Bevezetés

A projektünk témájául a HovaTovább nevű menetrendtervező alkalmazást választottuk, mivel a mindennapi közlekedés során kiemelten fontos a gyors, pontos és megbízható információk elérése. A magyar tömegközlekedési menetrendek jelenleg több különböző felületen érhetők el, amelyek gyakran nehezen átláthatók, lassúak, vagy nem minden felhasználói igényt szolgálnak ki megfelelően.

Projektünk célja az volt, hogy egy olyan alkalmazást tervezzünk, amely felülmúlja a már meglévő menetrend weboldalakat, kijavítja azok hiányosságait, és egy modernebb, felhasználóbarátabb megoldást kínál.

Személyes tapasztalatok alapján, egy utazás során felmerült egy olyan probléma hogy a késést nem jelzően nem kaptunk elég információt az adott járatról. Ebből fakadóan főképp gondoltuk hogy ezt a projekttervet választjuk.

A HovaTovább alkalmazás egy olyan menetrendtervező rendszer, amely a magyar tömegközlekedés menetrendjeit teszi könnyen és hatékonyan elérhetővé.

Az alkalmazás fejlesztése során kiemelt figyelmet fordítottunk arra, hogy a felhasználók minél kevesebb lépéssel, gyorsan és érthetően juthassanak hozzá az őket érdeklő információkhoz. A cél nem csupán egy újabb menetrend-alkalmazás létrehozása volt, hanem egy olyan rendszer megalkotása, amely valódi segítséget nyújt a mindennapi utazások megtervezésében.

Az alkalmazás egyik legfontosabb funkciója az automatikus útvonaltervezés, amely a megadott kiindulási pont és célállomás alapján megkeresi a legoptimálisabb útvonalat. Ez különösen hasznos lehet azok számára, akik nem ismerik pontosan a járatokat, vagy új útvonalon szeretnének közlekedni. Emellett a HovaTovább lehetőséget biztosít manuális menetrend-összeállításra is, így a tapasztaltabb felhasználók saját igényeik szerint állíthatják össze az utazásukat, például ha egy adott járatot vagy átszállást preferálnak.

A projekt kiválasztásakor fontos szempont volt számunkra a rugalmasság és a személyre szabhatóság.

A HovaTovább alkalmazás ezekre az igényekre reagál, hiszen különböző utazási szokásokhoz képes alkalmazkodni. Legyen szó napi munkába járásról, alkalmi utazásról vagy hosszabb út megtervezéséről, az alkalmazás célja, hogy minden helyzetben megbízható segítséget nyújtson.

A projekt megvalósítása kiváló lehetőséget biztosított arra, hogy gyakorlati tapasztalatot szerezzünk a szoftverfejlesztés különböző területein. A tervezési folyamat során megismerkedtünk a követelmények elemzésével, a funkciók megtervezésével, valamint a fejlesztési lépések logikus felépítésével. Emellett nagy hangsúlyt kapott a hibák felismerése

és javítása, hiszen a meglévő rendszerek hiányosságainak kijavítása az alkalmazás egyik alapvető célja volt.

A projekt során kiemelt figyelmet fordítottunk a felhasználói élmény (UX) megtervezésére is. Fontos szempont volt számunkra, hogy az alkalmazás kezelése intuitív legyen, vagyis a felhasználók különösebb magyarázat nélkül is könnyedén eligazodjanak benne. A letisztult felépítés, az átlátható menüpontok és az egyértelmű visszajelzések mind hozzájárulnak ahhoz, hogy az alkalmazás használata gyors és kényelmes legyen, akár mobiltelefonon, akár számítógépen történik a hozzáférés.

Végül, de nem utolsósorban a projekt célja az volt, hogy egy továbbfejleszthető alapot hozzunk létre. A HovaTovább alkalmazás a jövőben kiegészíthető további funkciókkal, például értesítésekkel, kedvenc útvonalak mentésével vagy valós idejű járatinformációk megjelenítésével. Ez biztosítja, hogy az alkalmazás hosszú távon is releváns és hasznos maradhasson a felhasználók számára.

Összességében a HovaTovább projektet azért választottuk, mert egy valós problémára kínál megoldást, és lehetőséget adott arra, hogy egy korszerű, hasznos és felhasználóközpontú alkalmazást tervezzünk. A projekt jól ötvözi az informatikai megoldásokat a mindennapi élet gyakorlati igényeivel, és tapasztalatot nyújt a modern alkalmazásfejlesztés területén.

A főoldal átfogó szakmai bemutatása

A HovaTovább webalkalmazás főoldala egy magyar nyelvű, tömegközlekedési útvonaltervező rendszer belépési felülete, amelynek elsődleges célja a felhasználók gyors és hatékony útvonaltervezésének támogatása. Az oldal sötét témájú megjelenést alkalmaz, amely modern, letisztult hatást kelt, és hosszabb használat során is kíméli a szemet. A vizuális kialakítás tudatosan a funkcionalitást helyezi előtérbe, miközben minden fontos elem jól elkülönül és könnyen felismerhető.

Az oldal felső részén található fejléc biztosítja az alkalmazás azonosítását és az alapvető felhasználói műveletek elérését. A bal felső sarokban elhelyezett HovaTovább logó és felirat egyértelműen jelzi az alkalmazás nevét, míg az alatta szereplő „Közösségi menetrendtervező” szöveg pontosan meghatározza a rendszer funkcióját. Ez különösen fontos első látogatáskor, mivel azonnal világossá teszi, hogy milyen szolgáltatást nyújt az oldal.

A fejléc jobb oldalán helyezkedik el a bejelentkezési és regisztrációs lehetőség, amely a felhasználói fiókhoz kötődő funkciók elérését teszi lehetővé. Ezek a funkciók a későbbi személyre szabás, mentett útvonalak és egyéni beállítások alapját képezik.

A fejléc alatt található a navigációs sáv, amely két fő nézet között teszi lehetővé a váltást. Az aktív „Keresés” fül jelzi, hogy a felhasználó jelenleg új útvonal tervezését végzi, míg a „Terveim” fül a korábban elmentett útvonalak kezelésére szolgál. Ez a felosztás logikus és felhasználóbarát, mivel elválasztja az egyszeri keresési műveleteket a hosszabb távon használt, személyes tartalmaktól.

A főoldal központi eleme a keresési felület, ahol az útvonaltervezéshez szükséges adatok megadása történik. Itt található a kiindulási hely megadására szolgáló mező, amely a *„Honnan…”* feliratot viseli, valamint a célállomás megadására szolgáló *„Hová…”* mező. Ezek a mezők az útvonaltervezés alapját jelentik, hiszen a rendszer ezek alapján képes kiszámítani az optimális közlekedési útvonalat. A két mező között elhelyezett iránycsere gomb lehetőséget biztosít arra, hogy a felhasználó egyetlen kattintással felcserélje a kiindulási és célállomást, ami különösen hasznos oda-vissza utak tervezése esetén.

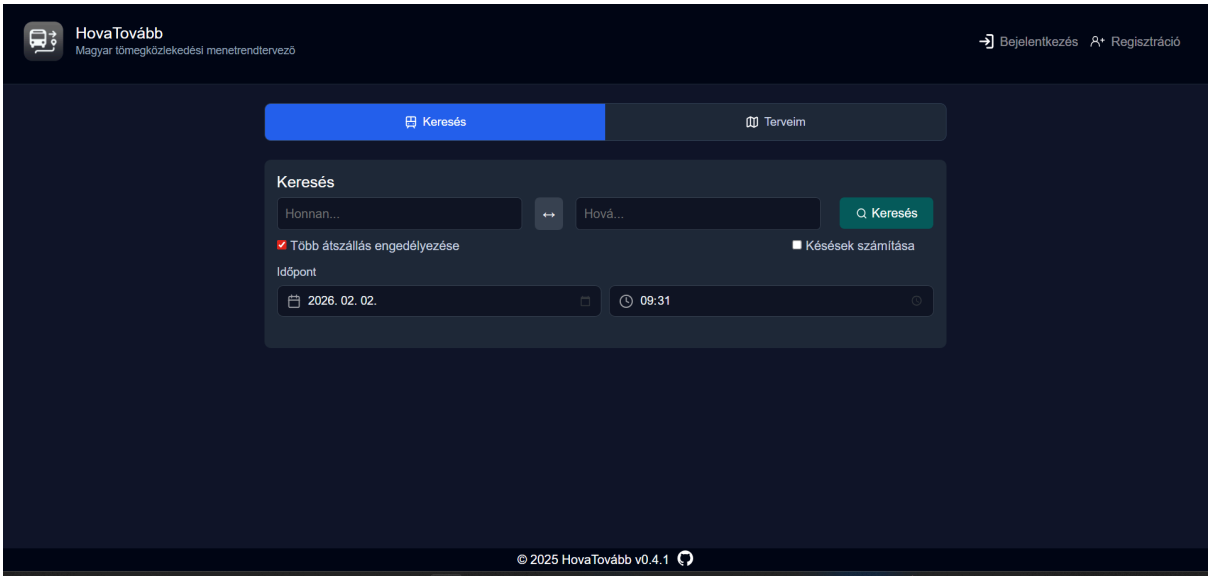
A keresési felületen további beállítások is elérhetők, amelyek az útvonaltervezés részleteit finomítják. A több átszállás engedélyezésére szolgáló jelölőnégyzet lehetővé teszi, hogy a rendszer összetettebb útvonalakat is figyelembe vegyen, ezáltal növelve az elérhető alternatívák számát. Ezzel szemben a késések számítását vezérlő opció a valós idejű közlekedési adatok figyelembevételét teszi lehetővé, amely pontosabb érkezési időket eredményezhet, amennyiben ilyen adatok rendelkezésre állnak.

A keresési feltételek megadásának fontos része az időpont beállítása, amely külön dátum- és időmező segítségével történik. A dátum kiválasztása azért kiemelten lényeges, mert a tömegközlekedési menetrendek eltérhetnek hétköznapokon, hétvégéken vagy ünnepnapokon. Az idő megadásával a felhasználó pontosíthatja, hogy mikor szeretne elindulni, így a rendszer az adott időponthoz legjobban illeszkedő járatokat tudja kiválasztani.

A keresési folyamatot a zöld színnel kiemelt „Keresés” gomb indítja el, amely vizuálisan is egyértelműen elkülönül a többi elemtől. Ez a gomb az oldal elsődleges műveleti eleme, amelynek megnyomásával a rendszer feldolgozza a megadott adatokat, majd megjeleníti a lehetséges útvonalakat.

Az oldal alsó részén elhelyezkedő lábléc tartalmazza az alkalmazás verziószámát és a szerzői jogi információkat, valamint egy GitHub ikon formájában hivatkozást biztosít a projekt fejlesztői felületére. Ez a rész elsősorban szakmai és fejlesztői szempontból jelentős, mivel lehetőséget ad a verziókövetésre és a projekt nyílt forráskódú jellegének megismerésére.

Összességében a HovaTovább főoldala egy jól strukturált, felhasználóközpontú felület, amely világosan elkülöníti az egyes funkcionális területeket, miközben egyszerű és gyors használatot biztosít. A kialakítás támogatja mind az alkalmi, mind a rendszeres felhasználókat, és megfelelő alapot nyújt a további funkciók bővítéséhez és fejlesztéséhez.



Fiókkezelés

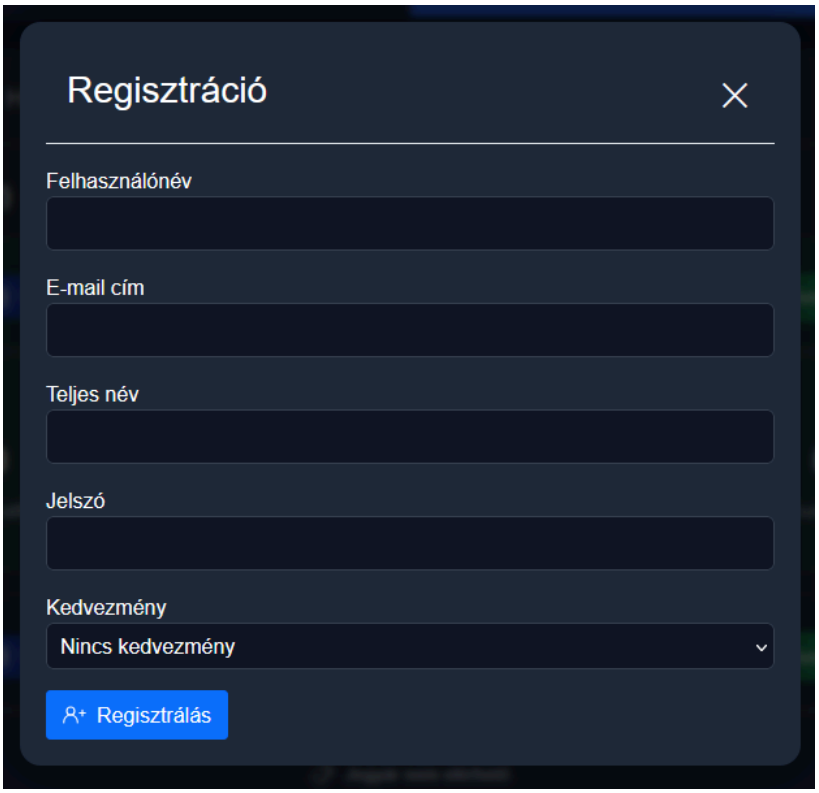
A fiókkezelés az alkalmazás személyre szabott működését biztosítsa. A modul lehetővé teszi, hogy a felhasználó saját fiókot hozzon létre, biztonságosan tudjon regisztrálni/bejelentkezni, személyes adatait kezelje, és az alkalmazás funkcióit (pl. tervek, kedvezmények) felhasználóhoz kötötten használja.

A fiókkezelés nélkülözhetetlen ahhoz, hogy az alkalmazás ne csak egy egyszeri keresőfelület, hanem tartósan használható utazásszervező rendszer legyen.

Regisztráció

A regisztráció egy **modal ablakban** történik, amely nem navigál el az oldalról, valamint gyors és megszakításmentes felhasználói élményt biztosít. Egyszerűen kezelhető és jól átlátható.

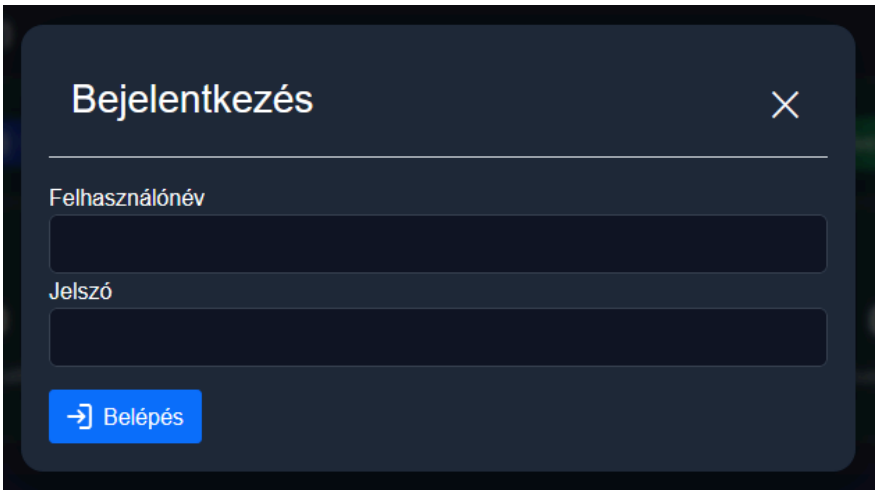
A felhasználónak meg kell adnia a felhasználónevét, az e-mail címét, a teljes nevét, a jelszavát, és a kedvezmény típusát (teljes árú, 50%-os, díjmentes).



A regisztráció csak akkor engedélyezett, ha minden kötelező mező ki van töltve. A jelszó nem jelenik meg visszajelzésként, védett mezőben kerül megadásra. Sikeres regisztráció után a felhasználó visszajelzést kap, majd bejelentkezhet.

Bejelentkezés

A bejelentkezés szintén egy modal ablakban történik, így a felhasználó nem veszti el az aktuális oldalt, és a folyamat gyors és intuitív marad. A felhasználónak bejelentkezéskor a felhasználónevét és jelszavát kell megadnia.

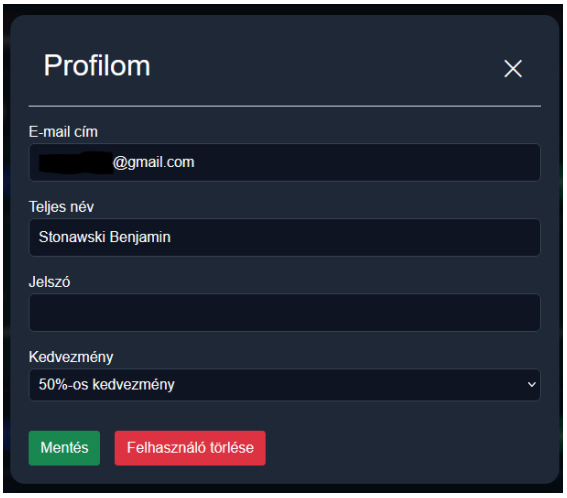


Hibás adatok esetén a rendszer egyértelmű hibaüzenetet jelenít meg. Sikeres bejelentkezés után a felhasználó neve megjelenik az alkalmazás felületén, elérhetővé válnak a személyes funkciók (pl. tervek), végül egy üzenet tájékoztatja a felhasználót.

Profil megtekintése és szerkesztése

A **Profilom** menüpont megnyitásakor a felhasználó egy dedikált modal felületen látja a saját adatait. Az adatokat pedig tudja szerkeszteni ha a későbbiekben akár a jelszavát vagy a felhasználó nevét is meg akarja változtatni, akkor itt lehetséges az is.

A felhasználó módosíthatja az e-mail címét, teljes nevét, a kedvezmény típusát, és a jelszavát.

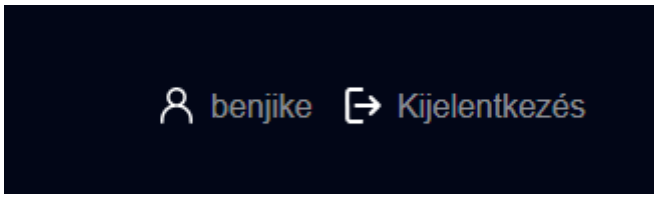


A felhasználó bármely adatát módosíthatja. A jelszó megadása opcionális, csak akkor változik, ha a felhasználó új jelszót ad meg.

Mentés után a rendszer visszajelzést ad a sikeres módosításról, biztonsági okokból kijelentkezteti a felhasználót. Ez növeli a rendszer biztonságát és átláthatóságát.

Kijelentkezés

A kijelentkezés bármikor elérhető, egyetlen gombnyomással történik, megnyomás után az alkalmazás visszatér nem bejelentkezett állapotba, azaz a főoldalra és a “Terveim” menüpont már nem lesz elérhető számára úgy mint amikor be van jelentkezve a felhasználó.



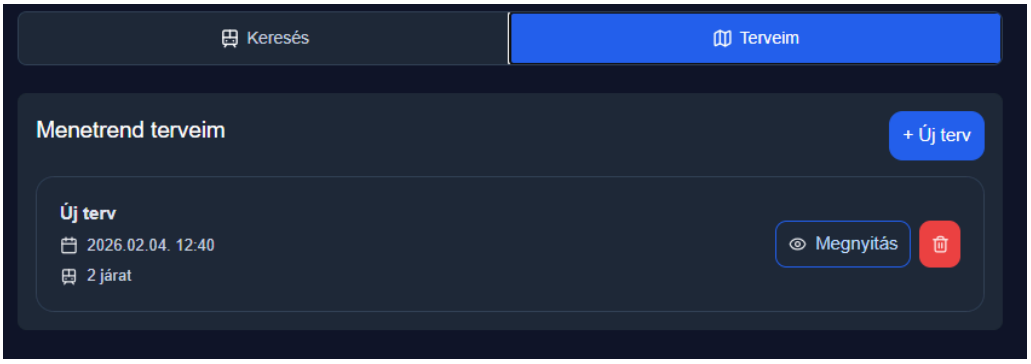
Menetrendtervek modul

A modul célja, hogy a felhasználó összegyűjthesse a számára releváns járatokat, több különálló utazási tervet hozhasson létre (pl. “hétvégi kirándulás”), és az egyes utazásokat áttekinthető, strukturált formában kezelhesse, nem egyszerű keresési találatokként.

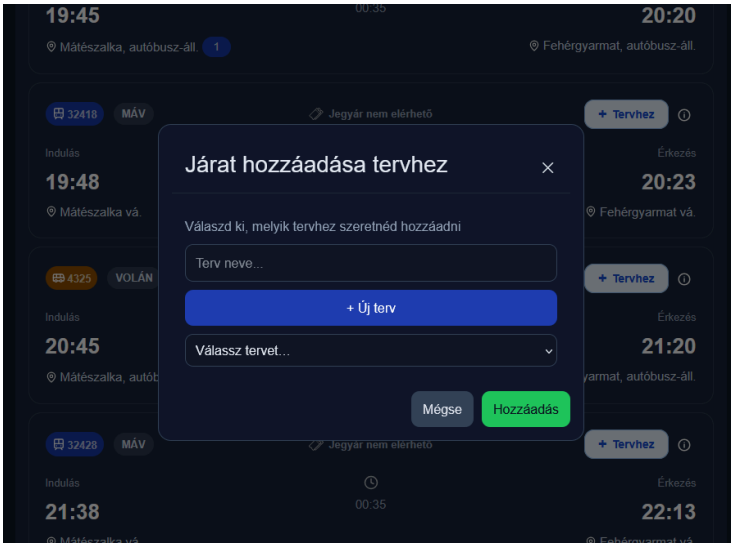
A funkció megoldást nyújt arra a problémára, hogy összetett, átszállásos útvonalaknál a felhasználónak ne kelljen külön jegyzetelnie az indulásokat, érkezéseket és járatszámokat.

Új terv létrehozása

A felhasználó új tervet hozhat létre a **„+ Új terv”** gombra kattintva. Egy modal ablakban a terv nevét kell megadnia. A létrehozás után a terv azonnal megjelenik a listában,
járathoz adáskor automatikusan kiválasztható az újonnan létrehozott terv.



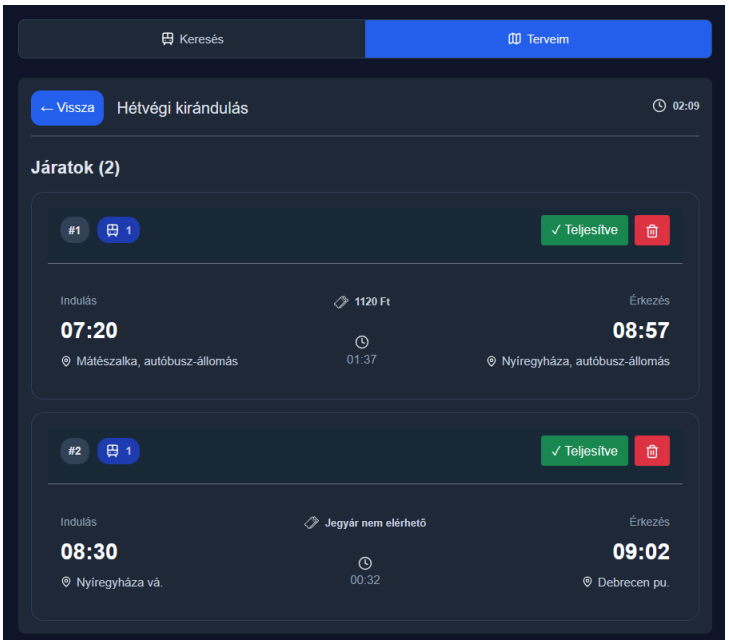
Azonban a főoldalon is létrehozhat a felhasználó új tervet egy járat keresésekor. Így nem szükséges hogy átmenjen a “Terveim” ablakra hogy majd ott hozza létre azt. Ezzel is egyszerűsítve van a felhasználónak a weboldal.



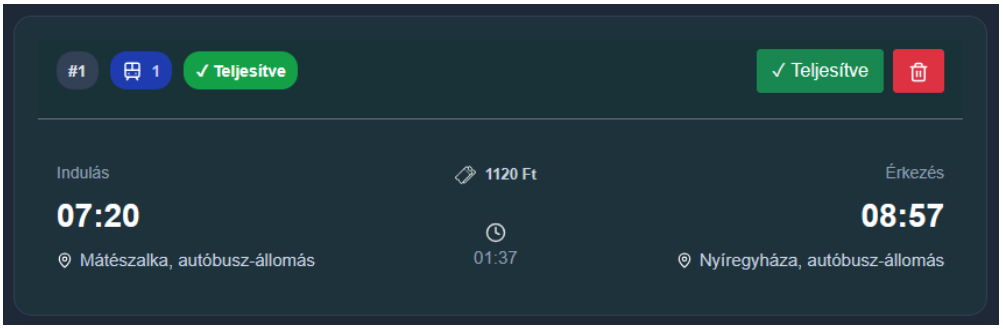
Terv részleteinek megtekintése

A **Megnyitás** gombra kattintva megjelenik a terv részletes nézete, leírással és adatok megadásával. A terv neve felül található, az összes járat megfelelő sorrendben jelenik meg, és minden járat külön kártyán jelenik meg.

A megjelenítés vizuálisan megegyezik a főoldalon található kereső logikájával, így a felhasználónak nem kell új felületet megtanulnia. És ugyanúgy könnyen és egyszerűen tudja majd ezt kezelni.



A járatok állapotát kezelheti a felhasználó, ez hasznos lehet több szakaszos, időérzékeny utazásoknál.



A **menetrendtervek modul** az alkalmazás egyik legfontosabb hozzáadott értéke.
Nem csupán egy mentési funkció, hanem egy **komplex utazásszervező eszköz**, amely hidat képez a keresés és a valós utazás között, támogatja az átszállásos, időérzékeny útvonalakat, és valódi problémára ad használható megoldást.

Az applikáció technológiai háttere

Frontend

Angular: Az Angular egy fejlesztő platform, amely a TypeScript-re épül.

Platformként az Angular tartalmazza a komponens alapú keretrendszert, ami kiváló skálázható webes alkalmazások készítéséhez, jól integrált könyvtárak gyűjteményét, amelyek számos funkciót lefednek, beleértve az űrlapkezelést, a kliens-szerver kommunikációt és egyebeket, valamint a fejlesztői eszközök csomagját, amely segít a kód fejlesztésében, felépítésében, tesztelésében és frissítésében.

Az Angular-ral egy olyan platform előnyeit élvezheted, amely az egyéni fejlesztői projektektől a vállalati szintű alkalmazásokig skálázható. A legjobb az egészben, hogy az Angular ökoszisztéma több mint 1,7 millió fejlesztőből, könyvtárszerzőből és tartalomkészítőből álló sokszínű csoportból áll.

A legtöbb modern keretrendszerhez hasonlóan az Angular is elvárja a HTML, CSS és JavaScript ismeretét.

Eszközei közé tartozik a **TypeScript**, ami alapértelmezés szerint minden Angular alkalmazással együtt érkezik, hogy jobb eszközöket és jobb karbantarthatóságot biztosítson a jobb fejlesztői élmény érdekében, és a **parancssori felület (CLI)**, ezt használja a keretrendszer az eszközök összetettségének absztraktálására és a kód optimalizálására, így az alkalmazás fejlesztésére egyszerűbben koncentrálhatsz.

Komponensek

A komponensek az Angularban történő alkalmazások létrehozásának alapvető építőkövei. A komponens-architektúra kihasználásával az Angular célja, hogy struktúrát biztosítson a projekt kezelhető, jól szervezett részekre szervezéséhez, egyértelmű felelősségi körökkel, így a kód karbantartható és skálázható.

Service-ek (szolgáltatások)

Az Angularban a service-ek (szolgáltatások) olyan osztályok, amelyek az alkalmazás üzleti logikáját és adatkezelését valósítják meg, elkülönítve a felhasználói felületért felelős komponensektől.

A service-ek célja, hogy a komponensek egyszerűek, átláthatók és könnyen karbantarthatók maradjanak. Ezek mellet megteremtik a kliens és a szerver közötti kommunikációt valósítja meg, adatokat dolgoz fel és alakít át, megvalósítja az üzleti logikát, valamint újrahasznosítható funkciókat biztosít több komponens számára.

A service-ek egyik legnagyobb előnye az újrahasznosíthatóság. Egyetlen service több különböző komponens által is használható, így elkerülhető a kódduplikáció, és egységes logika valósítható meg az alkalmazás különböző részein.

Kliens–szerver kommunikáció

Az Angular service-ek gyakran használják az Angular által biztosított *HttpClient* modult, amely lehetővé teszi a backend felé adott kérések küldését, valamint az aszinkron adatkezelést. a HTTP hívások általában *Observable* objektumokat adnak vissza, amelyek az RxJS könyvtárra épülnek. Ez lehetővé teszi az aszinkron műveletek hatékony kezelését, a hibakezelést, és az adatfolyamok kombinálását, feldolgozását.

Összegzés

Az Angular alkalmazásokban tehát a következő felelősségmegosztás figyelhető meg:

Komponensek: A megjelenítésért és a felhasználói interakciók kezeléséért felelősek.
Service-ek: Az üzleti logikát és az adatkezelést valósítják meg.

Modellek

A modellek az Angular alkalmazásban az adatok struktúráját írják le.
Segítségükkel egyértelműen meghatározható, hogy egy adott adatobjektum milyen mezőket és milyen típusú értékeket tartalmazhat.

A modellek típusbiztonságot biztosít TypeScript segítségével, azaz a fejlesztés során, még a kód futtatása előtt ellenőrzi, hogy az adatok típusa megfelelő-e, és megfelelően használjuk-e őket. Emelett egységes adatstruktúrát képez a komponensek és service-ek között, növeli a kód érthetőségét, mivel az adatok jelentése és felépítése egyértelmű. Egy modell általában egy *interface*, vagy *class (osztály)* formájában van megvalósítva, ami csak adatleírást tartalmaz, üzleti logikát nem.

A modellek használata különösen fontos akkor, amikor az alkalmazás API-kkal kommunikál, mivel segítenek az érkező adatok helyes kezelésében és az esetleges hibák korai felismerésében.

Az alkalmazás felépítése (frontend)

Az alkalmazás Angular keretrendszerben készült, amely lehetővé teszi a logikailag elkülönített, jól karbantartható és skálázható felépítést. A projekt struktúrája három fő pillérre épül: **komponensek**, **service-ek** és **modellek**. Ez a felosztás támogatja az elkülönített felelősségi körök elvét (separation of concerns), amely kulcsfontosságú nagyobb alkalmazások esetén.

Komponensek

A komponensek felelősek az alkalmazás megjelenítéséért és felhasználói interakcióinak kezeléséért. A projektben a komponensek funkcionális csoportok szerint, a *features* mappában helyezkednek el.

Főbb komponenscsoportok:

“user” komponens: Felhasználói fiókkezeléshez kapcsolódó komponensek (bejelentkezés, regisztráció, profil megtekintése és módosítása).

“search” komponens: Menetrend- és járatkereséshez kapcsolódó komponensek (keresőfelület, találatok listázása, járatkártyák).

“plan” komponens: Menetrendtervek kezelése (tervek listázása, terv részleteinek megjelenítése, járatok hozzáadása tervhez).

“misc” komponens: Egyéb, újrahasznosítható vagy segéd komponensek (pl. értesítések, modális ablakok).

A komponensek fő feladatai az adatok megjelenítése HTML sablon segítségével, felhasználói események kezelése (kattintás, űrlapkitöltés), valamint service-eken keresztül adatok lekérése vagy módosítása az adatbázisban.

Service-ek

A service-ek az alkalmazás üzleti logikáját és adatkezelését valósítják meg. Ezek a *services* mappában találhatók, és több komponens által is újrahasznosíthatók. A szolgáltatások célja a kliens-szerver kommunikáció kezelése, központi állapot vagy adatmegosztás biztosítása, valamint a komponensek tehermentesítése az üzleti logikától.

A projektben használt főbb service-ek:

UserService: Felhasználói műveletek kezelése (bejelentkezés, regisztráció, kijelentkezés, profiladatok frissítése).

SearchService: Menetrend- és járatkereséssel kapcsolatos műveletek (útvonalak lekérése, késési információk kezelése).

PlanService: Tervek és tervhez tartozó járatok kezelése (tervek listázása, létrehozása, járatok hozzáadása és törlése).

AlertService: Felhasználói visszajelzések és értesítések kezelése (sikeres vagy hibás műveletek megjelenítése).

Ez a megközelítés elősegíti a kód újrafelhasználhatóságát, valamint a könnyebb tesztelhetőséget.

Modellek

A modellek az alkalmazásban használt adatstruktúrák egységes leírására szolgálnak.
TypeScript *interface-ek* formájában jelennek meg, amelyek típusbiztonságot és jobb kódolvashatóságot biztosítanak.

A projektben a modellek két fő területhez kapcsolódnak:

“models” modell: Egy állomás struktúrája (pl. megálló, település), és egy járat struktúrája (indulás, érkezés, időtartam, jegyár, szolgáltató).

“plan” modell: Egy terv struktúrája (id, név, létrehozás dátuma), tervhez tartozó járatkapcsolatok struktúrája.

A modellek használatának előnyei az egységes adatkezelés az egész alkalmazásban, kevesebb futásidejű hiba, valamint a fejlesztés közbeni automatikus ellenőrzés és kódkiegészítés.

Források:

- Angular dokumentáció (https://v17.angular.io/guide)
- Angular (web framework) (https://en.wikipedia.org/wiki/Angular_(web_framework))
- Adam Freeman - Pro Angular 16 (https://www.manning.com/books/pro-angular-16)

Backend

Node.js: A Node.js egy nyílt forráskódú, szerveroldali futtatókörnyezet, amely a Google által fejlesztett *V8 JavaScript motort* használja. Segítségével JavaScript nyelven lehet szerveroldali alkalmazásokat fejleszteni, ami lehetővé teszi az egységes nyelvhasználatot a frontend és a backend között.

A Node.js legfontosabb jellemzői között van az eseményvezérelt működés, amely hatékonyan kezeli a párhuzamos kéréseket. Könnyen bővíthető a környezet az *npm (Node Package Manager)* csomagkezelő segítségével, mellyel egyszerűen telepíthetünk különböző kiegészítéseket a szerverünkhöz. Gyors fejlesztési ciklussal rendelkezik és széles körű közösségi támogatás van mögötte.

A Node.js különösen alkalmas olyan alkalmazásokhoz, ahol sok kliens kérés érkezik egyidejűleg, gyakori az adatlekérés és adatküldés, valamint fontos a gyors válaszidő és a skálázhatóság.

Ezek miatt a Node.js széles körben elterjedt modern webes és API-alapú rendszerek fejlesztésében.

REST API (Representational State Transfer)

A **REST API** egy architekturális stílus, amely meghatározza, hogyan kommunikál egymással a kliens és a szerver HTTP protokollon keresztül.

A REST API legfontosabb jellemzői között szerepel a **kliens-szerver architektúra,** azaz a frontend és backend egymástól független, a **stateless működés,** tehát minden kérés önálló, nem támaszkodik korábbi állapotra, az és az **erőforrás-alapú megközelítés,** ami az adatbázisban szereplő adatok erőforrásokként való megjelenését biztosítja.

A leggyakrabban használt HTTP metódusok:

GET: adatok lekérése, fogadása az adatbázisból
POST: új adatok létrehozása, feltötlése az adatbázisba
PUT: meglévő adatok módosítása (*update*) az adatbázisban
DELETE: adatok törlése az adatbázisból

A REST API általában JSON formátumban cserél adatokat, amely könnyen feldolgozható mind kliens-, mind szerveroldalon.

Node.js és REST API kapcsolata

A Node.js gyakran választott platform REST API-k készítésére, mivel hatékonyan kezeli a HTTP kéréseket, jól illeszkedik az aszinkron adatfeldolgozáshoz, és könnyen integrálható különböző adatforrásokkal és külső szolgáltatásokkal.

Egy Node.js alapú REST API lehetővé teszi, hogy a frontend alkalmazás strukturált módon, jól definiált végpontokon keresztül érje el az adatokat, miközben a backend biztosítja az üzleti logika és az adatfeldolgozás egységességét.

Az alkalmazás felépítése (backend)

Az alkalmazás szerveroldali része Node.js környezetben futó Express alapú REST API-ként készült. A backend feladata egyrészt a kliens (Angular frontend) kiszolgálása adatokkal (felhasználói műveletek, tervek kezelése), másrészt a külső menetrendi adatforrás (Menetrendek/MÁV adatbázis-API) „proxyzása” és egységes, frontenden könnyen kezelhető válaszformátum biztosítása.

Technológiai felépítés

A HTTP végpontok, útvonalak (route-ok) definiálását és a kérések kezelését az *Express* biztosítja. A *bodyParser* segítségével a kliens JSON törzsű kérései könnyen elérhetővé válnak a szerver számára. A frontend és backend eltérő domainről/tárolóról futtatása esetén szükséges, hogy a böngésző engedélyezze a cross-origin kéréseket, azaz lehetővé teszi, hogy egy weboldal biztonságosan adatokat kérjen le egy másik domainről, portról vagy protokollról - ehhez a *CORS* csomagot használtuk. Az API URL és adatbázis paraméterek a környezetből érkeznek, amit a *dotenv* csomag tölt be induláskor.

Külső menetrendi adatforrás integráció

A backend több végpontja valójában egy külső menetrendi szolgáltatás felé küld kérést (*MÁV API-ja*), majd visszaadja az eredményt a frontendnek. Ennek előnyei, hogy a frontend nem

közvetlenül kommunikál a külső API-val (kevesebb konfiguráció, egységesítés), ezek mellett pedig a backend egységesíti a paraméterezést (pl. dátum/óra/perc stringgé alakítás).

A külső hívásokhoz *fetch* jellegű Fetch API kompatibilis megoldás van használva (dinamikus importtal).

Menetrendi végpontok

Állomás keresés (*POST /api/searchStation*): Célunk az állomás/megálló keresése szöveg alapján.. A backend feladata az, hogy a keresőkifejezést becsomagolja a külső API által elvárt payloadba, majd visszaadja a kapott listát.

Járatok keresése (*POST /api/searchRoutesCustom*): A végpont a felhasználó dátum + idő beállításai szerint ad vissza járatokat.

Járat részletei / megállók listája (*POST /api/runDescription*): Célunk egy konkrét járat megállóinak és időadatainak lekérdezése (részletező nézethez).

Késés lekérés (POST /api/runsDelay): Több járatot kap meg listaként, ezek késésének mértékét pedig visszaadja. Válaszként a külső API eredménytömbje kerül visszaadásra (csak a ténylegesen szükséges adatrésszel).

Fiókkezelés a backendben

A jelszavak nem „plain text”-ként tárolódnak, hanem *bcrypt hash* formában titkosításra kerülnek.

Regisztráció *(POST /api/register):* A backend a kapott jelszót bcrypt-tel hash-eli (titkosított formára alakítja át) és csak a *hash* kerül eltárolásra.

Bejelentkezés *(POST /api/login):* A backend felhasználónév alapján lekéri a felhasználót, majd ellenőrzi a jelszót. Fontos, hogy a válaszból a jelszó *hash* törlésre kerül, így a frontend sosem kapja meg.

Profil módosítás (***PUT /api/user/:felhasznalonev*):** A felhasználó email-je, teljes neve és kedvezményének mértéke is frissíthető és módosítható. A jelszó csak akkor frissül, ha a felhasználó változtatni kíván rajta: ekkor új titkosítás készül.

4) Tervek funkció backend oldalon

A „Tervek” modul egy mentett menetrendgyűjteményt valósít meg: a felhasználó több tervet hozhat létre, egy tervhez pedig több járat rendelhető sorrenddel.

Tervkezelés

Tervek listázása (*GET /api/plans/:felhasznalonev*): A felhasználóhoz tartozó tervek kilistázása.

Terv létrehozása (*POST /api/addPlan*): A backend visszaadja az új terv azonosítóját az adatbázisból, hogy a frontend azonnal tudjon hozzá járatot kötni.

Terv törlése (*DELETE /api/planDelete/:tervId*): A terv azonosítója alapján törlésre kerül az adatbázisból.

Járatok tervhez rendelése

Egy terv járatai (*GET /api/planRoutes/:tervId*): Visszaadja a tervhez kapcsolt járatokat sorrend szerint az adatbázisból a megfelelő *SQL* kérések segítségével.

Járat létrehozása (*POST /api/addRoute*): A backend a létrehozott járat azonosítóját visszaküldi, hogy a frontend rögtön hozzá tudja adni a kapcsolótáblában.

Kapcsolat létrehozása (terv_jarat) (*POST /api/addPlanRoute*): Feltölti a backend az összeköttetést a *terv* és a *jarat* táblák közötti kapcsolótáblába.

Kapcsolat törlése (*DELETE /api/planRouteDelete/:jaratId*): A járat azonosítója alapján az összeköttetés törlésre kerül a kapcsolótáblából.

Státusz frissítése (*PUT /api/routeStatus/:jaratId*): A járat azonosítója lapján frissül az adott járat státusza (elérte/még nem érte el az adott járatot a felhasználó).

Források:

- Node.js Foundation (https://nodejs.org/en/about)
- Mozilla Developer Network: REST API: (https://developer.mozilla.org/en-US/docs/Glossary/REST)
- Fielding, R. T. - Architectural Styles and the Design of Network-based Software Architectures
- IBM Documentation - REST APIs (https://www.ibm.com/docs/en/integration-bus/10.0?topic=overview-rest-apis)

Verziókezelés

GitHub: A GitHub egy webalapú platform, amely a *Git* verziókezelő rendszerre épül.

Elsődleges célja a forráskód változásainak nyomon követése, a fejlesztési folyamat átlátható kezelése, valamint az együttműködés támogatása egyéni és csapatmunkában egyaránt. A *Git* elosztott verziókezelő rendszer, ami azt jelenti, hogy minden fejlesztő a teljes projekt történetét tartalmazó helyi másolattal dolgozik, nem egy központi szerverre támaszkodva.

Verziókezelés szerepe a szoftverfejlesztésben

A verziókezelés lehetővé teszi a forráskód változásainak visszakövetését, az alkalmazás korábbi állapotának visszaállítását, párhuzamos fejlesztési ágak (*branch-ek*) létrehozását, a hibák és új funkciók elkülönített fejlesztését, valamint a fejlesztési folyamat dokumentálását *commit* üzeneteken keresztül, ezzel könnyen visszakövethető a fejlesztési folyamat.

Ez különösen fontos egy összetettebb webalkalmazás esetén, ahol a frontend és a backend párhuzamosan fejlődik, és gyakoriak az iteratív módosítások.

A GitHub használata a projektben

Forráskód verziókezelése

A teljes frontend és backend kódbázis egy GitHub repository-ban került tárolásra. Minden jelentősebb módosítás külön *commitban* lett rögzítve, ami biztosította, hogy a fejlesztési lépések visszakövethetők legyenek, egy hibás változtatás esetén könnyen vissza lehessen térni egy stabil állapothoz, és a kód fejlődése időrendben dokumentálva legyen.

Funkciók fokozatos fejlesztése

A GitHub segítségével jól elkülöníthetőek voltak az egyes funkcionális egységek fejlesztési fázisai, mint például a keresési funkciók (állomás- és járatkeresés), tervek (menetrendtervek) kezelése, fiókkezelés (regisztráció, bejelentkezés, profil módosítás), valamint a frontend és backend tesztek hozzáadása.

Forrás: GitHub dokumentáció (https://docs.github.com/en/get-started/using-git/about-git)

Adatbázis

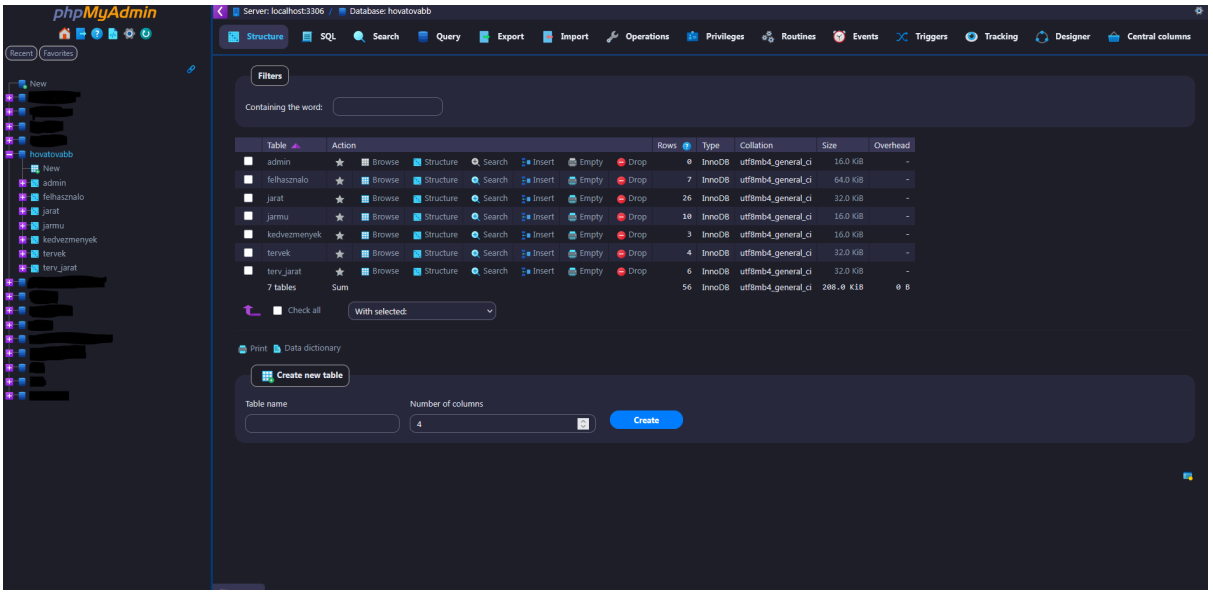
MySQL: A MySQL egy nyílt forráskódú, relációs adatbázis-kezelő rendszer, amely a *Structured Query Language (SQL)* nyelvet használja az adatok kezelésére. A relációs modell lényege, hogy az adatokat táblákban tárolja, ahol a táblák sorokból (rekordokból) és oszlopokból (mezőkből) állnak, és a táblák között kulcsok segítségével lehet kapcsolatokat létrehozni.

A MySQL egyik legfontosabb előnye a megbízhatóság és a teljesítmény. Széles körben alkalmazzák webes alkalmazások háttérrendszereként, mivel jól skálázható, stabil működést biztosít, és könnyen integrálható különböző programozási nyelvekkel, például JavaScript (Node.js), PHP vagy Java környezetben.

A MySQL előnyéhez tartozik az is, hogy platformfüggetlen, tehát futtatható akár Windows, Linux, vagy macOS alapú operációs rendszereken is. Gyakran használatos olyan alkalmazásokban, ahol strukturált adatok hosszú távú tárolására és hatékony lekérdezésére van szükség, például felhasználói adatok, üzleti adatok vagy alkalmazásállapotok kezelésére.

phpMyAdmin mint adminisztrációs eszköz

A phpMyAdmin egy webalapú grafikus felület, amely MySQL és MariaDB adatbázisok kezelésére szolgál. Célja, hogy megkönnyítse az adatbázis-adminisztrációs feladatokat azok számára is, akik nem kizárólag parancssoros SQL-környezetben szeretnének dolgozni.



A phpMyAdmin böngészőn keresztül érhető el, és lehetőséget biztosít az adatbázisok teljes körű kezelésére, adatbázisok és táblák hozhatunk létre, módosíthatunk és törölhetünk, adatokat szúrhatunk be, frissíthetünk és törölhetünk, *SQL* lekérdezéseket futtathatunk, valamint adatbázis-mentéseket és visszaállításokat készíthetünk egy grafikus felületen.

A phpMyAdmin előnye, hogy vizuálisan átláthatóvá teszi az adatbázis szerkezetét, és gyors hozzáférést biztosít az adatokhoz. Ez különösen hasznos fejlesztési és tesztelési környezetben, ahol gyakran szükség van az adatok ellenőrzésére vagy manuális módosítására.

MySQL és phpMyAdmin kapcsolata

A phpMyAdmin nem önálló adatbázis-kezelő rendszer, hanem egy adminisztrációs felület, amely a MySQL adatbázisszerverhez csatlakozva működik. A MySQL végzi az adatok tényleges tárolását és feldolgozását, míg a phpMyAdmin a felhasználó és az adatbázis közötti kényelmes grafikus interfészt (*GUI*) biztosítja.

Az alkalmazás felépítése (adatbázis)

Az alkalmazás adatkezeléséhez MySQL relációs adatbázist használtunk, amely stabil, széles körben elterjedt és jól integrálható Node.js alapú backenddel. Az adatbázis kezelésére és vizuális adminisztrációjára a phpMyAdmin eszközt alkalmaztuk, amely megkönnyítette az adatstruktúra tervezését, az adatok ellenőrzését, valamint a fejlesztés és hibakeresés folyamatát.

Adatbázis-tervezés és normalizálás

Az adatbázis felépítése relációs elveken alapul, az adatok logikailag elkülönített táblákban kerülnek tárolásra. A struktúra célja az volt, hogy elkerüljük az adatismétlést, biztosítsuk az adatok konzisztenciáját, átláthatóságát, valamint támogassuk az alkalmazás fő funkcióit (*keresés, tervezés, felhasználókezelés*).

A táblák közötti kapcsolatokat idegen kulcsokkal (*foreign key*) valósítottuk meg, így az adatbázis képes biztosítani az adatok közötti integritást.

Felhasználók és jogosultságok kezelése

A *felhasznalo* tábla tartalmazza az alkalmazás regisztrált felhasználóinak adatait. Itt kerül tárolásra a felhasználónév, az email cím, a teljes név, valamint a jelszó biztonságos, hash-elt formában.

A felhasználókhoz kapcsolódik egy kedvezmény azonosító, amely a kedvezmenyek táblára mutat. Ez lehetővé teszi, hogy az alkalmazás a különböző jegyár-kedvezményeket (*pl. 50%-os kedvezmény*) egységes módon kezelje.

Tervek és járatok kapcsolata

Az alkalmazás egyik központi funkciója az egyéni menetrendtervek létrehozása és kezelése. Ennek megfelelően az adatbázisban külön táblák szolgálnak a tervek és a járatok kezelésére:

“tervek” tábla: Egy adott felhasználó által létrehozott menetrendterv alapadatait tartalmazza *(azonosító, felhasználónév*).

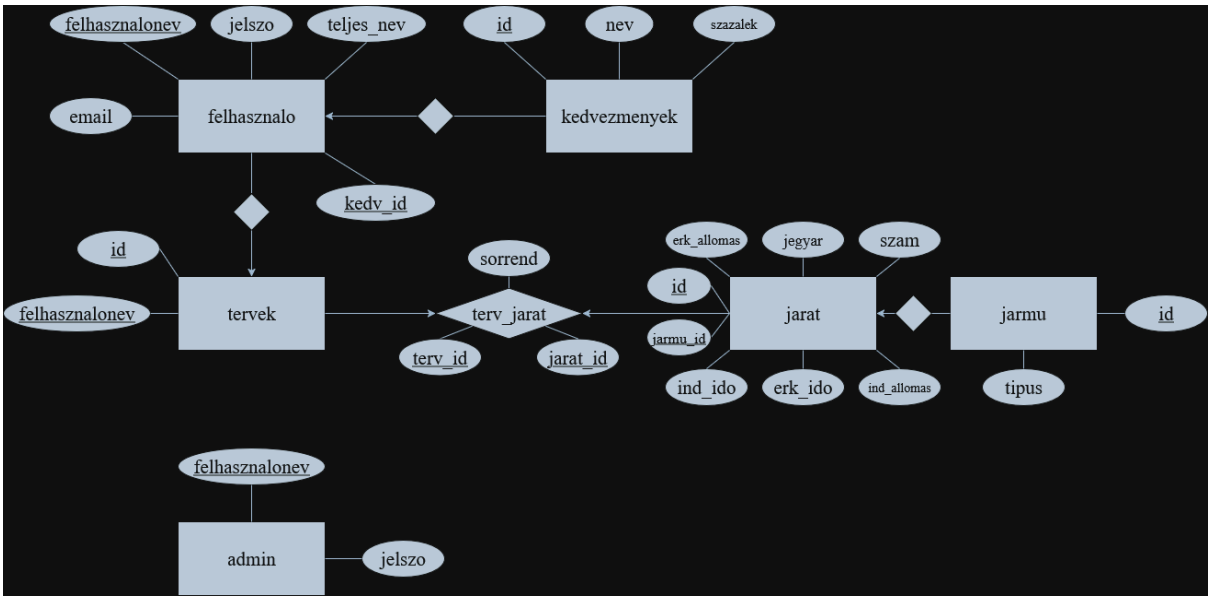
“jarat” tábla: Egy konkrét utazási szakasz adatait tárolja (*indulási és érkezési állomás, időpontok, menetidő, jegyár, járműtípu*s).

“terv_jarat” kapcsolótábla: Összeköti a terveket a hozzájuk tartozó járatokkal.

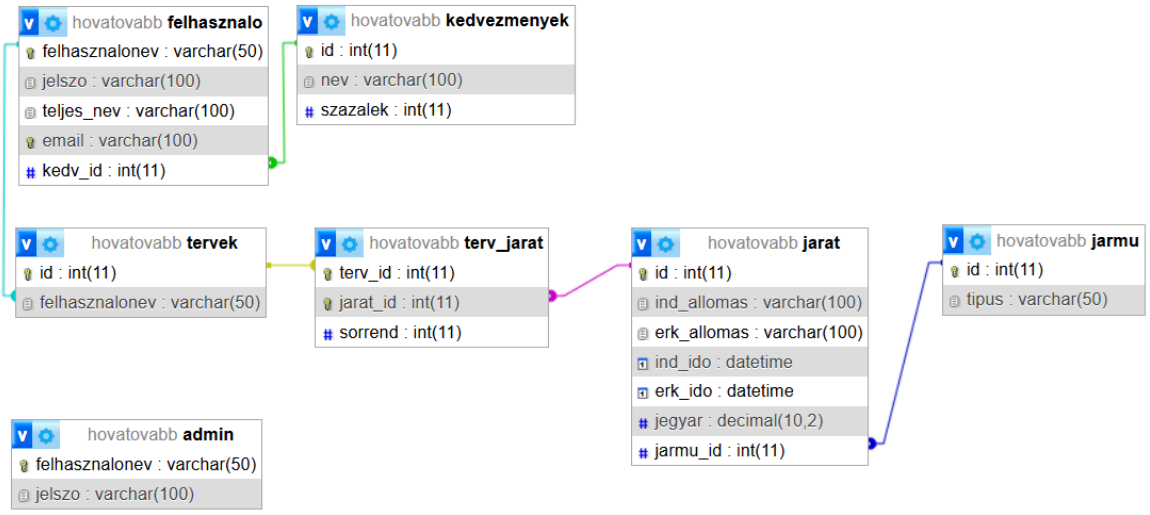
A **terv_jarat** egy kapcsolótábla, melynek használata lehetővé teszi, hogy egy terv több járatból álljon, a járatok sorrendje rögzíthető legyen, ezek mellett pedig az adatbázis megfeleljen a több-több kapcsolat elvének.

Járművek kezelése

A járatokhoz tartozó közlekedési eszközöket külön *jarmu* tábla írja le. Ez a megoldás biztosítja, hogy a járműtípusok (*pl. busz, vonat*) egységesen és redundancia nélkül legyenek kezelve, valamint lehetőséget ad későbbi bővítésre (*például új járműtípusok hozzáadására*).



ER-modell (összefüggések, kapcsolatok)



UML-modell (adatbázis felépítése)

phpMyAdmin szerepe a fejlesztés során

A phpMyAdmin felületet az adatbázis fejlesztése és karbantartása során intenzíven használtuk. Előny volt számunkra, hogy vizuális áttekintést adott az adatbázis-struktúráról és a kapcsolatokról, megkönnyítette az SQL lekérdezések tesztelését, segített az adatok manuális ellenőrzésében (pl. tervekhez tartozó járatok helyes mentése), valamint támogatta a hibakeresést, amikor az alkalmazás működése és az adatbázis állapota közötti eltéréseket kellett vizsgálni.

Források:

- MySQL dokumentáció (https://dev.mysql.com/doc/)
- MySQL Community Edition (https://www.mysql.com/products/community/)
- phpMyAdmin dokumentáció (https://www.phpmyadmin.net/docs/)

Szoftvertesztelés

A szoftvertesztelés a fejlesztési folyamat egyik legfontosabb minőségbiztosítási eleme. Célja annak ellenőrzése, hogy az elkészült rendszer megfelel-e a funkcionális és nem funkcionális követelményeknek, hibamentesen működik-e, valamint stabil és megbízható-e különböző környezetekben.

A tesztelés főbb céljai közé tartoznak a hibák korai felismerése, a rendszer stabilitásának biztosítása, és a regressziók (korábban működő funkciók elromlása) elkerülése.

A projekt során a *frontend* és a *backend* réteg külön keretrendszerrel került tesztelésre:

Frontend: Jasmine (*Angular környezetben*).
Backend: Jest (*Node.js környezetben, szimulált mock adatbázissal*).

A két réteg külön tesztelése lehetővé tette az üzleti logika és a kommunikációs rétegek elkülönített, kontrollált ellenőrzését.

Frontend

Az Angular keretrendszer alapértelmezett tesztelési eszköze a *Jasmine*, amely egy viselkedésvezérelt (BDD: Behavior Driven Development) tesztelő keretrendszer. A Jasmine az Angular CLI környezetbe integráltan működik, és a Karma futtatókörnyezet segítségével jeleníti meg az eredményeket.

Tesztelt frontend funkciók

A frontend oldalon az alábbi komponenseket és service-eket teszteltük:

PlanService teszt

A tesztelt funkció a menetrendtervek hívásáért felelős service volt, azaz a tervek a backenden keresztül az adatbázisból való lehívásának tesztje.

Ellenőrzött viselkedések között szerepelt az, hogy a service megfelelő *HTTP GET* kérést küld, a válaszból helyesen adja vissza a tervek tömbjét, valamint az, hogy a visszakapott adatok (pl. terv neve, felhasználónév) helyesek-e.

Eredményképpen a teszt sikeresen lefutott, a HTTP kérés URL-je és metódusa megfelelő volt, ezek mellett pedig a backend-ből kapott válasz feldolgozása is helyes volt.

SearchService teszt

Tesztelt funkciónk a menetrendkeresésért felelős service volt. Ellenőriztük, hogy a service megfelelő HTTP kérést, ebben az esetben POST kérést küld-e a backend felé, a backend válaszából tömböt ad-e vissza, és a válasz megfelelően feldolgozásra kerül-e.

A teszt sikeres volt, a metódus helyes HTTP konfigurációval működött, és a válasz tömbként került visszaadásra.

PlanComponent teszt

A menetrendtervek hívásáért felelős komponenst teszteltük, azt vizsgáltuk, hogy a komponens meghívja-e a *getPlans()* metódust a PlanService-ből, ami lényegében a tervek betöltéséért felelős metódus, valamint, hogy a visszakapott adat bekerül-e a *plans* tömbbe, ami a tervek tárolásáért felelős adattároló.

Tesztünk során a komponens megfelelően kommunikált a service-szel, és a tervek betöltése automatikusan megtörtént az adatbázisból lehívott tervek inicializáláskor.

SearchComponent teszt

Tesztelt funkciónk a menetrendkeresésért felelős komponens volt. Azt ellenőriztük, hogy meghívja-e a komponens a járatok keresésért felelős *searchRoutesCustom()* metódust a SearchService-ből, nem indít-e késésszámítást, és a journeys tömb feltöltődik-e.

Eredményképpen a teszt sikeresen lefutott, a keresési logika megfelelően működött, a késésszámítás opcionálisan kezelhető volt, valamint a komponens állapotkezelése stabil volt.

Integrációs teszt

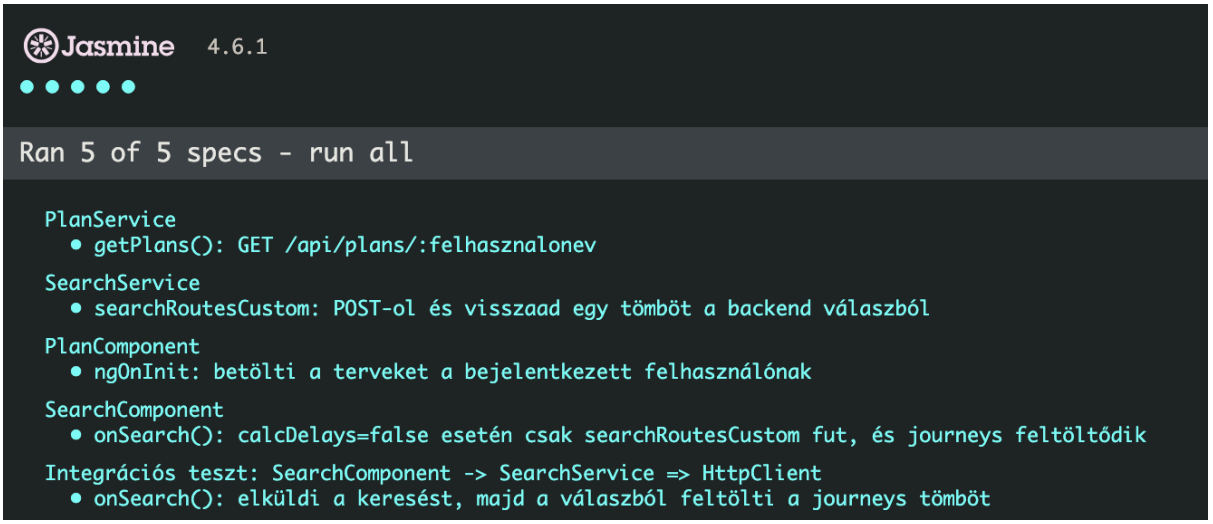
Tesztelt folyamat: **SearchComponent → SearchService → HttpClient**

Azt vizsgáltuk, hogy a járatok kereséséért felelős komponens elindítja-e a keresést, majd elküldi-e a service-nek, ezt követően pedig a service elküldi-e a HTTP kérést. Valamint ezután azt is vizsgáltuk, hogy a kapott válasz alapján feltöltődik-e a járatok tárolásáért felelős *journeys* tömb.

A teszt sikeres volt, a komponens és service közti integráció megfelelő volt, a teljes keresési folyamat validált volt.

Frontend tesztelés összegzése

A frontend oldalon öt specifikáció került futtatásra, minden teszt sikeresen lefutott, a kritikus funkciók (*keresés, tervek, API kommunikáció*) pedig lefedettek. Ez biztosítja, hogy a felhasználói felület és az üzleti logika stabil és megbízható.



Frontend tesztek eredménye

Backend

A backend oldalon a *Jest* tesztelő keretrendszert használtuk, amely modern JavaScript környezetben széles körben alkalmazott eszköz.

A tesztek futtatása során szimulált *mock adatbázist* használtunk, külső API hívásokat mockoltunk, valamint HTTP kéréseket *Supertest* segítségével validáltunk.

Tesztelt backend funkciók

Állomáskeresés teszt

Tesztelt végpont: *POST /api/searchStation*

Azt vizsgáltuk, hogy a végpont 200-as státuszkódot ad-e vissza (tehát hibátlanul lefut), a külső menetrendi API megfelelően meghívásra kerül-e, valamint, hogy a válasz JSON formátumban érkezik-e.

A teszt során a végpont sikeresen működött, a külső API hívás mock környezetben is validáltak voltak.

Járatkeresés teszt

Tesztelt végpont: *POST /api/searchRoutesCustom*

Ellenőriztük, hogy a backend megfelelő HTTP kérést, ebben az esetben POST kérést küld-e a menetrendi API felé, a válasz továbbításra kerül-e a frontend irányába, és hiba esetén 500-as státuszkódot ad-e.

A teszt sikeres volt, a megfelelő HTTP kérést küldte el a backend, ezek mellett pedig a hibakezelés megfelelően működött.

Bejelentkezés teszt

Tesztelt végpont: *POST /api/login*

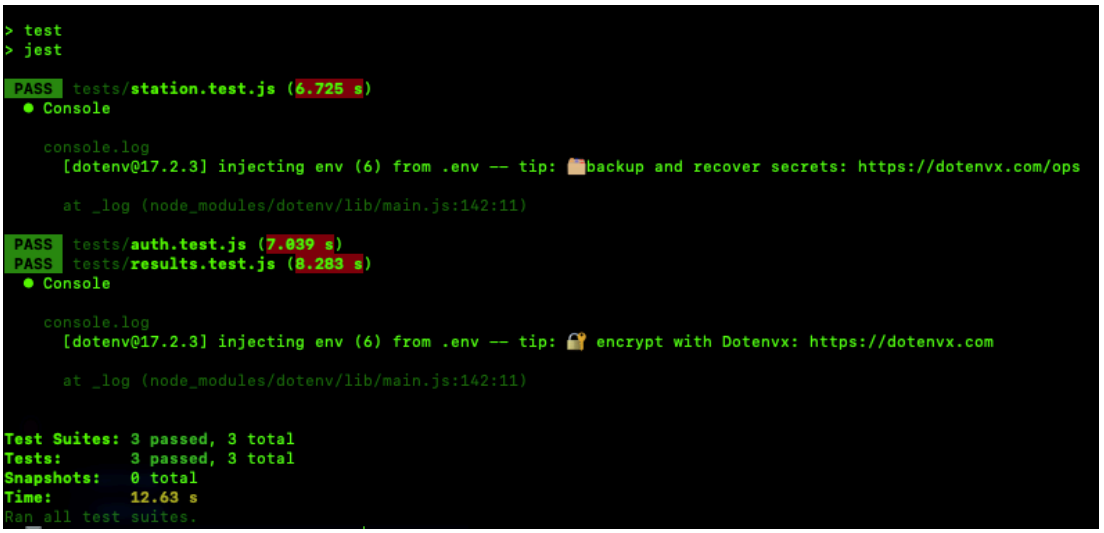
Megvizsgáltuk, hogy a backend helyes adatok fogadása esetén 200-as státuszkódot ad-e vissza, hibás adatok esetén 401-es státuszkódot ad-e vissza (hiányos belépési adatok hibája), valamint hogy működik-e a jelszó titkosítása.

Eredményképpen a hitelesítési mechanizmus stabil volt, valamint a jelszó titkosítása és ellenőrzése helyesen került validálásra.

Backend tesztelés összegzése

A backend oldalon három tesztfájl futott le sikeresen három különböző tesztkörnyezetben.

A tesztfutás igazolta a REST API végpontok helyes működését, a külső API integráció stabilitását (MÁV adatbázis), a hitelesítési mechanizmus megbízhatóságát, valamint a hibakezelési logika helyességét



Backend tesztek eredménye

Források:

- Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.
- Jasmine dokumentáció (https://jasmine.github.io/)
- Angular tesztelés dokumentáció (https://angular.io/guide/testing)
- Jest dokumentáció (https://jestjs.io/docs/getting-started)

Mobilfejlesztés

React: A React egy nyílt forráskódú JavaScript könyvtár, amelyet a Meta (korábban Facebook) fejlesztett ki felhasználói felületek létrehozására.

Elsődleges célja az interaktív, állapotvezérelt felületek hatékony és újrahasznosítható komponensek segítségével történő megvalósítása. A React deklaratív megközelítést alkalmaz, amely lehetővé teszi, hogy a fejlesztő az alkalmazás aktuális állapotát írja le, míg a megjelenítés részleteit a keretrendszer automatikusan kezeli.

A React filozófiájának központi eleme a komponens-alapú architektúra (*mint az Angular-ban*), amely elősegíti a kód modularitását, karbantarthatóságát és skálázhatóságát. A komponensek önálló egységek, amelyek saját állapottal és logikával rendelkezhetnek, és egymással jól definiált interfészeken keresztül kommunikálnak.

Virtuális DOM és teljesítmény

A React egyik legfontosabb technikai újítása a virtuális DOM (*Document Object Model*) használata. A virtuális DOM egy könnyű, memóriában tárolt reprezentációja a valódi DOM-nak.

Ez a megközelítés jelentősen csökkenti a DOM-manipulációk számát, ami különösen fontos a mobil eszközök korlátozott erőforrásai mellett, ahol a teljesítmény és az energiahatékonyság kiemelt szempont.

React Native

A React technológia nem kizárólag webes környezetben alkalmazható, hanem meghatározó szerepet tölt be a mobilalkalmazás-fejlesztésben is. Ennek egyik legismertebb példája a **React Native**, amely a React alapelveire építve teszi lehetővé natív mobilalkalmazások fejlesztését Android és iOS platformokra.

A React mobilfejlesztési szempontból rendkívülien előnyös, hiszen támogatja a **kódújrahasznosítást**, azaz a komponenslogika és az állapotkezelés nagy része közös lehet webes és mobilos környezetben. **Deklaratív UI-fejlesztést** biztosít, tehát a felhasználói felület állapotvezérelt módon épül fel, ami csökkenti a hibalehetőségeket. **Komponens-alapú**

struktúrát használ, így a felület kisebb, jól elkülöníthető részekre bontható, ami megkönnyíti a karbantartást. **Nagy közösségi támogatást** biztosít, széles körű dokumentációval rendelkezik, valamint könyvtárak és eszközök állnak rendelkezésre.

Az alkalmazás felépítése (mobil)

A **HovaTovább Lite** mobilalkalmazásunk a fő webes rendszer kiegészítő változataként készült. A teljes funkcionalitású alkalmazás Angular keretrendszerben készült, és progresszív webalkalmazásként működik, így mobileszközökön is teljes értékűen használható. A progresszív webalkalmazás lehetővé teszi a telepíthetőséget, valamint a mobilbarát megjelenítést.

Ennek ellenére a tervezési szempontok alapján indokolt volt egy külön, natív jellegű mobilalkalmazás létrehozása. Sok felhasználó számára ugyanis nem lehet szükséges a teljes menetrendtervező rendszer, elegendő csupán a gyors és egyszerű menetrendkeresés. A HovaTovább Lite célja éppen ezért egy letisztult, minimalista mobilalkalmazás biztosítása, amely kizárólag a menetrendkeresési funkcióra koncentrál.

Technológiai alapok

A mobilalkalmazás **React Native** és **Expo** környezetben készült. A React Native lehetővé teszi, hogy egyetlen kódbázisból natív Android és iOS alkalmazás készüljön. Az Expo keretrendszer egyszerűsíti a fejlesztési folyamatot, biztosítja a natív funkciók elérését és megkönnyíti az alkalmazás tesztelését fizikai eszközökön.

Az alkalmazás TypeScript nyelven íródott, amely típusbiztonságot és jobb karbantarthatóságot biztosít. A grafikus elemek React komponensekből épülnek fel, az ikonok megjelenítéséhez pedig SVG alapú megoldás került alkalmazásra, amely skálázható, vektorizálható, és ezek mellett vizuálisan konzisztens megjelenést biztosít különböző állított képernyőméreteken.

A felhasználói felület sötét témára épül, amely modern megjelenést és jobb vizuális kontrasztot biztosít mobil környezetben.

Működés

Az alkalmazás moduláris szerkezetű. A főbb egységek külön komponensekbe és logikai rétegekbe szervezve találhatók meg.

A felhasználói felületet komponensek alkotják. Ilyen például az **állomáskereső mező**, a **járatkártya**, az **átszállási szakasz megjelenítő elem** és a részletes **járatinformációt tartalmazó modális ablak.** Ezek a komponensek felelősek a vizuális megjelenítésért és a felhasználói interakció kezeléséért.

Az üzleti logika külön fájlokban található. A menetrendi adatok lekérése egy külön API rétegen keresztül történik, amely a backend REST végpontjait hívja meg. Ez a réteg felelős az adatok feldolgozásáért, az esetleges hibák kezeléséért és az API válaszok egységes formátumba alakításáért.

A modellek TypeScript interfészek formájában jelennek meg, amelyek meghatározzák az állomások és a járatok adatstruktúráját. Ez biztosítja, hogy a komponensek egységes, típusellenőrzött adatokat kapjanak.

A menetrendkeresés működése

A felhasználó indulási és érkezési állomást választhat ki, valamint beállíthatja az utazás dátumát és időpontját. A keresés indításakor az alkalmazás a backend szolgáltatáson keresztül továbbítja a paramétereket a menetrendi rendszer felé.

A backend a külső menetrendi szolgáltatótól lekéri a találatokat, majd azokat visszaküldi a mobilalkalmazás számára. Az alkalmazás ezután feldolgozza a kapott adatokat, kiszűri a releváns járatokat.

A járatkártyák megjelenítik az indulási és érkezési időt, az állomásneveket, az utazás teljes időtartamát, valamint az esetleges késési információkat. Átszállás esetén az alkalmazás jelzi a szakaszok számát, és lehetőséget biztosít az egyes szakaszok részletes megtekintésére.

Részletes járatinformáció

A felhasználó a járatkártyán található információ gomb segítségével megnyithat egy modális ablakot, amely az adott járat részletes megállólistáját tartalmazza. Több átszállásos útvonal esetén a szakaszok között külön választható elemek jelennek meg, amelyek lehetővé teszik az egyes járatrészek külön megtekintését.

A modális ablak táblázatos formában jeleníti meg a megálló nevét, az érkezési és indulási időt. Amennyiben rendelkezésre áll valós idejű adat, az alkalmazás a várható időpontot jeleníti meg, ellenkező esetben a tervezett menetrendi időt.

Felhasználói élmény és tervezési szempontok

A mobilalkalmazás tervezése során elsődleges szempont volt az egyszerűség és az átláthatóság.

A vizuális kialakítás konzisztens, az ikonok egységes stílusúak, a tipográfia jól olvasható. A felület mobilképernyőre optimalizált, az érintési felületek megfelelő méretűek, és az interakciók azonnali visszajelzést adnak.

A hibakezelés is beépítésre került, így amennyiben nincs találat, az alkalmazás nem omlik össze, hanem egyértelmű visszajelzést ad a felhasználónak.

Főképernyő

A mobilalkalmazás főképernyője az *index.tsx* fájlban került megvalósításra, amely az alkalmazás központi vezérlőrétegeként működik. Ez a fájl felelős a felhasználói interakciók kezeléséért, az állapotkezelésért, valamint a menetrendi adatok lekéréséért és megjelenítéséért.

A főképernyő tartalmazza az indulási és érkezési állomás kiválasztására szolgáló mezőket, a dátum- és időválasztást, valamint a keresés indításához szükséges vezérlőelemeket. Itt történik a keresési paraméterek összeállítása és a backend API meghívása. A keresési eredmények beérkezése után az *index.tsx* fájl felel azok feldolgozásáért, az esetleges hibakezelésért, valamint a találatok listába rendezett megjelenítéséért.

Az állapotkezelés React hookok segítségével történik, azaz speciális horgokat használunk az adatok tárolására, frissítésére és a felhasználói felület (*UI*) automatikus újrarajzolására. A kiválasztott állomások, a dátum, az idő, a betöltési állapot és az esetleges hibaüzenetek mind a komponens lokális állapotában kerülnek tárolásra. A főképernyő koordinálja a komponensek közötti adatáramlást is, például a kiválasztott járat adatainak továbbadását az információs modális ablak számára.

A főképernyő kialakítása mobilhasználatra optimalizált, egyszerű és átlátható struktúrával rendelkezik. A felület célja, hogy a lehető legkevesebb lépéssel biztosítsa a menetrendkeresés lehetőségét.

Komponensek

A mobilalkalmazás több újrafelhasználható komponensből épül fel, amelyek külön felelősségi körrel rendelkeznek.

“JourneyCard” komponens: A keresési találatok egy-egy elemének megjelenítéséért felel. Megjeleníti az indulási és érkezési időt, az állomásneveket, az utazás teljes időtartamát, valamint az esetleges késési információkat. Több szakaszból álló utazás esetén jelzi az átszállások számát is. A komponens tartalmazza az információs gombot, amely megnyitja a részletes megállólistát tartalmazó modális ablakot.

“JourneyInfoModal” komponens: A kiválasztott járat részletes adatait jeleníti meg. A backend szolgáltatás segítségével lekéri a megállók listáját, majd táblázatos formában megjeleníti az érkezési és indulási időpontokat. Több átszállás esetén lehetőséget biztosít a szakaszok közötti váltásra, így minden járatszegmens külön-külön megtekinthető.

“StationInput” komponens: Az állomáskeresési funkció megvalósításáért felel. A felhasználó által beírt szöveg alapján API-hívást indít a backend felé, majd a találatokat listában jeleníti meg. A kiválasztott állomás visszakerül a főképernyő állapotába.

“TransportChip” komponens: A járattípus és a szolgáltató vizuális megjelenítésére szolgál. Ikonok és színek segítségével különbözteti meg a különböző közlekedési módokat, valamint több szakasz esetén vizuálisan jelzi az átszállásokat.

Ezen komponensek együtt biztosítják a felület modularitását, átláthatóságát és karbantarthatóságát. A komponensalapú felépítés lehetővé teszi az egyes részek önálló fejlesztését és tesztelését.

Modellek

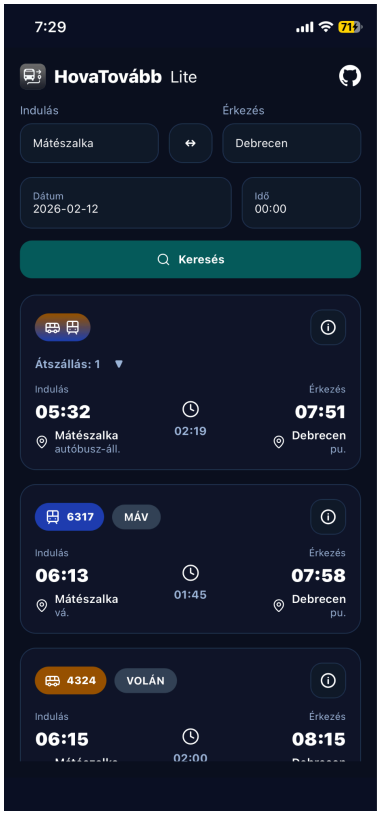
A mobilalkalmazás három fő adatmodellt használ az adatszerkezetek egységes kezelésére.

“Station” modell: Az állomások adatait írja le. Tartalmazza az állomás nevét, azonosítóját, településazonosítóját és egyéb, a kereséshez szükséges paramétereket. Ez a modell biztosítja, hogy az állomáskeresés és a járatkeresés során egységes formában kerüljenek továbbításra az adatok.

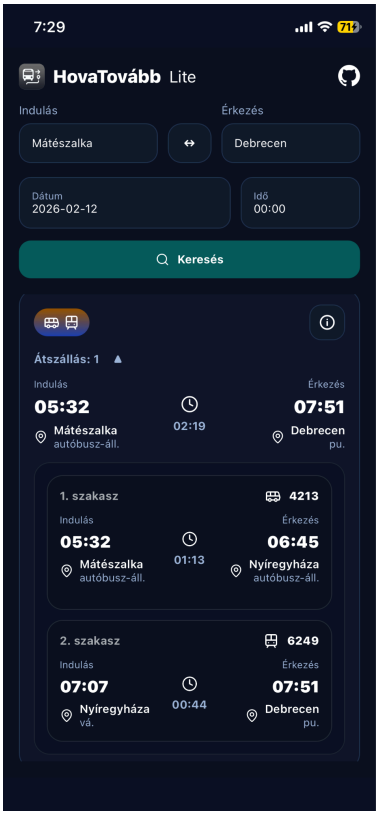
“JourneySegment” modell: Egy utazás egyetlen szakaszát reprezentálja. Tartalmazza többek között a járatszámot, a szolgáltató nevét, az indulási és érkezési időt, valamint a megállókhoz kapcsolódó azonosítókat. Ez a modell teszi lehetővé az átszállásos utak strukturált kezelését.

“Journey” modell: Egy teljes utazást ír le, amely egy vagy több *JourneySegment* elemből állhat. Tartalmazza az összes járat adatait, az utazás teljes időtartamát, valamint az esetleges késési információkat. A *Journey* modell biztosítja a keresési találatok egységes reprezentációját, és alapját képezi a *JourneyCard* és a *JourneyInfoModal* komponensek működésének.

A modellek alkalmazása hozzájárul az alkalmazás típusbiztonságához, az adatok konzisztens kezeléséhez és a fejlesztés során fellépő hibák csökkentéséhez.



Keresési találatok



Átszállások kezelése

Egy utazás részletei

Források:

- React hivatalos dokumentáció (https://react.dev/)
- React Native hivatalos dokumentáció (https://reactnative.dev/)
- Meta Open Source: React (https://opensource.fb.com/projects/react/)
- Abramov, D. - The Virtual DOM Explained

A HovaTovább továbbfejleszthetősége

A HovaTovább magyar tömegközlekedési menetrendtervező alkalmazás már jelenlegi formájában is stabil technológiai alapokra épül, azonban a rendszer architektúrája és funkcionalitása számos irányban továbbfejleszthető. A modern webes technológiák alkalmazása (Angular frontend, Node.js backend, MySQL adatbázis) lehetővé teszi a skálázhatóságot, a moduláris bővítést és a hosszú távú fenntarthatóságot. A következőkben részletesen bemutatjuk a lehetséges fejlesztési irányokat technológiai, funkcionális és üzleti szempontból.

Funkcionális továbbfejlesztési lehetőségek

A HovaTovább alkalmazás jelenlegi funkcionalitása stabil alapot biztosít a jövőbeni bővítésekhez, különösen a felhasználói élmény növelése terén. Az egyik legjelentősebb fejlesztési irány a közösségi funkciók bevezetése lehet. A rendszer lehetőséget adhat arra, hogy a felhasználók saját összeállított menetrendjeiket megosszák másokkal, kedvenc útvonalaikat publikussá tegyék, valamint visszajelzést adjanak egyes járatok megbízhatóságáról vagy kényelméről. Egy ilyen közösségi alapú működés nemcsak növelné az alkalmazás használati értékét, hanem kiegészítené a hivatalos adatforrásokat valós idejű, felhasználói tapasztalatokon alapuló információkkal is. Ez különösen hasznos lehet rendkívüli események, zsúfoltság vagy váratlan járatkimaradások esetén.

A személyre szabhatóság további erősítése szintén fontos fejlesztési irány. Egy intelligens értesítési rendszer képes lehet figyelni a felhasználó által rendszeresen használt útvonalakat, és automatikusan értesítést küldeni késésekről, menetrendi változásokról vagy forgalmi fennakadásokról. Hosszabb távon mesterséges intelligenciával támogatott elemző rendszer is integrálható, amely a korábbi keresések és utazási szokások alapján javaslatokat tesz alternatív útvonalakra. Ez nemcsak kényelmi funkció, hanem jelentős időmegtakarítást is eredményezhet a mindennapi közlekedés során.

További funkcionális bővítési lehetőséget jelent a multimodális közlekedési forma integrálása. A jelenlegi tömegközlekedési fókusz mellett a rendszerbe beépíthető lenne kerékpáros útvonaltervezés, elektromos roller szolgáltatások kezelése, gyalogos szakaszok optimalizálása, valamint akár autómegosztó rendszerek csatlakoztatása is. A modern városi

közlekedés egyre inkább több közlekedési eszköz kombinációjára épül, így egy ilyen fejlesztés jelentősen növelné az alkalmazás versenyképességét és komplexitását.

Technológiai fejlesztési irányok

A projekt technológiai háttere lehetővé teszi a rendszer skálázható továbbfejlesztését. Nagyobb felhasználószám esetén szükségessé válhat felhőalapú infrastruktúra bevezetése, amely biztosítja a folyamatos rendelkezésre állást és a megfelelő teljesítményt. A backend architektúra mikroszolgáltatás-alapú megközelítéssel tovább optimalizálható, amely rugalmasabb fejlesztést és könnyebb karbantarthatóságot eredményez. A konténerizált környezet alkalmazása szintén hozzájárulhat a stabil működéshez és a gyors telepítéshez.

Az adatkezelés optimalizálása kiemelt fontosságú a valós idejű információk kezelése miatt. A jövőben indokolt lehet gyorsítótárazási (cache) mechanizmusok bevezetése, adatbázis-replikáció alkalmazása, illetve nagy mennyiségű valós idejű adat esetén alternatív adatbázis-megoldások vizsgálata. Ezek a fejlesztések csökkenthetik a válaszidőt és növelhetik a rendszer megbízhatóságát.

Mobilplatform és offline működés

A HovaTovább jelenleg webalapú alkalmazásként működik, azonban hosszú távon indokolt lehet natív mobilalkalmazás fejlesztése Android és iOS platformra. Egy dedikált mobilalkalmazás gyorsabb működést, hatékonyabb push értesítéseket és pontosabb helyadat-kezelést biztosíthat. Emellett lehetővé válhat részleges offline működés is, amely során a gyakran használt menetrendi adatok gyorsítótárazásra kerülnek, így internetkapcsolat hiányában is elérhetők maradnak. Ez különösen hasznos lehet olyan területeken, ahol a mobilinternet-lefedettség nem megfelelő.

Mesterséges intelligencia és prediktív elemzés

A projekt egyik leginnovatívabb jövőbeni fejlesztési iránya a mesterséges intelligencia alkalmazása lehet. A historikus adatok elemzésével előre jelezhetők lennének a késések, forgalmi torlódások vagy csúcsidőszakok. A rendszer képes lehet dinamikusan optimalizálni az útvonaltervezést a korábbi minták alapján, illetve személyre szabott ajánlásokat nyújtani a

felhasználók számára. Az ilyen prediktív funkciók jelentős versenyelőnyt biztosíthatnak más menetrendtervező alkalmazásokkal szemben.

Földrajzi és üzleti bővítési lehetőségek

Hosszú távon a projekt földrajzi értelemben is bővíthető. A magyarországi menetrendek mellett integrálhatók lennének regionális vagy akár nemzetközi járatok is, valamint bevezethető lenne a többnyelvű támogatás. Ez új felhasználói rétegek bevonását tenné lehetővé, és növelné az alkalmazás piaci értékét.

A fenntartható működés érdekében üzleti modell kialakítása is szükséges lehet. Ez magában foglalhat prémium funkciókat, partnerségeket közlekedési szolgáltatókkal, vagy akár integrált jegyvásárlási rendszer bevezetését. Egy stabil pénzügyi háttér biztosítaná a folyamatos fejlesztést és az infrastruktúra fenntartását.

Összegzés

Összességében megállapítható, hogy a HovaTovább projekt jelentős továbbfejlesztési potenciállal rendelkezik. A rendszer jelenlegi technológiai alapjai lehetővé teszik a fokozatos, tervezett és fenntartható bővítést. Megfelelő fejlesztési stratégiával az alkalmazás nem csupán egy iskolai projekt maradhat, hanem egy országos, sőt akár nemzetközi szinten is versenyképes, modern menetrendtervező rendszerré válhat.

Munkamegosztás

A fejlesztési folyamat nem egyszerre, hanem több egymásra épülő szakaszban történt. A projekt elején a legfontosabb cél az alkalmazás architektúrájának megtervezése volt. Ekkor közösen meghatározásra került az adatszerkezet, a kommunikáció módja a kliens és a szerver között, valamint az, hogy a rendszer REST alapú API-ra fog épülni. Ez az előkészítési szakasz körülbelül egy hetet vett igénybe, mivel ekkor dőlt el a teljes alkalmazás felépítése.

Backend

A fejlesztés első tényleges szakaszát a backend kialakítása jelentette. Ennek során készült el a *Node.js* alapú szerver, amely a menetrendi adatok külső szolgáltatásból történő lekérését és feldolgozását végzi. Ebben az időszakban a fő feladat az API végpontok megtervezése és implementálása volt, például az állomáskeresés, járatkeresés, valamint a késésinformációk kezelése. Emellett ki kellett alakítani az adatbázis szerkezetét és a mentési logikát a tervek kezeléséhez. A backend alapfunkcióinak elkészítése körülbelül egy hónapot vett igénybe, amelyet további finomítások és hibajavítások követtek a későbbi fázisokban.

Frontend

Ezzel párhuzamosan indult el a webes felület fejlesztése Angular keretrendszerben. A kliensoldali munka során először a keresőfelület készült el, majd a találatok megjelenítése, végül a tervkezelési funkciók. A frontend fejlesztés körülbelül két hónapig tartott, mivel itt nemcsak a működés, hanem a felhasználói élmény és az átlátható megjelenés is fontos szerepet kapott. A backend és frontend ebben az időszakban folyamatosan együtt fejlődött, az API szerkezete több alkalommal módosult a könnyebb használhatóság érdekében.

Mobilalkalmazás

A webalkalmazás elkészülte után készült el a mobilalkalmazás *React Native* környezetben. Ennek célja nem a teljes funkcionalitás átvétele volt, hanem egy egyszerű, gyorsan használható menetrendkereső biztosítása. A mobilalkalmazás fejlesztése körülbelül másfél hetet vett igénybe, mivel a már meglévő API-ra épült, azonban külön felületet és kezelési logikát kellett kialakítani az érintőkijelzős használathoz.

Tesztelés

A projekt utolsó szakaszát a tesztelés és finomhangolás jelentette. Ekkor történt a frontend és backend tesztek elkészítése, a hibák javítása, valamint a felhasználói visszajelzések alapján a kezelőfelület módosítása. Ez a szakasz nagyjából egy hétig tartott, de kisebb tesztelések a teljes fejlesztési folyamat során folyamatosan történtek.

Összegzés

A munkamegosztás a fejlesztők között természetesen alakult ki: a szerveroldali logika és az adatkezelés főként backend oldalon készült, míg a felhasználói felület és a mobilalkalmazás inkább kliensoldali fókuszt kapott. Ugyanakkor minden nagyobb funkció közösen került megtervezésre, és a hibakeresés is együtt történt, így mindkét fél átlátta a teljes rendszer működését.

Összességében a projekt fejlesztése iteratív módon zajlott: a backend alapokra épült a webes alkalmazás, majd arra a mobilalkalmazás, végül a rendszer tesztelése és optimalizálása zárta a fejlesztési folyamatot.

SWOT analízis

Az alkalmazás értékeléséhez SWOT analízist alkalmaztunk, amely a rendszer erősségeit, gyengeségeit, lehetőségeit és veszélyeit vizsgálja.

Az elemzés célja annak bemutatása, hogy a HovaTovább rendszer milyen pozíciót foglal el a hasonló menetrendi szolgáltatások között, valamint milyen irányokban fejleszthető tovább.

Erősségek

Az alkalmazás egyik legnagyobb előnye az egységes keresés biztosítása különböző közlekedési szolgáltatók járatai között. A felhasználónak nem szükséges különböző menetrendi oldalakat vagy alkalmazásokat használni, egyetlen felületen keresztül képes teljes útvonalakat tervezni. Ez jelentősen javítja a felhasználói élményt és csökkenti a kereséshez szükséges időt.

Fontos erősség továbbá a közösségi szemléletű tervkezelés. A felhasználók saját utazási terveket hozhatnak létre és menthetnek el, így a rendszer nem csupán egyszeri keresésekre alkalmas, hanem hosszabb távon is használható személyes útvonalkezelőként.

A késésinformációk integrálása szintén növeli a hasznosságot, mivel valós idejű döntéshozatalt tesz lehetővé.

Technikai szempontból előnyt jelent a platformfüggetlen működés. A PWA alapú webalkalmazás mobileszközön is teljes értékűen használható, míg a külön mobilalkalmazás egyszerűbb, gyorsabb hozzáférést biztosít azok számára, akik kizárólag a keresési funkciót igénylik.

A moduláris felépítés és a REST API architektúra lehetővé teszi a későbbi bővítést.

Gyengeségek

A rendszer külső menetrendi szolgáltatásra támaszkodik, ezért működése erősen függ az elérhető API stabilitásától és válaszidejétől. Ha a külső szolgáltatás lassú vagy nem elérhető, az közvetlenül rontja a felhasználói élményt.

A menetrendi adatok struktúrája nem minden esetben egységes, ezért az adatok feldolgozása több helyen külön logikát igényel. Ez növeli a kód komplexitását és a hibalehetőségek számát. Továbbá a tervek kezelése jelenleg alapvető funkciókat biztosít, de nem tartalmaz fejlettebb szervezési lehetőségeket, például megosztást vagy automatikus frissítést.

A mobilalkalmazás funkcionalitása szándékosan korlátozott, ami egyszerűséget ad, ugyanakkor csökkenti az ott elérhető szolgáltatások körét.

Lehetőségek

A rendszer továbbfejleszthető közösségi funkciókkal, például tervek megosztásával vagy ajánlott útvonalak megjelenítésével. Lehetőség van értesítések bevezetésére is, amely figyelmezteti a felhasználót a közelgő indulásra vagy jelentős késésre.

A későbbiekben további közlekedési szolgáltatók integrálása növelheti a lefedettséget. A mobilalkalmazás funkcióinak bővítése, például offline tárolással vagy widget támogatással szintén növelheti a használhatóságot.

Az alkalmazás alkalmas lehet adatgyűjtésre is, amely anonim módon elemezheti az utazási szokásokat, és optimalizált ajánlásokat adhat a felhasználóknak.

Veszélyek

A legnagyobb kockázatot a külső menetrendi szolgáltatások változása jelenti. Az API megszűnése, módosítása vagy korlátozása az alkalmazás működését jelentősen befolyásolhatja. Emellett egy hivatalos szolgáltató saját, fejlettebb alkalmazása konkurenciát jelenthet.

Biztonsági szempontból kockázatot jelent a felhasználói adatok kezelése, ezért az adatvédelem és a megfelelő jogosultságkezelés folyamatos figyelmet igényel. Továbbá a valós idejű adatok pontatlansága a felhasználók bizalmát csökkentheti.

Statisztikai adatok

A projekt lezárását követően egy rövid felhasználói elégedettségi kérdőívet készítettünk, amelynek célja az volt, hogy visszajelzést kapjunk az alkalmazás használhatóságáról és gyakorlati értékéről. A beérkezett válaszok közül három kérdést emeltünk ki részletesebb értékelésre, mivel ezek mutatták meg leginkább a rendszer erősségeit és hiányosságait.

Az alkalmazás kezelőfelülete mennyire átlátható?

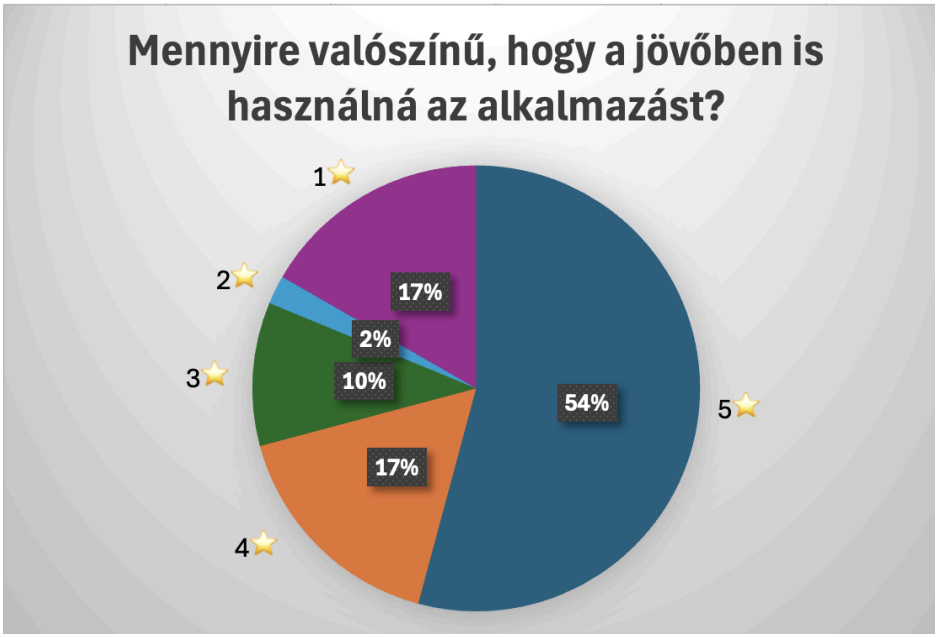
A válaszok alapján a felhasználók többsége kifejezetten átláthatónak találta az alkalmazás felületét, a legmagasabb értékelés aránya kiemelkedő volt. Ugyanakkor megjelent egy jelentősebb, közepes értékelést adó csoport is. Ennek oka feltehetően a mobilos megjelenítés sajátossága: kisebb kijelzőn, sok adat egyidejű megjelenítésekor bizonyos esetekben az információk egymáshoz közel kerülnek, ami zsúfoltabb érzetet kelthet. Ez nem használhatósági hibát, inkább ergonómiai korlátot jelent, amely a nagy mennyiségű menetrendi adat természetéből fakad. Összességében a felület logikája érthetőnek bizonyult, de mobilkijelzőn egyes helyzetekben vizuálisan terheltebb lehet.

Mennyire tartja hasznosnak a menetrendterv funkciót?

A menetrendterv funkció értékelése rendkívül pozitív képet mutatott. A válaszadók döntő többsége maximális értékelést adott, ami egyértelműen jelzi, hogy a funkció valódi problémára nyújt megoldást. Mindössze egy nagyon kis arány adott közömbös értékelést, ami arra utal, hogy gyakorlatilag senki nem tartotta feleslegesnek. A négycsillagos értékelések nagy valószínűséggel olyan felhasználóktól származnak, akik hasznosnak tartják ugyan a lehetőséget, de a saját utazási szokásaik miatt ritkán van rá szükségük. A funkció tehát nemcsak technikailag működőképes, hanem a felhasználók szemszögéből is releváns.

Mennyire valószínű, hogy a jövőben is használná az alkalmazást?

A legmegosztóbb kérdés az alkalmazás jövőbeni használatára vonatkozott. Bár itt is a legmagasabb értékelés dominált, az alacsonyabb értékek nagyobb arányban jelentek meg, mint a többi kérdésnél. Ez arra utal, hogy a felhasználók többsége szívesen használná a rendszert a jövőben, azonban a használat gyakorisága erősen függ az egyéni közlekedési szokásoktól. Akik ritkán utaznak tömegközlekedéssel, kevésbé érzik szükségesnek az alkalmazás rendszeres használatát, míg az aktív utazók számára kifejezetten hasznos eszköz lehet. A válaszok tehát nem a rendszer minőségét kérdőjelezik meg, hanem a célcsoport eltérő élethelyzeteit tükrözik.



Összegzés

Az eredmények alapján az alkalmazás használhatósága és funkcionalitása kedvező megítélés alá esett. A kezelőfelület jól érthetőnek bizonyult, a menetrendterv funkció egyértelműen értéket képvisel a felhasználók számára, és a többség a jövőben is szívesen alkalmazná a rendszert. A megosztóbb válaszok elsősorban nem a rendszer hibáiból, hanem véleményünk szerint a különböző felhasználási igényekből és a mobilkijelző fizikai korlátaiból adódtak. Ez azt jelzi, hogy az alkalmazás megfelelő alapokra épül, és elsősorban finomhangolással tovább javítható a felhasználói élmény.

A projekt személyes és szakmai hozadéka, valamint jövőbeli elvárásaink

A HovaTovább rendszer fejlesztése nem csupán egy programozási feladat volt, hanem egy komplex, több területet érintő mérnöki munka, amely során a tervezéstől a megvalósításon át az üzemeltetésig a teljes fejlesztési folyamatban részt vettünk. A projekt során megszerzett tapasztalatok jelentősen hozzájárultak szakmai fejlődésünkhöz, valamint reális képet adtak egy valódi szoftvertermék életciklusáról.

Személyes fejlődés

A fejlesztés során megtapasztaltuk, hogy egy alkalmazás elkészítése nem kizárólag a kód megírásából áll. A követelmények pontosítása, a felhasználói élmény megtervezése, az adatmodellezés, a hibakezelés és a karbantarthatóság mind olyan tényezők, amelyek legalább annyira fontosak, mint maga a programozás.

Megtanultuk, hogy egy rosszul átgondolt struktúra később jelentős átalakításokat eredményez, ezért a tervezési fázisra fordított idő hosszú távon megtérül.

A projekt során fejlődött problémamegoldó képességünk is. Több alkalommal találkoztunk olyan hibákkal, amelyek nem dokumentált külső rendszerekből, eltérő adatformátumokból vagy váratlan felhasználói viselkedésből adódtak. Ezek megoldása során nem kész megoldásokat követtünk, hanem elemzéssel, naplózással és teszteléssel jutottunk el a működő implementációig. Ez jelentősen növelte önálló munkavégzési képességünket.

Szakmai tapasztalat

A rendszer fejlesztése során gyakorlati tapasztalatot szereztünk a kliens–szerver architektúrában működő alkalmazások felépítéséről. Megismertük, hogyan kommunikál egymással a frontend, a backend és az adatbázis, valamint hogyan kell egy külső szolgáltatást biztonságosan és megbízhatóan integrálni.

A projekt során különösen fontossá vált a tiszta kód és a modularitás. A komponensek és szolgáltatások szétválasztása nemcsak az átláthatóságot növelte, hanem a hibakeresést és a bővíthetőséget is megkönnyítette.

A tesztelés bevezetése szintén meghatározó tapasztalat volt, mert rávilágított arra, hogy egy működő program és egy megbízható program nem ugyanaz.

Megtanultuk továbbá az üzemeltetés alapjait is: környezetváltozók használatát, adatbázis karbantartást, naplózást és hibakezelést. Ez a rész különösen fontos volt, mert itt szembesültünk azzal, hogy a szoftver valójában a felhasználóknál kezd el „élni”, és ekkor jelennek meg az addig nem tapasztalt problémák.

Mit várunk a projekttől a jövőben

A HovaTovább alkalmazást nem egyszeri iskolai feladatként tekintjük, hanem egy folyamatosan fejleszthető rendszerként. Rövid távon cél a stabil működés fenntartása, a hibák javítása és a felhasználói visszajelzések feldolgozása. A projekt jelenlegi állapotában már használható, de a valós használat során gyűjtött tapasztalatok alapján további finomításokra lesz szükség.

Hosszabb távon az alkalmazás bővíthető közösségi funkciókkal, értesítésekkel és személyre szabott ajánlásokkal. Emellett cél lehet további közlekedési rendszerek integrálása, valamint a mobilalkalmazás képességeinek növelése.

A projekt legfontosabb hozadéka azonban az, hogy egy valós problémára készült, ténylegesen használható szoftvert hoztunk létre. A továbbiakban azt várjuk tőle, hogy gyakorlati tapasztalatot biztosítson az üzemeltetésben, és alapot adjon későbbi fejlesztésekhez, akár más projektekben is.

Köszönetnyilvánítás

Ezúton szeretnénk köszönetet mondani mindazoknak, akik segítségükkel és támogatásukkal hozzájárultak a szakdolgozat és a HovaTovább alkalmazás elkészítéséhez.

Külön köszönettel tartozunk **Lakatos Sándor** és **Berki Balázs** szakmai oktatóknak, akik szakmai tanácsaikkal, útmutatásaikkal és folyamatos segítségükkel végigkísérték a fejlesztési folyamatot, valamint értékes javaslataikkal hozzájárultak a projekt szakmai színvonalának emeléséhez.

Köszönjük továbbá **Németh Péterné Petrikán Valéria** szakmai angoltanárnak a nyelvi támogatást és az idegen nyelvű szakmai anyagok feldolgozásában nyújtott segítséget.

Segítségük és támogatásuk nélkül projektünk nem készülhetett volna el ebben a formában.