. Rahman, Syed Fazle. *Jump Start Bootstrap*. O'REILLY, 2014. E-BOOK.
12. Trevor Burnham. *Test-Driven React, 2nd Edition*. O'REILLY, 2024. E-BOOK.

## Ábrajegyzék

- 1.1. ábra: az Ulises Gascón által írt Node.js for Beginners című könyvből származik.
- 1.2. ábra: A felhasználó nevének lekérése.
- 1.3. ábra: Az adatbázis kapcsolat létrehozása.
- 1.4. ábra: A jelszó titkosításáért felelős kód.
- 1.5. ábra: Kód az isMatch változóról, ezzel adjuk át true vagy false értéként az összehasonlítás eredményét.
- 1.6. ábra: jsonwebtoken használva tokenek generálása a bejelentkezéshez, hogy a szerver tudja, hogy ki van bejelentkezve.
- 1.7. ábra: A jwt.verify a token érvényességét határozza meg.
- 1.8. ábra: A szerver, az internet és a több kliens közötti kapcsolatot mutatja. a Ulises Gascón által írt Node.js for Beginners című könyvből származik.
- 1.9. ábra: a legismertebb MySQL parancsok jelentései.

- 1.10. ábra: Az adatbázis szerkezete.
- 1.11. ábra: A Bootstrap reszpónzív grid rendszere egy kód példájában.
- 1.12. ábra: az agilis módszertan, <https://promanconsulting.hu/wp-content/uploads/2022/03/agilis-modszertanok-optimized.jpg>.
- 1.13. ábra: Node tesztek.
- 1.14. ábra: Angular tesztek.
- 1.15. ábra: A vásárlói felület regisztrációs oldala.
- 1.16. ábra: A vásárlói felület bejelentkezős oldala.
- 1.17. ábra: A vásárlói felület főoldalának kezdőképe.
- 1.18. ábra: A vásárlói felület főoldalának tartalmi részlete.
- 1.19. ábra: A vásárlói felület kosár oldala.
- 1.20. ábra: A vásárlói felület rendelés összesítő oldalából egy részlet.
- 1.21. ábra: A leadott rendelés után a visszajelző email.
- 1.22. ábra: A módosult rendelési állapotról jövő email.
- 1.23. ábra: A vásárlói felületen a leadott rendelések megtekintésének az oldala.
- 1.24. ábra: Az eladói felület bejelentkezésre szolgáló oldala.
- 1.25. ábra: Az eladói felület főoldala.
- 1.26. ábra: Az eladói felület rendelés megtekintésére és státusz módosítására alkalmas oldala.