Baranya Megyei SzC Radnóti Miklós Közgazdasági Technikum

Szakma megnevezése: Szoftverfejlesztő és -tesztelő

A szakma azonosító száma: 5 0613 12 03

Vizsgaremek

Festményvilág

Témavezető:

Tóth Péter Gergő

Készítették:

Harsányi Zsófia

Orbán Tivadar

Pécs

2024

Tartalomjegyzék

Bevezetés

Az alábbi dokumentációban a Festményvilág munkanévre hallgató projektünket szeretnénk bemutatni. Záródolgozatunk témájául egy online piacfelület fejlesztését választottuk a társammal. A Festményvilág egy piactér, mely összehozza a műkedvelőket és a művészeket. A művészek feltölthetik alkotásaikat, az érdeklődők pedig többféle technikával készült festményekből és rajzokból válogathatnak, valamint egyedi megrendelés leadására is lehetőségük van fotó alapján. Az alkotások téma szerint is kategóriákra vannak osztva, hogy minél többféle keresési lehetőség legyen.

Témaválasztásunk hátterében az állt, hogy tapasztalatot szerezzünk a webfejlesztésben és az online piacterek működésében. Úgy gondoltuk, hogy érdemes egy szűkebb termékcsoportot célba venni, mert a pályakezdő piacterek gyakran specializációval tudnak csak sikeresek lenni, ahol még kisebb a piaci dominancia és ezáltal könnyebben elérhetnek egy adott célcsoportot.

A fejlesztés során a frontend és backend részek elkülönített munkafolyamatokban készültek, a munkamegosztást rugalmasan kezeltük, mindketten kivettük a részünket mindkét rész fejlesztéséből.

Az alábbi dokumentációban részletesen bemutatjuk az alkalmazásunk fejlesztési folyamatát, a felhasznált technológiákat, valamint az elkészült rendszer működését és funkcionalitását.

Felhasználói dokumentáció

2. 1. A program célja, lényegesebb funkciói

A Festményvilág egy online piactér, ahol a felhasználók bejelentkezést követően meghirdethetik alkotásaikat, a látogatók pedig böngészhetnek a feltöltött termékek között. A hirdetések között többféle szempont alapján lehet szűrni (háromféle méret, hétféle technika és tizenhét féle téma, például csendélet, tájkép, enteriőr, portré), valamint a megjelenítés sorrendjét is rendezni lehet. A főoldalon részletes leírást találhatunk a különféle rajz- és festészeti technikákról, az egyedi megrendelés leadása menüpont alatt pedig leírás megadásával és fénykép feltöltésével illusztrálva kérhet ajánlatot az érdeklődő.

2. 2. Rendszerkövetelmények

A weboldal még nincs közzé téve, ezért nem elegendő a böngésző címsorába beírni a weboldal elérési címét. Hanem rendelkeznünk kell a megfelelő alkalmazásokkal, hogy futtatni tudjuk a

weboldalt. Szükségünk van egy fejlesztőkörnyezetre (mi Visual Studio Code-ot, illetve IntelliJ IDEA-t használtunk). Emellett szükséges a Xampp szoftver, amely segít az adatbázis kezelésében és a weboldal lokális futtatásában, valamint a Spring Boot és a JDK 17.

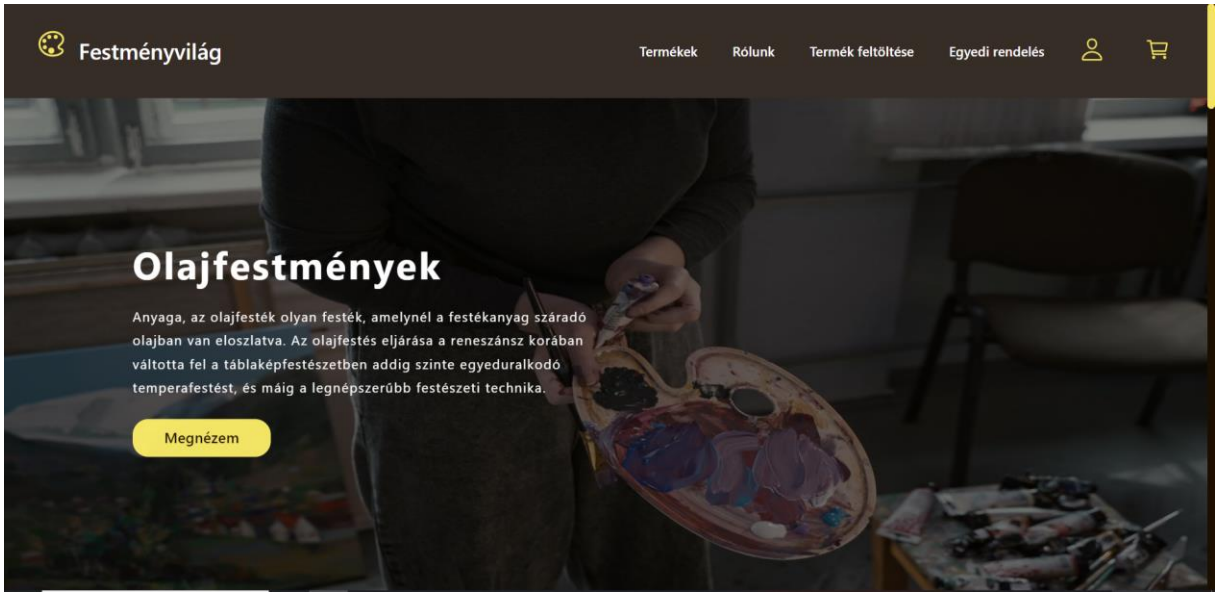
2. 3. A program telepítése

A program böngészőben fut, így nincs szükség az előző pontban említett szoftverek telepítésén kívül másra.

2. 4. A program használatának részletes bemutatása

A weboldal külön funkciókat enged a bejelentkezett felhasználóknak és a nem bejelentkezett látogatóknak. A weboldal betöltését követően az oldal funkcióit a navigációs sávból könnyedén elérhetjük.

2. 4. 1. Főoldal

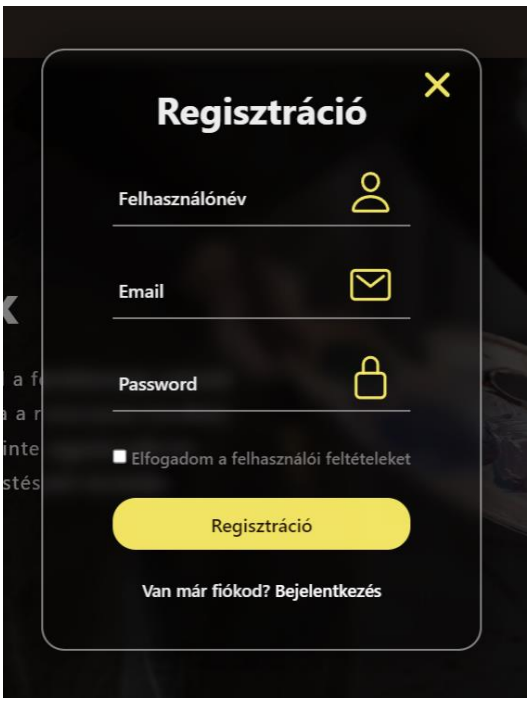


1. ábra: Főoldal

Regisztráció és bejelentkezés nélkül is elérhetjük. Tetején egy mp4 fájl (videó) fogadja az látogatókat, figyelemfelhívó címmel. Lejjebb görgetve az online piactér szolgáltatójáról található információ. Ha tovább görgetünk, a kiemelt alkotásokat láthatjuk, képernyőmérettől függően egyet, kettőt, vagy hármat. Itt az alkotásokról közzétett kép kicsinyített méretben látható, ezenkívül csak a legfontosabb információk láthatók rajta (az ára és a mérete), valamint egy „bővebben” gomb, melyre kattintva külön html oldalra kerülünk, ahol az összes információt láthatjuk az adott festményről vagy rajzról. Ezen a ponton még mindig tudunk lejjebb görgetni,

ekkor az egyes technikákról készített, kártyákba rendezett leírásokat tekinthetjük meg. Végül, de nem utolsó sorban galéria is helyett kapott a főoldalon, ahol szintén néhány kiemelt rajz vagy festmény tekinthető meg.

2. 4. 2. Regisztráció és bejelentkezés



2. ábra: Regisztrációs pop-up

Ha még nem rendelkezünk érvényes regisztrációval és szeretnénk feltölteni egy alkotásunkat, a navigációs sáv jobb szélén lévő ikonra, vagy a láblécben elhelyezett linkre kattintva érhetjük el a regisztrációt, mely pop-up ablakban ugrik fel. Itt meg kell adnunk egy email címet, annak formai követelményeinek betartásával, valamint leendő felhasználónevünket és jelszavunkat.

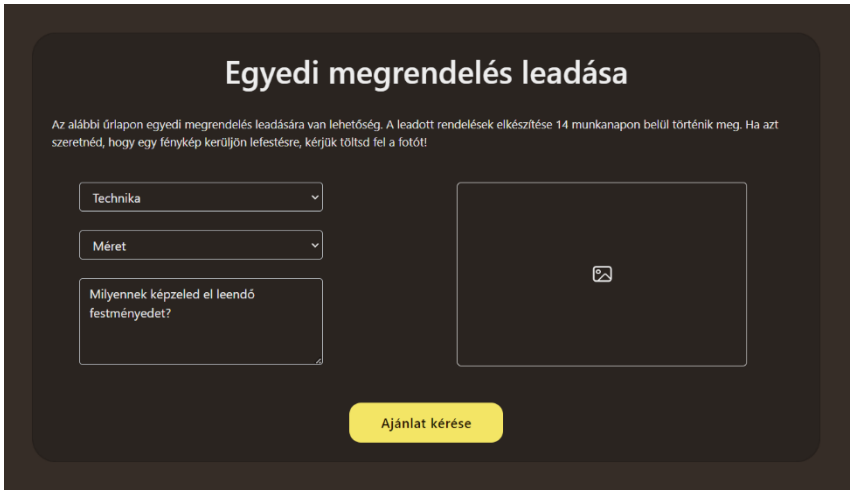
2. 4. 3. Termék feltöltése

3. ábra: Termék feltöltése

Ez a menüpont csak akkor érhető el, ha a felhasználó be van jelentkezve – azt megelőzően ha rákattint, a belépéshez kalauzoljuk el őt. Ha ez megtörtént, megjelenik az űrlap a felhasználó számára. Itt a festmény nevét, a művész nevét, a készítés évét, a termék árát, a rajz- vagy festészeti technikát, a témát, a méretet, illetve egy rövid leírást van lehetőség megadni, valamint egy képet beilleszteni a számítógépről. A termék áránál, illetve a készítés événél egy külön gomb segítségével is növelhetjük vagy

csökkenthetjük a beírt értéket. A technika és a téma esetében egy-egy lenyíló menüből választhatunk, melynél egyszerre többet is meg lehet jelölni.

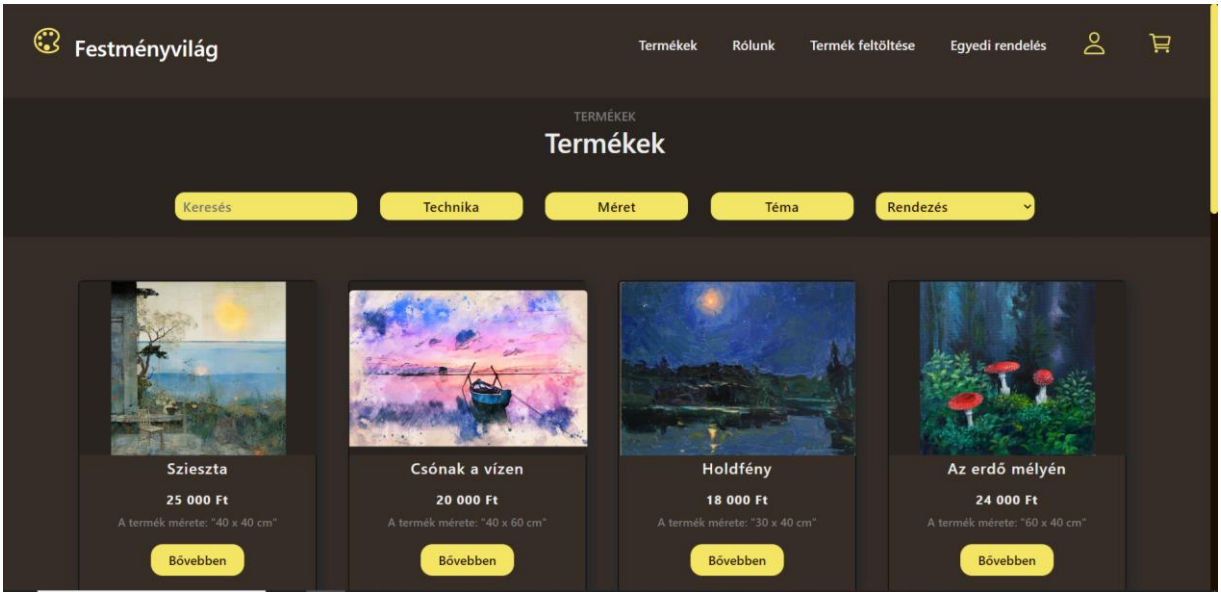
2. 4. 4. Egyedi megrendelés



4. ábra: Egyedi megrendelés leadása

Ennek a menüpontnak az elérése is bejelentkezéshez kötött. A felhasználónak kötelezően méretet és rövid leírást, opcionálisan technikát kell megadnia, valamint fotó feltöltésére is lehetősége van.

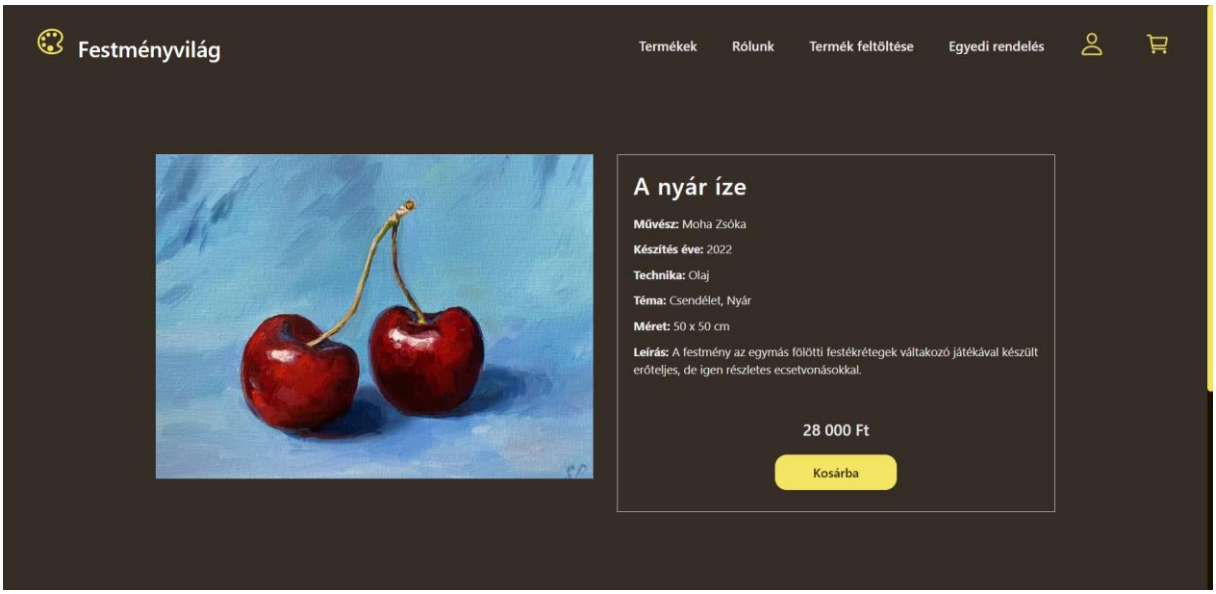
2. 4. 5. Kosárba helyezés és törlés



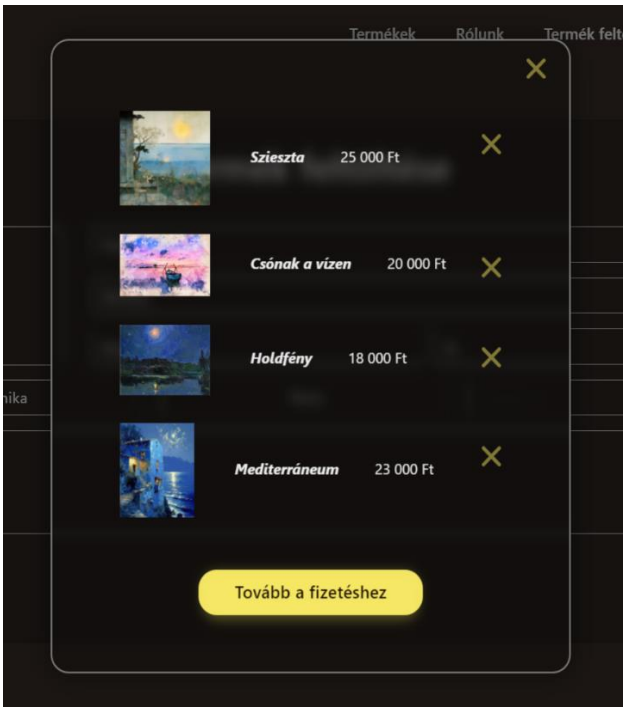
5. ábra: Termékek

A termékek menüpont alatt lehetőségünk van szűrni a termékekre, annak technikája, mérete és témája szerint. Sorba is rendezhetjük őket annak tükrében, hogy milyen szabály szerint szeretnénk látni őket: ár szerint növekvő, csökkenő, téma vagy technika szerinti ABC sorrend, illetve a legfrissebb megjelenítése legelől. Az alkotások 12-ével rendeződnek egymás mellett, tehát egyszerre 12-t láthatunk az oldalon, majd lapozással léptethetjük őket, hogy továbbiakat

lássunk. A kártyákon szereplő kép kicsinyített méretben látható, ezenkívül csak a legfontosabb információk szerepelnek rajta (az ára és a mérete), valamint egy „bővebben” gomb, melyre kattintva külön html oldalra kerülünk, ahol az összes információt láthatjuk az adott festményről vagy rajzról. Itt tehetjük kosárba a terméket.



6. ábra: A termék bővebb részletei



7. ábra: A kosár tartalma

A kosár tartalmát a navigációs sáv jobb szélén elhelyezett kosár ikonra kattintva tekinthetjük meg. A regisztrációhoz és a bejelentkezéshez hasonlóan pop-up ablakban kapott helyet. Ha a termék neve melletti X-re kattintunk, törölhetjük a kosarunkból.

Fejlesztői dokumentáció

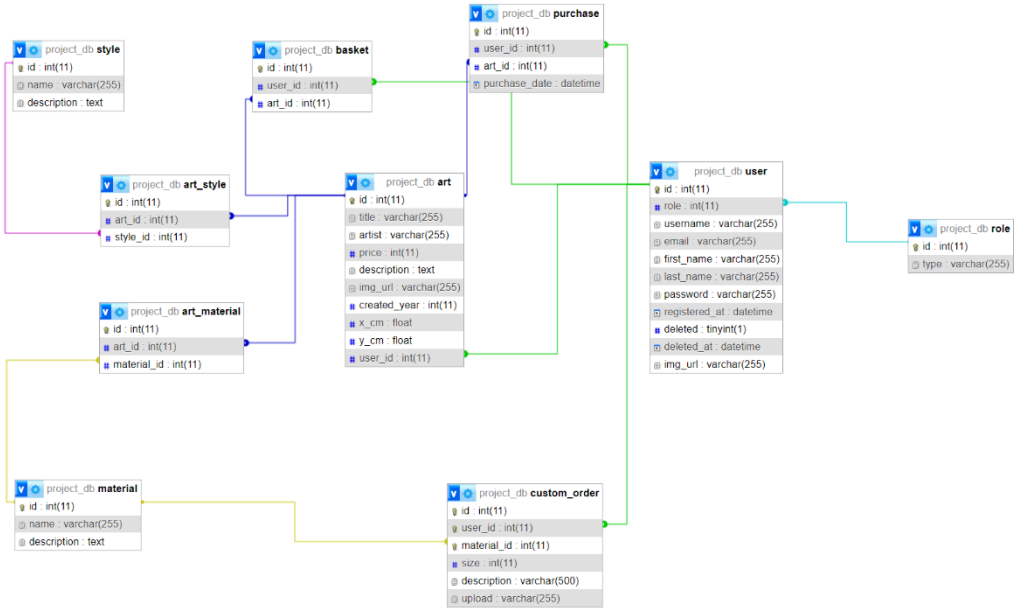
A fejlesztés során igyekeztünk a clean code elveinek megfelelni, vagyis, hogy a kód olvasható, könnyen megérthető legyen. Például a változók, függvények és osztályok neveinek megválasztásakor figyelembe vettük azok leíró jellegét, hogy a kód struktúrája egyértelmű legyen és könnyen követhetővé váljon. Az ismétlődést igyekeztünk minimalizálni, és az egyszerűség elvét követni, hogy a kód a jövőbeli fejlesztések során is könnyen karbantartható és bővíthető legyen.

3. 1. Az alkalmazott fejlesztői eszközök

A frontend rész, amely az alkalmazás felhasználói felületét biztosítja, Visual Studio Code fejlesztőkörnyezetben készült. A kód HTML, CSS és JavaScript nyelven íródott, biztosítva ezzel a felhasználóbarát és interaktív felületet. Az adatbázis létrehozása és kezelése során PHPMyAdmin szoftvert használtunk. Ezáltal lehetőségünk volt hatékonyan kezelni az adatbázissal kapcsolatos teendőket, mint például az adatbázis létrehozása, szerkezetének módosítása és az adatok kezelése. A backend rész Spring Boot keretrendszerben készült Java programozási nyelven, IntelliJ IDEA fejlesztőkörnyezetben. Postman segítségével teszteltük és dokumentáltuk az API végpontokat, valamint a GitHubot használtuk a verziókezelés és a csapatmunka hatékony koordinálása céljából.

3. 2. Adatmodell leírása

Az adatbázis tervezéséhez a dbdiagram.io platformot használtuk, ami segítségünkre volt a táblák, az oszlopok és kapcsolatok kialakításának megtervezésében.



8. ábra: Adatbázismodell-diagram

A fenti ábrán az adatbázisunk felépítése látható, a táblák elnevezése beszédes, hogy ránézésre megállapítható legyen, hogy miért felelnek. Az elsődleges kulcs minden táblában az „id” mező, így az ID automatikusan generálódik le. Azok az oszlopok, melyek egy idegen kulcsot jelölnek, úgy kerültek elnevezésre, hogy az első szó a másik tábla neve, amiből az adatot átvettük, majd egy alsóvonás után annak a mezőnek a neve szerepel, amit átvettünk. Az adatmezők meghatározásakor összesen hatféle adattípust alkalmaztunk: int, tinyint, float, varchar, text és datetime típusokat. A karakterláncok mérete a hozzá tartozó elem valószínűsíthető méretéhez igazodik, így például legtöbb helyen 255 karaktert határoztunk meg a varcharnak, míg a descriptionnél (leírásnál) 500 karaktert.

Az art táblában tároljuk az alkotásokat, azok minden jellemzőjével, így például méretével. A szélesség és a magasság két külön paraméterként kerül meghatározásra. A user_id mező egy idegen kulcs, ennek segítségével tároljuk el, hogy melyik felhasználó hirdette meg az adott alkotást.

Az art_style és az art_material táblák kapcsolótáblák, amik összekapcsolják az artot (a festményeket és rajzokat tartalmazó táblát) a material és a style táblával, amik a technikákat és a témákat tartalmazzák.

A basket tábla a kosarat jelöli, benne az alkotás azonosítójával. A custom_order tábla az egyedi megrendeléseket tartalmazza.

A user táblában tároljuk a felhasználókat. Több időbélyeget is tárolunk, mint, hogy mikor történt a regisztráció (registered_at), illetve mikor került törlésre (deleted_at). A felhasználó jogosultságát a role mező jelöli, a jelszavak SHA256 titkosítási algoritmussal vannak ellátva. Ez egyike a legelterjedtebb és legbiztonságosabb hash függvényeknek, amelyeket jelszavak vagy más érzékeny információk titkosítására használnak. Ez a függvény egy olyan hashet hoz létre, amely egy fix hosszú karakterláncot eredményez a bemeneti adat alapján.

3. 3. Részletes feladatspecifikáció, algoritmusok

A frontend rész fejlesztése során a Bootstrap grid rendszerét használtuk fel az elrendezések kialakításához, hogy reszponszív és rugalmas megjelenést hozzunk létre, mely jól igazodik a különböző méretű kijelzőkhöz. A Bootstrap alapértelmezett megjelenését gyakran felülírtuk saját CSS fájllal is, hogy testreszabjuk a stílust.

A JavaScript kód nagyrészét aszinkron fetch kérések alkotják, melyek segítségével kommunikálunk a backenddel. Ezekkel a kérésekkel történik meg az adatok lekérése (GET), az összes adat lekérése (GET ALL), új adatok hozzáadása (ADD), valamint a meglévő adatok frissítése vagy módosítása (POST) a szerver oldalán.

A munka során az egyik kihívást az jelentette, hogy elkülönítsük a bejelentkezett és nem bejelentkezett felhasználók számára elérhető funkciókat. A loginscript.js fájlban található „login” függvény felelős a felhasználó bejelentkezésének kezeléséért. Először a felhasználó által megadott felhasználónév és jelszó adatokat kéri be a megfelelő input mezőkből. Ezután egy POST kérést küld az "/auth/login" végpontra, amely az adott felhasználó bejelentkezését kezeli. A kérés törzsében JSON formátumban elküldjük a felhasználónév és a jelszó adatokat. Ha a kérés sikeresen teljesül, azaz a szerver visszaigazolja a sikeres bejelentkezést, akkor a válaszban kapott token értéket eltároljuk a localStorage-ban a "token" kulcs alatt. A sikeres bejelentkezésről tájékoztató üzenet jelenik meg a felhasználónak.

```
export function redirectToLogin() {
  console.log(localStorage.getItem("token"));
  const url = "http://127.0.0.1:8080/auth/validate";

  fetch(url, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Authorization: localStorage.getItem("token"),
    },
  })
    .then((response) => response.json())
    .then((data) => {
      console.log(data)
      if (data == false) {
        alert("Bejelentkezés szükséges");
        window.location.href = "/Main/index.html";
      } else {
        console.log("Be vagy jelentkezve");
      }
    })
    .catch((error) => {
      console.log(error);
      window.location.href = "/Main/index.html";
    });
}
```

9. ábra: Token ellenőrzés

A permission-checker.js fájlban megírt „redirectToLogin” függvény felelős érte, hogy ellenőrizze, be van-e jelentkezve a felhasználó. Először megpróbálja lekérni a tárolt token értékét a localStorage-ból, amely azt jelzi, hogy a felhasználó be van-e jelentkezve. Ezután a fetch függvény segítségével egy GET kérést küldünk a szerver felé az "/auth/validate" végpontra, amely megpróbálja ellenőrizni a tárolt token érvényességét és a felhasználó bejelentkezési státuszát. Ha a válasz "false" értékkel tér vissza, azaz a token érvénytelen vagy nem létezik, akkor felugrik egy figyelmeztető üzenet, hogy be kell jelentkezni, és az oldal átirányítódik a bejelentkezési oldalra. Ellenkező esetben, ha a válasz "true" értékkel tér vissza, a felhasználó be van jelentkezve.

A redirectToLogin függvényt importáltuk és meghívtuk az egyedi megrendelés leadásához és a termék meghirdetéséhez tartozó JavaScript fájlokban. Ezzel biztosítva, hogy ezen két menüpontra kattintáskor automatikusan megtörténjen a bejelentkezés ellenőrzése, és az átirányítás.

A következő kihívást maguknak a termékeknek a dinamikus megjelenítése okozta. A „Termékek” menüpontra kattintva kártyákba rendezve jelennek meg az alkotások, ha a felhasználó a „Bővebben” gombra kattint, akkor láthatja róluk az összes információt, valamint ott helyezheti őket kosárba. A product-details.js fájlban találhatjuk meg az utóbbi függvényeket. Itt a fetch függvény segítségével küldünk egy GET kérést a szervernek, amelynek az URL-je tartalmazza a termék azonosítóját. Erre azért van szükség, hogy biztosítsuk, hogy amikor a felhasználó az egyik terméknél rákattint a "bővebben" gombra, akkor a megfelelő termékhez navigáljuk át. Először az volt a tervünk, hogy egy pop up ablakban fognak megjelenni a részletes információk az adott alkotásról, ezt azonban elvetettük, mert kisebb méretű kijelzőn a sok információt nehéz szépen elhelyezni egy pop up ablakban, és kevésbé elterjedt is.

A „kosarba” függvény a "Kosárba" gombra való kattintáskor hívódik meg. Ez a függvény először létrehoz egy formData objektumot, amely tartalmazza a felhasználó és a termék

azonosítóját. Ezután egy POST kérést küld a szervernek a /basket/save végpontra, amely a kiválasztott termék felvétele a kosárba műveletet végrehajtja. A „felhasznalo” függvény segítségével pedig kinyerjük a felhasználó azonosítóját a localStorage-ból.

Ezt követően, ha a felhasználó a navigációs sávban található kosár ikonra kattint, megtekintheti, hogy miket tett eddig a kosarába. Ezért a basket.js fájlban lévő kód felel, ami kezeli a pop-up ablakot. Lekéri a szerverről a /basket/get végponton keresztül a felhasználó kosarában található termékek listáját aszinkron módon, a fetchProducts() függvénnyel. Miután a szerver válaszolt, a kapott adatokat egy listába rendezzük, majd megjelenítjük őket a renderProducts() függvény segítségével.

Fontosnak tartottuk megvalósítani, hogy a termékek között szűrni, keresni és rendezni is lehessen.

```
// Szűrés funkció
function filterProducts() {
    const selectedStyles = getSelectedStyles(); // Kiválasztott stílusok lekérése
    const selectedMaterials = getSelectedMaterials(); // Kiválasztott anyagok lekérése

// Ellenőrizzük, hogy van-e kiválasztott szűrő
if (selectedStyles.length === 0 && selectedMaterials.length === 0) {
    // Ha nincs kiválasztott szűrő, akkor az összes termék megjelenjen
    filteredProducts = prodList;
} else {
    // Ha csak az egyik szűrő van kiválasztva, akkor az összes többi opció automatikusan kiválasztódik
    if (selectedStyles.length === 0 && selectedMaterials.length > 0) {
        console.log("csak material")
        filteredProducts = prodList.filter((termek) => {
            return selectedMaterials.some((material) => termek.material.find((m) => m.name === material))})
    } else if (selectedMaterials.length === 0 && selectedStyles.length > 0) {
        console.log("csak style")
        filteredProducts = prodList.filter((termek) => {
            return selectedStyles.some((style) => termek.style.find((s) => s.name === style))})
    } else if (selectedMaterials.length > 0 && selectedStyles.length > 0) {
        console.log("mind kettő")
        filteredProducts = prodList.filter((termek) => {
            return selectedStyles.some((style) => termek.style.find((s) => s.name === style)) &&
                   selectedMaterials.some((material) => termek.material.find((m) => m.name === material))})
    }
}
    renderProducts(); // Termékek újraszűrése
}
```

10. ábra: A szűrés

A szűrés funkció három fő részből áll: a „filterProducts” függvény határozza meg, hogy mely termékeket kell megjeleníteni a felhasználó választása alapján. A szűrést a „prodList” tömbön végezzük, majd a szűrt termékek kerülnek tárolásra a „filteredProducts” tömbben.

A „getSelectedStyles” és a „getSelectedMaterials” függvények a felhasználó által kiválasztott stílusokat és anyagokat gyűjtik össze a szűrőkből. Amikor a felhasználó megváltoztatja a szűrők állapotát, automatikusan meghívódik a „filterProducts” függvény, és újraszűri a termékeket az új szűrési feltételek alapján.

A termékek rendezésére többféle szempont szerint is lehetőség van: ár szerint növekvő vagy csökkenő, téma, illetve anyag szerinti ABC sorrend, valamint a feltöltés dátuma szerinti rendezés. Az „orderProducts” függvény biztosítja, hogy a felhasználó az általa kívánt sorrendben láthassa a termékeket az oldalon. Egy switch utasítás segítségével ellenőrizzük, hogy melyik rendezési opció lett kiválasztva, és a „filteredProducts” tömböt rendezve módosítjuk a megjelenítés sorrendjét.

A „searchProducts” függvény végzi a termékek keresését a felhasználó által beírt kulcsszó alapján. Ebben a függvényben a „filteredProducts” tömbön alkalmazzuk a „filter” metódust, mely minden terméket átvizsgál, és csak azokat tartja meg, amelyek címe tartalmazza a keresési kulcsszót (beleértve a kis- és nagybetűket is). Így szűrjük és frissítjük a termékek listáját, hogy csak azok jelenjenek meg, amelyek megfelelnek a keresési feltételeknek.

A backend rész Java nyelven készült Spring Boot keretrendszerben. Elsőként az entity osztályokat hoztuk létre. Ezek az osztályok tartalmazzák az adatbázisban tárolt adatok struktúráját, leképzik az adatbázis tábláit Java osztályokká. A DTO (Data Transfer Object) osztályok az adatátvitel egyszerűsítésére szolgálnak a különböző rétegek között. A repository osztályok az adatbázis műveletek végrehajtásáért felelősek. Ezek tartalmazzák a CRUD (create, read, update, delete) műveleteket az adatbázissal való interakcióhoz. Ezt követően létrehoztuk a mapper osztályokat, amik a Dto és az Entity osztályok közötti átalakításért felelősek.

A service osztályok közül a BasketService valósítja meg a kosárkezeléssel kapcsolatos fontosabb funkciókat. A saveBasket metódus egy terméket ad a kosárhoz, ellenőrzi a termék és a felhasználó létezését, majd létrehoz egy új kosarat, ha szükséges. A removeArtFromBasket metódus eltávolít egy adott terméket a kosárból, ellenőrzi a termék és a kosár létezését, valamint a felhasználó jogosultságát. A deleteWholeBasket metódus törli a felhasználó teljes kosarát és annak tartalmát, ellenőrzi a kosár létezését és a felhasználó jogosultságát. A getOwnedBasketItems metódus pedig visszaadja a felhasználó összes kosárban lévő termékét DTO-ként, lekéri a felhasználó kosarait és azok tartalmát.

Egy másik fontos service osztály az ArtService. Az itt megírt metódusok lehetővé teszik új termék mentését az adatbázisba, egy adott termék lekérését az adatbázisból ID alapján, vagy törlését a felhasználó jogosultságának ellenőrzése után. Le lehet kérni továbbá a felhasználó összes feltöltött termékét, illetve az összes terméket.

```
1 usage  Tivadar
public ResponseDto login(UserDto userDto) throws NoSuchAlgorithmException {
    if (userDto.getUserName() == null || userDto.getUserName().isBlank()
            || userDto.getPassword() == null || userDto.getPassword().isBlank()){
        throw new RuntimeException("Hibások az adatok");
    }
    Optional<User> byUsername = userRepository.findByUsername(userDto.getUserName());
    if (byUsername.isEmpty()){
        throw new RuntimeException("Nincs ilyen felhasználónév");
    }
    if (!byUsername.get().getPassword().equals(hashUtil.getSHA256Hash(userDto.getPassword()))){
        throw new RuntimeException("Hibás jelszó");
    }

    return new ResponseDto(jwtUtil.generateJwt(byUsername.get()));
}
```

11. ábra: Login

A UserService osztály a felhasználók regisztrációját és bejelentkezését valósítja meg. A fenti képen a bejelentkezés látható. Ha a bejelentkezés sikeres, egy JWT token kerül generálásra, melyet a ResponseDto tartalmaz.

A controller osztályok a REST API végpontokat tartalmazzák. Ezek az osztályok fogadják a beérkező HTTP kéréseket, feldolgozzák azokat, és visszaadják a válaszokat a kliensnek.

```
public boolean validateJwt(String authHeader) {
    if(authHeader.equals("null")){
        return false;
    }
    try{
        String cleanToken = authHeader.split( regex: " ")[1];
        Jwts.parser().setSigningKey(secretKey).parseClaimsJws(cleanToken);

        String[] parts = cleanToken.split( regex: "\\.");
        JSONObject data = new JSONObject(decode(parts[1]));

        if (data.getLong( key: "exp") > (System.currentTimeMillis() / 1000)) {
            return true;
        }
        return false;
    }catch (IllegalArgumentException | MalformedJwtException | NullPointerException |
            ExpiredJwtException e){
        return  false;
    }
}

3 usages  Tivadar
private static String decode(String encodedString) {
    return new String(Base64.getUrlDecoder().decode(encodedString));
}
```

12. ábra: Token ellenőrzés

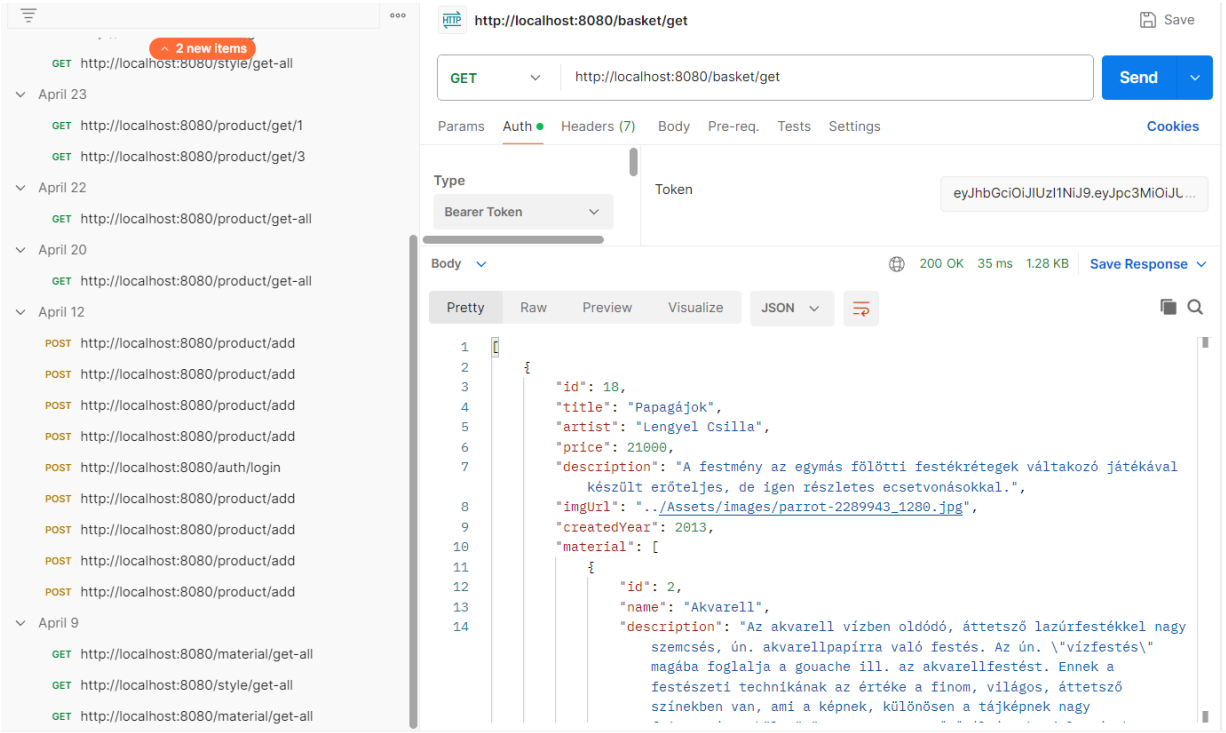
A JwtUtil osztály a JWT tokenek kezeléséhez szükséges segédfüggvényeket tartalmazza: JWT token generálása a felhasználó adatai alapján, a token érvényességének ellenőrzése, tartalmának dekódolása a felhasználó adatainak kiolvasásához. Valamint az azonosító kinyerése az authentikációs fejlécből származó JWT tokennel.

A backend rész tartalmaz továbbá egy RoleEnum osztályt, ami egy szerepköröket (felhasználó és admin) felsoroló enum. Valamint security osztályokat, melyek felelősek az autentikációért,

autorizációért és a felhasználói adatok kezeléséért a biztonságos működés érdekében. A RequestFilter szűrő minden bejövő HTTP kérést ellenőriz, és JWT autentikációt hajt végre az érkező kérésekben. A SecurityConfig osztály konfigurálja a biztonsági beállításokat, meghatátozza azokat a kéréseket, amelyek elérhetőek, és azokat, melyekhez hitelesítés szükséges. A UserDetailsServiceImp osztály kezeli a felhasználói adatok betöltését. Amikor egy felhasználó bejelentkezik, a Spring Security ezt az osztályt használja az azonosító és a jelszó ellenőrzésére, valamint a felhasználói szerepkörök betöltésére a megfelelő jogosultságokhoz.

3. 4. Tesztelési dokumentáció

A program tesztelése a fejlesztéssel párhuzamosan valósult meg. Főként egységteszteket és funkcionális teszteket hajtottunk végre. Ehhez külön teszt kódokat nem írtunk, hanem külön kézzel próbálgattuk a REST API kéréseket „Postman” program segítségével.



13. ábra: Tesztelés

Az alábbi ábrán az egyik Postmanes teszt látható. Példaként az adatbázis „kosár” táblájának elemeit kértük le. Ehhez backenden megírtuk a végpontot, amire egy GET kérést kell elküldeni. Sikeres válasz eléréséhez egy érvényes „JWT” szükséges, amit bejelentkezéskor kap az adott felhasználó. Ebből a Jason Web Tokenből, aztán kiszedi a bejelentkezett felhasználó „id”-ját, ami alapján lekéri a hozzá tartozó kosarat és a benne lévő termékeket. Sikeres kérés esetén egy

200-as http kódot és a kosárban lévő termékeket kapjuk vissza JSON formában. Érvénytelen token, vagy nem létező felhasználó id esetén egy 400-as, vagy 500-as http kódot kapunk vissza, a hozzátartozó üzenettel. Néhány endpoint-hoz külön hibaüzeneteket írtunk, amit frontend oldalon meg is jelenítünk. Például bejelentkezéskor kiírja felugró ablakban, hogy „Sikeres bejelentkezés", vagy épp „Hibás felhasználónév vagy jelszó" stb. Ezek tesztelés során is megkönnyítették a dolgunkat és éles helyzetben is pontosabb magyarázatot adhatnak a felhasználóknak. Tesztelés során, ami hibákat találtunk igyekeztünk egyből kijavítani. Legtöbbször szintaktikai hibákat vétettünk.

3. 5. Továbbfejlesztési lehetőségek

A fejlesztés során arra törekedtünk, hogy tiszta és átlátható kódot írjunk, ami karbantartható és sokoldalúan fejleszthető. A legkézenfekvőbb bővítés következő lépésben egy olyan űrlap létrehozása lenne, melyen a felhasználók módosíthatnak a felhasználói profiljukon, például profilképet adhatnak meg. További logikus kiegészítés lenne, hogy az általuk meghirdetett alkotásról megadott információkon is módosítani tudjanak. A felhasználók közötti levelezést is érdemes lenne lehetővé tenni. A projekt ugyanakkor jelenlegi formájában átalakítható webshoppá is.

Összegzés

A projekt az elsődleges célját teljesítette, tekintve, hogy ez az volt, hogy minél többet tanulhassunk belőle. A szerzett tudás értékes alapkőként szolgál egy későbbi aktív projekthez, illetve azt sem vetjük el, hogy továbbfejlesszük a projektet és új funkciókkal kísérletezzünk. A fejlesztési folyamat egy rögös út, sok hibával találkoztunk, ami fejlesztette a problémamegoldó készségünket. A csapatmunka pedig lendületet adott a munkához.

Irodalomjegyzék, forrásmegjelölés

Webes hivatkozások:

- W3 School tutorialok:
  - W3S CSS Tutorial: https://www.w3schools.com/css/
  - W3S JavaScript Tutorial: https://www.w3schools.com/js/

- W3S Java Tutorial:
  https://www.w3schools.com/java/