**Utazástervező és költségkalkulátor**

**REST API: Úti célok, költségek, útvonalak**

### Célja

A REST API elsődleges célja, hogy biztosítsa az adatok strukturált, biztonságos és rugalmas kezelését az utazástervező weboldalon. Ez az architektúra lehetővé teszi, hogy a frontend és a backend hatékonyan kommunikáljon egymással, miközben az adatok jól szervezetten, szabványos formában kerülnek továbbításra. A REST API-n keresztül a felhasználók különféle műveleteket végezhetnek, amelyek az utazás megtervezésének és költségkezelésének alapvető elemei:

* **Úti célokat keressenek, mentsenek és szerkesszenek**: A felhasználók böngészhetnek a rendelkezésre álló úti célok között, részletes információkat kérhetnek le (pl. város, ország, leírás, képek), valamint lehetőségük van saját úti célokat hozzáadni, módosítani vagy törölni. Ez lehetővé teszi a személyre szabott utazási tervek kialakítását.
* **Költségeket kalkuláljanak és nyomon kövessenek**: A rendszer támogatja a különböző költségtípusok (szállás, közlekedés, étkezés, programok) rögzítését és kezelését. A REST API segítségével a felhasználók új költségeket vihetnek fel, módosíthatják a meglévőket, illetve lekérdezhetik az összesített vagy részletes költségadatokat. Ez segít a pénzügyi tervezésben és a költségkeret betartásában.
* **Útvonalakat állítsanak össze és módosítsanak**: A felhasználók több úti célból álló útvonalakat hozhatnak létre, amelyekhez megadhatják az utazás kezdő és záró dátumát, valamint a sorrendet. A REST API lehetőséget biztosít az útvonalak mentésére, szerkesztésére és törlésére, így az utazás teljes struktúrája dinamikusan kezelhető.

### Főbb végpontok (endpoints)

| **Végpont** | **Művelet típusa** | **Leírás** |
| --- | --- | --- |
| /api/destinations | GET | Elérhető úti célok listázása |
| /api/destinations/:id | GET | Egy adott úti cél részletes adatai |
| /api/destinations | POST | Új úti cél hozzáadása |
| /api/destinations/:id | PUT | Úti cél adatainak módosítása |
| /api/destinations/:id | DELETE | Úti cél törlése |
| /api/costs | POST | Költségek hozzáadása (szállás, közlekedés, étkezés stb.) |
| /api/costs/:id | GET | Egy adott költségtétel lekérdezése |
| /api/routes | POST | Útvonal létrehozása több úti célból |
| /api/routes/:id | GET | Útvonal részleteinek lekérdezése |
| /api/routes/:id | PUT | Útvonal módosítása |
| /api/routes/:id | DELETE | Útvonal törlése |

### Adatszerkezetek (példák)

**Úti cél (Destination):**

****

Költség (Cost):

****

Útvonal (Route):

****

### Hitelesítés és jogosultság

A rendszer biztonságos működésének alapja a megfelelő hitelesítési és jogosultságkezelési mechanizmus. Ennek célja, hogy a felhasználók csak a saját adataikhoz férjenek hozzá, míg az adminisztrátorok speciális jogosultságokkal rendelkezzenek a globális tartalom kezeléséhez.

* **JWT token alapú hitelesítés**: A felhasználók bejelentkezés után egy JSON Web Token-t (JWT) kapnak, amelyet minden további kérésnél elküldenek a szervernek. Ez a token tartalmazza a felhasználó azonosítóját és jogosultsági szintjét, így a szerver gyorsan és biztonságosan ellenőrizheti, hogy ki küldi a kérést, és milyen műveleteket végezhet.
* **Felhasználók csak saját útvonalakat és költségeket módosíthatnak**: A rendszer úgy van kialakítva, hogy minden felhasználó kizárólag a saját fiókjához tartozó adatokat kezelheti. Ez azt jelenti, hogy más felhasználók útvonalaihoz, költségterveihez vagy személyes beállításaihoz nem férhet hozzá, így biztosított az adatvédelem és a személyes információk biztonsága.
* **Admin jogosultság szükséges úti célok globális kezeléséhez**: Az alkalmazásban szereplő úti célok – például városok, országok, turisztikai helyszínek – központilag kerülnek kezelésre. Ezek módosításához, törléséhez vagy új elemek hozzáadásához adminisztrátori jogosultság szükséges. Ez biztosítja, hogy a publikus adatbázis megbízható és konzisztens maradjon.

### Előnyök

A fent bemutatott hitelesítési és jogosultságkezelési rendszer számos előnyt kínál, különösen a skálázhatóság, biztonság és integrálhatóság terén:

* **Könnyen bővíthető és integrálható mobilalkalmazással**: A JWT tokenek használata lehetővé teszi, hogy a webes alkalmazás egyszerűen összekapcsolható legyen mobilalkalmazásokkal is. A tokenek platformfüggetlenek, így ugyanazt a hitelesítési logikát használhatjuk Androidon, iOS-en vagy bármilyen más eszközön.
* **Strukturált adatkezelés**: A jogosultsági szintek és token-alapú hozzáférés révén az adatok jól szervezetten, logikusan és biztonságosan kezelhetők. Ez megkönnyíti a fejlesztést, a hibakeresést és az adatvédelmi megfelelést is.
* **Gyors válaszidő és skálázhatóság**: Mivel a JWT tokenek nem igényelnek állandó adatbázis-lekérdezést a felhasználó azonosításához, a rendszer gyorsan tud reagálni a kérésekre. Ez különösen fontos nagyobb felhasználói szám esetén, ahol a teljesítmény és a skálázhatóság kulcsfontosságú.

## Webes kliens: Interaktív térkép és költségösszesítő

### Célja

A webes felület célja, hogy az utazástervező alkalmazás használata során a felhasználók számára egy intuitív, vizuálisan vonzó és könnyen kezelhető környezetet biztosítson. A felület kialakítása során kiemelt figyelmet kap a felhasználói élmény, a logikus navigáció, valamint az esztétikus megjelenés, amely segíti a látogatót abban, hogy gyorsan és hatékonyan tudja megtervezni utazását. A rendszer úgy épül fel, hogy a felhasználók minimális technikai tudással is képesek legyenek kihasználni annak lehetőségeit, miközben a haladó felhasználók számára is biztosított a testreszabhatóság és a részletes adatkezelés.

A webes kliens két kiemelt funkcióra épül, amelyek az utazástervezés legfontosabb elemeit támogatják:

* **Interaktív térkép**: Ez a funkció lehetővé teszi az útvonalak vizuális megjelenítését, valamint az úti célok kijelölését közvetlenül a térképen. A felhasználó egyszerűen rákattinthat a kívánt városokra vagy országokra, és hozzáadhatja őket az utazási tervéhez. A térkép dinamikusan frissül, és vonalakkal köti össze az egyes pontokat, így jól láthatóvá válik az utazás sorrendje és logikája. Az interaktív térkép nemcsak esztétikai élményt nyújt, hanem praktikus eszközként is szolgál az útvonalak összeállításához.
* **Költségösszesítő**: A tervezett utazás pénzügyi oldalának kezelése szintén központi szerepet kap. A költségösszesítő funkció részletes és grafikus formában mutatja be az utazás során felmerülő kiadásokat, különböző kategóriákra bontva (pl. szállás, közlekedés, étkezés, programok). A rendszer lehetőséget biztosít arra, hogy a felhasználó valós időben nyomon kövesse a költségeket, összehasonlítsa az egyes elemeket, és akár költségkeretet is beállítson. A vizuális megjelenítés – például oszlopdiagramok, kördiagramok – segít abban, hogy az adatok könnyen értelmezhetők legyenek, és a felhasználó átlássa az utazás pénzügyi vonatkozásait.

Ez a két funkció együtt biztosítja, hogy a webes felület ne csupán információt közvetítsen, hanem aktív tervezési eszközként szolgáljon, amely támogatja a felhasználót az utazás minden szakaszában – az ötleteléstől a konkrét megvalósításig.

## Interaktív térkép

Az interaktív térkép az utazástervező weboldal egyik legfontosabb vizuális eleme, amely lehetővé teszi a felhasználók számára, hogy térképes nézetben állítsák össze utazásuk útvonalát. A térkép nem csupán látványos megjelenítést kínál, hanem aktív tervezési eszközként is szolgál, amely intuitív módon segíti a felhasználót a döntéshozatalban. A funkciók úgy lettek kialakítva, hogy a kezelőfelület egyszerűen használható legyen, miközben részletes információkat nyújt az egyes úti célokról.

### Funkciók

* **Úti célok kijelölése**: A felhasználó a térképen megjelenő városokra, országokra kattintva könnyedén hozzáadhatja azokat az utazási tervéhez. Ez a funkció lehetővé teszi, hogy a tervezés vizuálisan történjen, így az utazás földrajzi elhelyezkedése azonnal átláthatóvá válik. A kijelölt úti célok automatikusan bekerülnek az útvonalba, amely később szerkeszthető és bővíthető.
* **Útvonal vizualizáció**: A rendszer a kiválasztott úti célokat vonalakkal köti össze, ezzel megjelenítve az utazás sorrendjét és logikai felépítését. A vizualizáció segít abban, hogy a felhasználó lássa, milyen útvonalon halad majd, és szükség esetén módosíthassa a sorrendet vagy új célpontokat adjon hozzá. Ez különösen hasznos többállomásos utazások esetén, ahol a térképes nézet segít optimalizálni az útvonalat.
* **Információs ablakok**: Amikor a felhasználó egy adott úti célra kattint, megjelenik egy információs ablak, amely részletes adatokat tartalmaz az adott helyszínről. Ezek az adatok lehetnek például leírások, képek, ajánlott programok, valamint az adott célponthoz kapcsolódó költségek. Az információs ablakok célja, hogy a felhasználó gyorsan és egyszerűen hozzáférjen minden releváns információhoz anélkül, hogy elhagyná a térképes nézetet.
* **Zoom és navigáció**: A térkép szabadon nagyítható és mozgatható, így a felhasználó könnyedén fókuszálhat egy adott régióra vagy országra. A zoom funkció lehetővé teszi, hogy részletesen megvizsgálja az egyes területeket, míg a navigációs lehetőségek biztosítják, hogy a térkép kezelése gördülékeny és intuitív legyen. Ez különösen fontos mobil eszközökön, ahol az érintéses vezérlés révén a térkép könnyen kezelhető.

## Költségösszesítő

A költségösszesítő funkció célja, hogy a felhasználó átlátható és részletes képet kapjon az utazás pénzügyi vonatkozásairól. Ez a modul lehetővé teszi a különböző kiadások kategorizálását, vizuális megjelenítését és elemzését, így segít a tudatos tervezésben és a költségek optimalizálásában. A funkciók úgy lettek kialakítva, hogy egyszerre legyenek informatívak, könnyen értelmezhetők és vizuálisan vonzóak.

### Funkciók

* **Költségtípusok bontása**: A rendszer lehetőséget biztosít arra, hogy a felhasználó külön kategóriákba sorolja az utazás során felmerülő kiadásokat. Ezek a kategóriák lehetnek például szállás, közlekedés, étkezés, programok stb. A bontás segít abban, hogy a felhasználó pontosan lássa, mely területeken költ a legtöbbet, és hol van lehetőség megtakarításra.
* **Grafikus megjelenítés**:
  + **Oszlopdiagram**: A költségek típus szerinti bontását oszlopdiagram formájában jeleníti meg, így könnyen összehasonlíthatók az egyes kategóriák.
  + **Kördiagram**: Az összes költség arányait kördiagram formájában mutatja, amely vizuálisan jól érzékelteti, hogy az egyes kiadástípusok milyen súlyt képviselnek az összköltségben.
  + **Idővonal**: A költségek időbeli eloszlását idővonalon ábrázolja, amely segít megérteni, hogy az utazás mely szakaszában jelentkeznek a legnagyobb kiadások.
* **Valutaátváltás**: A rendszer automatikusan átváltja a költségeket a felhasználó által választott pénznemre. Ez különösen hasznos nemzetközi utazások esetén, ahol többféle valuta is szerepelhet a költségtervben. Az átváltás valós idejű árfolyamadatok alapján történik, így mindig pontos értékek jelennek meg.
* **Összesítő panel**: A költségösszesítő egy külön panelen jeleníti meg a legfontosabb pénzügyi adatokat, mint például a teljes költség, a napi átlagos kiadás, valamint a legdrágább elem kiemelése. Ez az összesítő segít gyorsan áttekinteni az utazás pénzügyi kereteit, és támogatja a költségtudatos döntéshozatalt.

## Felhasználói élmény

A felhasználói élmény központi szerepet játszik az utazástervező weboldal sikerességében. A cél az, hogy az alkalmazás használata ne csak funkcionálisan legyen hatékony, hanem élvezetes és intuitív is. A felület kialakítása során kiemelt figyelmet kap a könnyű kezelhetőség, a testreszabhatóság, valamint az azonnali visszajelzés, amely segíti a felhasználót a gyors és pontos döntéshozatalban. Az alábbi funkciók mind hozzájárulnak ahhoz, hogy az alkalmazás valóban felhasználóbarát legyen:

* **Egyszerű kezelhetőség**: A rendszer lehetővé teszi, hogy a felhasználó drag-and-drop módszerrel adjon hozzá új úti célokat az utazási tervéhez. Ez azt jelenti, hogy a térképen vagy a listában megjelenő városokat egyszerűen áthúzhatja az útvonalba, így gyorsan és intuitívan állíthatja össze az utazást. Emellett a költségadatok valós időben frissülnek, így minden módosítás azonnal láthatóvá válik, ami megkönnyíti a pénzügyi tervezést.
* **Testreszabhatóság**: A felhasználó számos beállítást módosíthat annak érdekében, hogy az alkalmazás az egyéni igényeihez igazodjon. Ilyen lehetőségek például a színpaletta kiválasztása, a preferált valuta beállítása, az utazás időtartamának megadása, valamint a költségkeret rögzítése. Ezek a testreszabási lehetőségek nemcsak esztétikai szempontból fontosak, hanem segítenek abban is, hogy a felhasználó pontosan olyan környezetben tervezze meg az utazását, amely számára a legkényelmesebb és legáttekinthetőbb.
* **Valós idejű frissítés**: Az alkalmazás minden módosítást azonnal megjelenít a felületen, legyen szó új úti cél hozzáadásáról, költségadatok módosításáról vagy útvonalak átrendezéséről. Ez a valós idejű visszajelzés különösen fontos a térképes nézetben és a költségösszesítő modulban, ahol a felhasználó azonnal láthatja a változások hatását. A gyors reakcióidő és az automatikus frissítés hozzájárul a gördülékeny felhasználói élményhez, és csökkenti a hibalehetőségeket a tervezés során.

## Felhasználói fiókkezelés és regisztráció

### Célja

A felhasználói fiókkezelés célja, hogy az utazástervező weboldal személyre szabott élményt nyújtson minden látogatónak, miközben biztosítja az adatok biztonságos tárolását és elérhetőségét. Ez a funkció lehetővé teszi, hogy a felhasználók saját fiókot hozzanak létre, amelyhez egyedi azonosítóval (pl. email cím és jelszó) férnek hozzá. A regisztráció után a felhasználók bejelentkezhetnek, és egy biztonságos környezetben kezelhetik utazási terveiket, költségadataikat és egyéb személyes beállításaikat.

A fiókhoz kötött adatok – mint például az útvonalak, költségösszesítők, preferált valuta vagy időzóna – nemcsak biztonságosan tárolódnak, hanem szinkronizálhatók is több eszköz között. Ez azt jelenti, hogy a felhasználó bárhonnan hozzáférhet a saját információihoz, legyen szó számítógépről, táblagépről vagy mobiltelefonról. A szinkronizáció automatikusan történik, így az adatok mindig naprakészek maradnak, függetlenül attól, hogy melyik eszközön történt a módosítás.

Ez a megközelítés nemcsak kényelmet biztosít, hanem hozzájárul az adatbiztonsághoz és a felhasználói élmény egységességéhez is. A fiókhoz kötött működés révén az utazástervezés folyamata gördülékenyebbé válik, és a felhasználók hosszú távon is visszatérhetnek korábbi terveikhez, költségadataikhoz vagy beállításaikhoz.

**Funkciók**

A felhasználói fiókkezeléshez kapcsolódó funkciók célja, hogy biztonságos, megbízható és kényelmes módon biztosítsák a regisztrációt, bejelentkezést és a hozzáférést az utazástervező rendszerhez. A funkciók kialakítása során kiemelt figyelmet kapott az adatvédelem, a hibakezelés, valamint a felhasználói élmény egyszerűsége és gördülékenysége.

### Regisztráció

* **Adatok bekérése: Név, email cím, jelszó** A regisztrációs folyamat során a felhasználónak meg kell adnia néhány alapvető adatot, amelyek szükségesek a fiók létrehozásához. Ezek az adatok a név, az email cím és egy biztonságos jelszó. Az email cím egyedi azonosítóként szolgál, míg a név segít a személyre szabott felhasználói élmény kialakításában.
* **Jelszó biztonság: Minimum 8 karakter, kis- és nagybetű, szám, speciális karakter** A rendszer megköveteli, hogy a felhasználók erős jelszót válasszanak, amely legalább 8 karakterből áll, és tartalmaz kis- és nagybetűt, számot, valamint speciális karaktert. Ez a szabályozás az adatok védelme érdekében került bevezetésre, hogy minimalizálja a jogosulatlan hozzáférés kockázatát.
* **Email validáció: Aktiváló link küldése emailben** A regisztráció befejezéséhez a rendszer egy aktiváló linket küld a megadott email címre. A felhasználónak erre a linkre kell kattintania, hogy megerősítse az email cím valódiságát. Ez a lépés biztosítja, hogy a fiókhoz tartozó email valóban a felhasználóhoz tartozik, és elkerülhetővé teszi a hamis regisztrációkat.

### Bejelentkezés

* **Email + jelszó páros** A bejelentkezés során a felhasználó az előzőleg megadott email cím és jelszó kombinációjával tud hozzáférni a fiókjához. A rendszer ellenőrzi az adatokat, és sikeres egyezés esetén engedélyezi a belépést a személyes felületre.
* **„Maradj bejelentkezve” opció** A felhasználó választhatja a „Maradj bejelentkezve” lehetőséget, amely lehetővé teszi, hogy a rendszer megjegyezze a munkamenetet, így a következő látogatáskor automatikusan be legyen jelentkezve. Ez különösen hasznos mobil eszközökön vagy gyakori használat esetén.
* **Hibakezelés: Rossz jelszó, nem létező fiók, inaktív email** A rendszer részletes hibakezelést biztosít a bejelentkezés során. Ha a felhasználó hibás jelszót ad meg, nem létező fiókot próbál elérni, vagy az email cím még nem lett aktiválva, akkor pontos és érthető hibaüzenetet kap. Ez segíti a gyors hibaelhárítást, és javítja a felhasználói élményt.

## Fiókkezelés

A fiókkezelés funkció lehetővé teszi, hogy a felhasználók saját profiljukat testreszabják, biztonságosan módosítsák adataikat, valamint szükség esetén véglegesen töröljék fiókjukat. A cél az, hogy minden felhasználó teljes kontrollt gyakorolhasson a saját fiókja felett, miközben az adatok védelme és a rendszer integritása is biztosított marad.

* **Profil szerkesztése: Név, profilkép, preferált valuta, időzóna** A felhasználók módosíthatják a fiókjukhoz tartozó alapvető információkat, mint például a nevüket, a profilképüket, valamint beállíthatják az általuk preferált valutát és időzónát. Ezek a testreszabási lehetőségek hozzájárulnak a személyre szabott felhasználói élményhez, és segítenek abban, hogy az alkalmazás a felhasználó egyéni igényeihez igazodjon.
* **Jelszó módosítása** A rendszer lehetőséget biztosít arra, hogy a felhasználók bármikor megváltoztassák jelszavukat. Ez különösen fontos biztonsági szempontból, például ha a felhasználó gyanítja, hogy jelszava illetéktelen kezekbe került, vagy egyszerűen csak szeretné rendszeresen frissíteni azt.
* **Fiók törlése: Végleges adateltávolítás** A felhasználók dönthetnek úgy, hogy véglegesen törlik fiókjukat. Ebben az esetben a rendszer minden, a fiókhoz kapcsolódó adatot – beleértve az utazási terveket, költségadatokat és beállításokat – visszavonhatatlanul eltávolít. Ez a funkció biztosítja az adatvédelmi jogok érvényesülését, és lehetőséget ad arra, hogy a felhasználó teljes mértékben megszüntesse jelenlétét az alkalmazásban.

## Jelszó-visszaállítás

A jelszó-visszaállítási funkció akkor válik szükségessé, ha a felhasználó elfelejti jelszavát, vagy nem tud bejelentkezni a fiókjába. A rendszer biztonságos és egyszerű módot kínál a hozzáférés visszaállítására.

* **Emailes link küldése** A felhasználó kérésére a rendszer egy egyedi, időkorlátos visszaállító linket küld a regisztrált email címre. Ez a link lehetővé teszi, hogy a felhasználó biztonságosan elindítsa a jelszó-visszaállítási folyamatot, anélkül hogy harmadik fél hozzáférhetne az adataihoz.
* **Új jelszó megadása biztonságos formában** A visszaállító linkre kattintva a felhasználó egy biztonságos felületen új jelszót adhat meg. A rendszer ellenőrzi, hogy az új jelszó megfelel-e a biztonsági követelményeknek (pl. hossz, karaktertípusok), majd frissíti a fiókhoz tartozó hitelesítési adatokat. Ezután a felhasználó ismét hozzáférhet a fiókjához, és folytathatja az utazástervezést.

## Felhasználói élmény

A felhasználói élmény kiemelt szerepet kap az utazástervező weboldal kialakításában, hiszen a cél az, hogy az alkalmazás használata minden eszközön kényelmes, gyors és biztonságos legyen. A rendszer úgy lett megtervezve, hogy a technikai háttér és a vizuális megjelenés egyaránt támogassa a gördülékeny működést, miközben a felhasználók számára intuitív és élvezetes felületet biztosít.

* **Reszponzív felület: Mobilon és asztali gépen is könnyen használható** A weboldal kialakítása reszponzív technológiával történik, így automatikusan alkalmazkodik a különböző képernyőméretekhez és eszköztípusokhoz. Legyen szó okostelefonról, táblagépről vagy asztali számítógépről, a felhasználó minden esetben könnyedén navigálhat az oldalon, és teljes funkcionalitással használhatja az alkalmazást.
* **Felhasználóbarát űrlapok: Valós idejű hibajelzések, jelszóerősség-ellenőrzés** Az adatbeviteli űrlapok úgy lettek kialakítva, hogy a felhasználó számára egyszerű és egyértelmű legyen a kitöltésük. A rendszer valós időben jelzi, ha egy mező hibásan lett kitöltve, vagy ha hiányzik egy kötelező adat. Emellett a jelszó megadásakor az alkalmazás vizuálisan jelzi annak erősségét, ezzel segítve a biztonságos jelszóválasztást.
* **Biztonságos munkamenet: Token-alapú védelem, automatikus kijelentkezés inaktivitás esetén** A rendszer minden felhasználói munkamenetet biztonságos token-alapú hitelesítéssel véd, amely megakadályozza a jogosulatlan hozzáférést. Emellett, ha a felhasználó hosszabb ideig inaktív, az alkalmazás automatikusan kijelentkezteti, ezzel tovább növelve az adatbiztonságot és csökkentve a kockázatokat nyilvános vagy megosztott eszközök használata esetén.

## Mobilbarát felület: Gyors útvonaltervezés és offline mentés

### Célja

A mobilbarát felület célja, hogy az alkalmazás minden környezetben – különösen mobil eszközökön – kényelmesen és hatékonyan használható legyen. A modern utazók gyakran útközben, telefonról vagy táblagépről tervezik meg útvonalaikat, ellenőrzik a költségeiket, vagy módosítanak a terveiken. Ezért elengedhetetlen, hogy az alkalmazás gyors válaszidővel, optimalizált megjelenítéssel és intuitív kezelőfelülettel rendelkezzen, amely lehetővé teszi a gördülékeny navigációt kisebb képernyőkön is.

A mobilbarát kialakítás nem csupán a vizuális elrendezésre vonatkozik, hanem a funkciók elérhetőségére és használhatóságára is. A felhasználó számára biztosított, hogy minden lényeges funkció – például az útvonaltervezés, költségösszesítés, úti célok kezelése – teljes értékűen működjön mobilon is, kompromisszumok nélkül.

Emellett az alkalmazás lehetőséget ad arra, hogy a felhasználó offline módban is hozzáférjen a korábban mentett útvonalakhoz és költségtervekhez. Ez különösen hasznos olyan helyzetekben, amikor nincs elérhető internetkapcsolat – például külföldi utazás során, repülőn, vagy vidéki területeken. Az offline hozzáférés biztosítja, hogy a felhasználó bármikor meg tudja tekinteni a legfontosabb információkat, és ne legyen kiszolgáltatva a hálózati körülményeknek.

A mobilbarát felület és az offline mentés együttesen hozzájárulnak ahhoz, hogy az alkalmazás valóban rugalmas, megbízható és felhasználóközpontú legyen, bárhol és bármikor.

## Reszponzív dizájn

### Funkciók

* **Mobiloptimalizált elrendezés:** Az alkalmazás felülete úgy lett kialakítva, hogy a legfontosabb elemek – mint például a menü, a térkép, a költségösszesítő és az útvonaltervező – automatikusan igazodnak a különböző képernyőméretekhez. Ez biztosítja, hogy a felhasználók okostelefonon és táblagépen is könnyedén hozzáférjenek minden funkcióhoz, kompromisszumok nélkül.
* **Érintésbarát vezérlés:** A kezelőfelület elemei – például gombok, ikonok és navigációs sávok – nagy méretűek és jól elkülöníthetők, így könnyen kezelhetők érintőképernyőn. A gesztusvezérlés (mint a húzás, koppintás vagy csúsztatás) intuitív működést tesz lehetővé, ami különösen fontos mobilhasználat során.
* **Gyors betöltés:** Az alkalmazás technikai háttere optimalizálva van a gyors működés érdekében. Minimalizált adatforgalommal, tömörített képekkel és hatékony komponenskezeléssel biztosítja, hogy a felhasználó ne tapasztaljon késlekedést, még gyengébb hálózati környezetben sem.

## Gyors útvonaltervezés

### Funkciók

* **Úti cél hozzáadása egy koppintással:** A felhasználó egyetlen érintéssel hozzáadhat új helyszíneket az útvonalhoz, ami gyorsabbá és egyszerűbbé teszi a tervezést, különösen mobilon.
* **Automatikus útvonaljavaslat:** Az alkalmazás intelligens algoritmusai az úti célok alapján automatikusan javaslatot tesznek az optimális útvonalra, figyelembe véve a távolságot, az utazási időt és a várható költségeket. Ez segít a felhasználónak abban, hogy hatékonyan tervezzen, anélkül hogy manuálisan kellene összehasonlítania az alternatívákat.
* **Drag-and-drop sorrendmódosítás:** A felhasználó egyszerűen átrendezheti az útvonal elemeit mobilon is, csupán húzással. Ez a funkció lehetővé teszi a rugalmas tervezést, amikor változnak az igények vagy a prioritások.

### Felhasználói élmény

* **Valós idejű frissítés:** Az alkalmazás azonnal megjeleníti a térképen a módosításokat, így a felhasználó azonnal láthatja az új útvonalat vagy a frissített információkat. Ez különösen hasznos, ha gyors döntésekre van szükség utazás közben.
* **Költségbecslés az útvonal alapján:** A mobilon is elérhető összesítő panel megjeleníti az adott útvonalhoz tartozó várható költségeket. Ez segít a felhasználónak abban, hogy előre kalkuláljon, és tudatosan döntsön az utazás pénzügyi vonatkozásairól.

### Offline mentés

### Funkciók

* **Mentett útvonalak és költségtervek elérhetősége internet nélkül:** Az alkalmazás lehetővé teszi, hogy a felhasználók internetkapcsolat hiányában is hozzáférjenek a korábban elmentett útvonalaikhoz és költségterveikhez. Ez különösen hasznos utazás közben, amikor nem mindig áll rendelkezésre stabil hálózat, például repülőn, vidéki területeken vagy külföldön.
* **Automatikus szinkronizálás:** Amint az eszköz újra csatlakozik az internethez, az alkalmazás automatikusan frissíti az adatokat. Ez azt jelenti, hogy a felhasználónak nem kell manuálisan elindítania a szinkronizálást – minden háttérben történik, zökkenőmentesen és megbízhatóan.
* **Helyi tárolás:** Az offline elérhetőség technikai alapját a böngésző vagy a mobilalkalmazás cache-e biztosítja. Az adatok ideiglenesen az eszközön tárolódnak, így azok gyorsan és közvetlenül elérhetők, anélkül hogy külső szerverekhez kellene kapcsolódni.

### Előnyök

* **Utazás közbeni használhatóság:** Az offline mentés lehetősége biztosítja, hogy az alkalmazás akkor is használható maradjon, amikor nincs aktív internetkapcsolat. Ez növeli a rugalmasságot és a megbízhatóságot, hiszen a felhasználó bármikor hozzáférhet a számára fontos információkhoz.
* **Gyors reakcióidő:** A helyi tárolásnak köszönhetően az alkalmazás mobilon is gyorsan reagál a felhasználói műveletekre. Az adatok betöltése és megjelenítése nem függ a hálózati sebességtől, így a működés gördülékeny marad még gyenge kapcsolat esetén is.
* **Felhasználói elégedettség:** Az offline mentés funkciója hozzájárul ahhoz, hogy az alkalmazás modern, megbízható és felhasználóbarát élményt nyújtson minden eszközön. A felhasználók értékelik, ha egy alkalmazás nem hagyja őket cserben nehezebb körülmények között sem, és ez növeli az általános elégedettséget és bizalmat a szolgáltatás iránt.

**Funkciók és szolgáltatások bővítése**

## 1.1 Dinamikus árfigyelés

### Célkitűzés

A dinamikus árfigyelés funkció elsődleges célja, hogy a felhasználók számára folyamatosan frissülő, megbízható és naprakész információkat biztosítson az utazással kapcsolatos költségek alakulásáról. Az utazás tervezése során az egyik legfontosabb szempont a költségek optimalizálása, különösen akkor, ha a felhasználó rugalmasan tudja alakítani az indulás időpontját, az úti célt vagy a programokat. A rendszer úgy lett kialakítva, hogy automatikusan nyomon kövesse a különböző utazási elemek – például repülőjegyek, szállások és programok – árváltozásait, és ezekről időben, jól érthető formában értesítést küldjön a felhasználónak.

Ez a funkció nem csupán kényelmi szolgáltatásként működik, hanem aktívan hozzájárul a költséghatékony utazástervezéshez. A felhasználók számára lehetőséget biztosít arra, hogy az árak alakulását figyelembe véve hozzanak döntéseket, például mikor érdemes lefoglalni egy szállást vagy repülőjegyet. A rendszer által küldött értesítések segítenek abban, hogy a felhasználó ne maradjon le egy kedvező ajánlatról, és ne kelljen naponta manuálisan ellenőriznie az árakat. Ezáltal a döntéshozatal nemcsak gyorsabbá, hanem megalapozottabbá is válik.

A dinamikus árfigyelés tehát egy olyan intelligens modul, amely az utazási költségek optimalizálásán túl hozzájárul a felhasználói élmény növeléséhez, az időgazdálkodás hatékonyságához, valamint a tudatos tervezéshez. A funkció különösen hasznos lehet azok számára, akik csoportosan utaznak, vagy több alternatívát mérlegelnek az utazás során.

### Funkcionális leírás

A dinamikus árfigyelés modul több egymással összefüggő funkcióból áll, amelyek együttesen biztosítják a rendszer teljes körű működését. Az alábbiakban részletezve bemutatjuk a modul főbb funkcióit:

* **Árfigyelés repülőjegyekre:** A rendszer közvetlenül kapcsolódik különböző repülőjegy-szolgáltatókhoz, például légitársaságokhoz és aggregátor platformokhoz, API-n keresztül. A felhasználó által megadott útvonalakra és időpontokra vonatkozóan a rendszer folyamatosan monitorozza az aktuális árakat, és rögzíti az esetleges változásokat. Ez lehetővé teszi, hogy a felhasználó az optimális ár eléréséhez időben tudjon foglalni. A funkció támogatja az egyirányú, retúr és több szakaszos útvonalakat is, valamint képes figyelembe venni az átszállások számát és időtartamát.
* **Szállásárak követése:** A modul integrálva van olyan népszerű szállásfoglaló platformokkal, mint például a Booking.com vagy az Airbnb. A felhasználó által kiválasztott szálláshelyek árváltozásait a rendszer automatikusan rögzíti, és összehasonlítja a korábbi árakkal. Ez segít abban, hogy a felhasználó a legkedvezőbb időpontban tudja lefoglalni a szállást. A rendszer képes megkülönböztetni a különböző szobatípusokat, szolgáltatásokat és lemondási feltételeket, így a kalkulált árak pontosan tükrözik a választott opciókat.
* **Programok és belépők figyelése:** A rendszer képes nyomon követni különféle programok – például kulturális események, múzeumi belépők, túrák – jegyáraiban bekövetkező változásokat. Ez különösen hasznos lehet azok számára, akik előre szeretnék megtervezni programjaikat, és szeretnék elkerülni a drágább időszakokat. A funkció támogatja a helyszín szerinti szűrést, az esemény típusát, valamint a jegytípusok (pl. felnőtt, diák, csoportos) árainak összehasonlítását is.
* **Értesítési rendszer:** A rendszer többféle értesítési formát támogat, beleértve az e-mailes értesítéseket, push üzeneteket mobil eszközökre, valamint webes felugró ablakokat. Az értesítések célja, hogy a felhasználó azonnal tudomást szerezzen az árak csökkenéséről vagy emelkedéséről, így időben tud reagálni. A felhasználó beállíthatja, milyen gyakorisággal szeretne értesítést kapni, és milyen árküszöb esetén legyen figyelmeztetve. Az értesítések tartalmazzák az aktuális árat, az előző árhoz viszonyított változást, valamint a foglalásra mutató hivatkozást.
* **Árdiagram és előzmények:** A rendszer grafikus formában jeleníti meg az árak alakulását, amely lehetővé teszi az ártrendek vizuális követését. A felhasználó visszamenőleg is megtekintheti, hogyan változtak az árak az adott szolgáltatás esetében, ami segíti a döntéshozatalt. A diagramok interaktívak, lehetőség van időintervallum kiválasztására, valamint az egyes szolgáltatások összehasonlítására. Az előzmények exportálhatók, és a felhasználó saját profiljához menthetők, így később is visszakereshetők.

### Felhasználói felület és dizájn

A dinamikus árfigyeléshez tartozó felhasználói felület kialakításánál elsődleges szempont volt a letisztultság, az áttekinthetőség és a könnyű kezelhetőség. A cél az volt, hogy a felhasználók gyorsan és intuitív módon hozzáférjenek az árakkal kapcsolatos információkhoz, és könnyedén értelmezhessék azok változásait. A felület vizuális megjelenése és funkcionális elrendezése egyaránt támogatja a hatékony döntéshozatalt, miközben esztétikailag is illeszkedik az utazástervező weboldal egységes arculatához.

**A főbb jellemzők az alábbiak szerint részletezhetők:**

* **Színvilág:** A rendszer ármozgásokat sötétkék és világoskék színkódokkal jelöli. A sötétkék árnyalat a stabil vagy csökkenő árakat reprezentálja, míg a világoskék az emelkedő árakat emeli ki. Ez a színpaletta nemcsak vizuálisan kellemes, hanem funkcionálisan is jól használható, mivel segíti a felhasználót az árak gyors értelmezésében. A színek kontrasztja biztosítja, hogy a fontos információk kiemelkedjenek, ugyanakkor nem zavarják meg a felhasználói élményt. A színvilág harmonizál a weboldal többi moduljával, így egységes, professzionális megjelenést eredményez.
* **Interaktív grafikonok:** A felhasználó számára lehetőség nyílik különböző időintervallumok kiválasztására, amelyek alapján megtekintheti az árak alakulását. A grafikonok dinamikusan frissülnek, és lehetőséget biztosítanak az ártrendek részletes elemzésére. A diagramok interaktívak: az egérrel vagy érintéssel történő mozgás során megjelennek az adott időponthoz tartozó konkrét árértékek, valamint az előző időszakhoz viszonyított változások. A grafikonok típusai között választható vonaldiagram, oszlopdiagram és területdiagram, attól függően, hogy a felhasználó milyen vizuális megjelenítést preferál.
* **Kártyás elrendezés:** Minden figyelt tétel – legyen az repülőjegy, szállás vagy program – külön kártyán jelenik meg. A kártyák tartalmazzák az aktuális árat, az előző árhoz viszonyított változást, valamint egy foglalásra irányuló gombot, amely közvetlenül elérhetővé teszi a szolgáltatást. A kártyák elrendezése rácsos vagy listás formában is megjeleníthető, a felhasználó preferenciái szerint. A kártyákon belül ikonok, színkódok és rövid szöveges összefoglalók segítik az információ gyors feldolgozását.
* **Mobilbarát kialakítás:** A felület teljes mértékben reszponzív, azaz különböző képernyőméretekhez automatikusan alkalmazkodik. Mobil eszközökön is gyorsan betöltődik, és az értesítések integrálhatók a rendszer értesítési sávjába, így a felhasználó bárhol és bármikor hozzáférhet az információkhoz. A mobilos nézet optimalizált gombméreteket, egyszerűsített menüstruktúrát és érintésre érzékeny elemeket tartalmaz, amelyek megkönnyítik a navigációt. A mobilos felület támogatja az értesítések közvetlen megnyitását, valamint a foglalási folyamat elindítását egyetlen érintéssel.

### Technikai megvalósítás

A dinamikus árfigyelés modul technikai háttere korszerű, megbízható és skálázható megoldásokra épül, amelyek biztosítják a rendszer stabil működését, a gyors adatfeldolgozást és a biztonságos felhasználói élményt. A fejlesztés során figyelembe vettük a modularitást, az integrálhatóságot és a jövőbeli bővíthetőséget is.

A technikai komponensek az alábbiak szerint épülnek fel:

* **Backend:** A szerveroldali működést egy Node.js alapú rendszer biztosítja, amely REST API-n keresztül kommunikál az árfigyelő szolgáltatásokkal. A backend felelős az adatok lekéréséért, feldolgozásáért, tárolásáért és továbbításáért a frontend felé. A REST API struktúrája jól dokumentált, és támogatja az aszinkron adatkezelést, amely lehetővé teszi a gyors válaszidőt és a skálázhatóságot. A szerveroldali logika tartalmazza az értesítési szabályokat, az árküszöbök kezelését, valamint az adatnaplózást.
* **Frontend:** A felhasználói felület React technológiával készült, amely lehetővé teszi a dinamikus tartalomkezelést és a gyors felhasználói interakciókat. A React komponensalapú felépítése elősegíti a moduláris fejlesztést, a könnyű karbantartást és az új funkciók gyors integrálását. A frontend felelős az árak megjelenítéséért, a grafikonok rendereléséért, az értesítések fogadásáért, valamint a felhasználói beállítások kezeléséért. A felület támogatja a többnyelvű megjelenítést, valamint a sötét és világos témák közötti váltást.
* **Adatbázis:** A rendszer MySQL-t használ az adatok tárolására, amely rugalmasan kezeli az árfigyelési előzményeket, a felhasználói beállításokat és az értesítési naplókat. A MySQLstruktúra lehetővé teszi a gyors lekérdezéseket, az adatok hatékony indexelését és a skálázható adattárolást. Az adatbázisban tárolt információk titkosítva kerülnek mentésre, és a hozzáférés jogosultságokhoz kötött, így biztosítva az adatbiztonságot.

### Használhatóság és működés

A dinamikus árfigyelés modul használata során a felhasználó egyszerűen és intuitív módon tudja beállítani, hogy mely utazási elemeket szeretné figyelemmel kísérni. Ez lehet például egy konkrét repülőjárat, egy kiválasztott szálláshely vagy egy adott program, esemény. A rendszer lehetőséget biztosít arra, hogy a felhasználó egyéni preferenciái alapján válassza ki azokat a tételeket, amelyek számára relevánsak, és amelyek árának alakulása befolyásolhatja az utazási döntéseit.

Miután a felhasználó elvégezte a kívánt beállításokat, a rendszer automatikusan elindítja az árfigyelési folyamatot. Ez azt jelenti, hogy a háttérben folyamatosan monitorozza az adott szolgáltatás árát, és összeveti azt az előzőleg rögzített értékekkel. Amennyiben az ár eléri vagy meghaladja a felhasználó által beállított küszöbértéket – például egy meghatározott maximum vagy minimum ár –, a rendszer értesítést küld. Ez az értesítés lehet e-mail formájában, push üzenetként mobil eszközön, vagy akár webes felugró ablakon keresztül.

A felhasználó bármikor módosíthatja a figyelési paramétereket, például megváltoztathatja az árküszöböt, új elemeket adhat hozzá a figyelési listához, vagy eltávolíthatja a már nem releváns tételeket. Emellett lehetősége van arra is, hogy ideiglenesen szüneteltesse az értesítéseket, vagy teljesen kikapcsolja azokat, ha már nem kívánja követni az adott ármozgásokat. A rendszer minden módosítást azonnal érvényesít, és ennek megfelelően frissíti a figyelési logikát.

A modul működése során kiemelt figyelmet fordít a felhasználói élményre: a beállítási folyamat egyszerű, lépésről lépésre vezetett, és nem igényel technikai előképzettséget. A felület gyorsan betöltődik, az interakciók gördülékenyek, és a rendszer visszajelzései (pl. sikeres mentés, figyelmeztetés, hibaüzenet) egyértelműek és informatívak. A cél az, hogy a felhasználó teljes kontrollt gyakorolhasson az árfigyelés felett, miközben a rendszer a háttérben megbízhatóan végzi a feladatát.

### Tesztelés és validálás

A dinamikus árfigyelés funkció megbízható működésének biztosítása érdekében elengedhetetlen a rendszer alapos tesztelése és validálása. A tesztelési folyamat célja, hogy feltárja az esetleges hibákat, ellenőrizze a rendszer stabilitását, valamint igazolja, hogy az implementált megoldások megfelelnek a specifikációban rögzített elvárásoknak. A validálás során azt vizsgáljuk, hogy a rendszer valóban azt nyújtja-e, amit a felhasználók elvárnak tőle, és hogy a felhasználói élmény megfelel-e a tervezett szintnek.

**A funkció tesztelése során az alábbi szempontokat vizsgáljuk:**

* **Árfrissítések pontossága:** A rendszer által lekért áraknak pontosan meg kell egyezniük a szolgáltatók aktuális kínálatával. Ennek érdekében összehasonlító teszteket végzünk, ahol a lekért adatok valós idejű árakkal kerülnek összevetésre. Vizsgáljuk, hogy az árak frissítése megfelelő időközönként történik-e, és hogy a rendszer képes-e kezelni a hirtelen árváltozásokat, például akciók vagy szezonális kiugrások esetén.
* **Értesítések működése:** Az értesítési rendszer működésének tesztelése során ellenőrizzük, hogy az értesítések időben, megfelelő formában és tartalommal érkeznek-e. Vizsgáljuk, hogy a felhasználó által beállított küszöbértékek alapján történik-e az értesítés generálása, valamint hogy az értesítések kikapcsolása, szüneteltetése és újraaktiválása hibamentesen működik-e. Tesztfiókokkal szimuláljuk a különböző értesítési beállításokat, és figyeljük, hogy a rendszer megfelelően reagál-e az ármozgásokra.
* **Felhasználói interakciók:** A felület intuitív használhatóságát manuális teszteléssel ellenőrizzük. Vizsgáljuk, hogy a felhasználói műveletek – például beállítások módosítása, új figyelési elem hozzáadása, értesítések kezelése – gyorsan és hibamentesen végrehajthatók-e. A tesztelés során különböző felhasználói profilokat hozunk létre, és szimuláljuk a tipikus használati eseteket, hogy feltárjuk az esetleges UX problémákat.
* **Mobiltesztelés:** A rendszer mobil eszközökön történő működését külön teszteljük Android és iOS platformokon. Vizsgáljuk a felület reszponzivitását, az értesítések megjelenését és kezelhetőségét, valamint az interakciók (pl. érintés, görgetés, zoom) megfelelő működését. A cél az, hogy a funkció mobilon is teljes funkcionalitással, gyorsan és hibamentesen működjön.

### Fejlesztési környezet

A dinamikus árfigyelés modul fejlesztése során korszerű, iparági szinten elismert eszközöket és technológiákat alkalmaztunk, amelyek biztosítják a hatékony munkavégzést, a kód minőségét, valamint a projekt átláthatóságát és fenntarthatóságát.

* **Fejlesztői eszközök:** A fejlesztéshez használt főbb eszközök közé tartozik a Visual Studio Code, amely modern, testreszabható fejlesztői környezetet biztosít.
* **Verziókezelés:** A projekt Git alapú verziókövetést használ, amely lehetővé teszi a fejlesztési folyamat strukturált nyomon követését. Külön ágakat hoztunk létre a frontend és backend fejlesztéshez, valamint a teszteléshez, így biztosítva a párhuzamos munkavégzést és a hibamentes integrációt. A commit üzenetek szabványos formátumban készülnek, és minden változtatás dokumentálva van.

## 1.2 Időjárás-előrejelzés integráció

### Célkitűzés

Az időjárás-előrejelzés integráció célja, hogy az utazástervező rendszer felhasználói az utazás előkészítése és lebonyolítása során megbízható, valós idejű és előrejelzett meteorológiai információkhoz jussanak az általuk kiválasztott úti célokra vonatkozóan. Az időjárási viszonyok jelentős hatással lehetnek az utazás élményére, a programok megvalósíthatóságára, valamint a csomagolás és öltözködés tervezésére. Éppen ezért elengedhetetlen, hogy a felhasználók már az utazás megtervezésekor hozzáférjenek az aktuális és várható időjárási adatokhoz.

A funkció lehetővé teszi, hogy az utazók az időjárási körülmények figyelembevételével alakítsák ki napi programjaikat, kiválasszák a megfelelő ruházatot és felszerelést, valamint eldöntsék, hogy az adott időszak alkalmas-e az utazásra. Például egy tengerparti nyaralás esetén fontos tudni, hogy várható-e eső vagy vihar, míg egy hegyi túra esetén a hőmérséklet és a szélsebesség lehet döntő tényező. A rendszer által biztosított információk segítik a felhasználókat abban, hogy elkerüljék a kellemetlen meglepetéseket, és az időjárásnak megfelelően alakítsák ki utazási terveiket.

Az integráció célja továbbá az is, hogy a felhasználók ne külön alkalmazásokból vagy weboldalakról kelljen, hogy beszerezzék az időjárási adatokat, hanem közvetlenül az utazástervező felületen, egy helyen, összefüggésében láthassák azokat. Ez nemcsak kényelmesebbé teszi a tervezést, hanem növeli a rendszer használhatóságát és komplexitását is.

### Funkcionális leírás

Az időjárás-előrejelzés modul több egymással összefüggő funkciót tartalmaz, amelyek együttesen biztosítják a felhasználók számára a teljes körű meteorológiai tájékoztatást. A funkciók célja, hogy az időjárási információk ne csupán statikus adatokként jelenjenek meg, hanem aktívan támogassák az utazási döntéshozatalt.

* **Aktuális időjárási adatok megjelenítése:** A rendszer valós időben jeleníti meg az adott úti cél aktuális időjárási viszonyait. Az adatok között szerepel a hőmérséklet (°C), a páratartalom (%), a szélsebesség (km/h), az UV-index (skálán), valamint a látási viszonyok (pl. köd, tiszta égbolt). Ezek az információk segítenek a felhasználónak abban, hogy az adott pillanatban milyen körülményekre számíthat, és ennek megfelelően alakítsa ki napi programját vagy öltözetét.
* **Rövid távú előrejelzés (1–3 nap):** A rendszer óránkénti bontásban biztosítja az időjárási előrejelzést a következő 72 órára vonatkozóan. Ez különösen hasznos lehet azok számára, akik rövid utazást terveznek, vagy akik napi szinten szeretnék optimalizálni programjaikat. Az előrejelzés tartalmazza a várható hőmérsékletet, csapadék valószínűségét, szélirányt és sebességet, valamint az UV-indexet. Az adatok vizuálisan is megjelennek, például grafikonokon vagy idővonalakon, amelyek segítik az értelmezést.
* **Hosszú távú előrejelzés (4–10 nap):** A rendszer napi bontásban biztosítja az időjárási előrejelzést a következő 4–10 napra. Az adatok között szerepelnek a napi átlaghőmérséklet, a várható csapadék mennyisége és formája (pl. eső, hó), valamint a hőmérsékleti ingadozások. Ez a funkció különösen hasznos azok számára, akik hosszabb utazást terveznek, és szeretnék előre látni, hogy milyen időjárási viszonyokra számíthatnak az egyes napokon. A rendszer figyelembe veszi a regionális időjárási sajátosságokat is, így pontosabb előrejelzést nyújt.
* **Időjárási figyelmeztetések:** A rendszer automatikusan értesítést küld a felhasználónak, ha az adott úti célon extrém időjárási jelenség várható. Ilyen lehet például vihar, heves esőzés, extrém hőség, havazás, jégeső vagy erős szél. Az értesítések célja, hogy a felhasználó időben tudomást szerezzen a potenciálisan veszélyes körülményekről, és ennek megfelelően módosíthassa terveit. Az értesítések formája lehet e-mail, push üzenet vagy webes figyelmeztetés, és tartalmazzák az esemény típusát, várható időpontját és javasolt teendőket.
* **Időjárás alapú ajánlások:** A rendszer nemcsak adatokat szolgáltat, hanem intelligens ajánlásokat is tesz a felhasználó számára. Például ha eső várható, a rendszer javasolhat beltéri programokat, múzeumlátogatást vagy wellness szolgáltatásokat. Ha napsütéses idő várható, akkor szabadtéri tevékenységeket, túrákat vagy strandolást ajánlhat. Az ajánlások figyelembe veszik a felhasználó korábbi preferenciáit, az úti cél adottságait, valamint a programok elérhetőségét. Ezáltal az időjárás nem akadály, hanem tervezési szempontként jelenik meg az utazás során.

### Felhasználói felület és dizájn

Az időjárás-előrejelzés modul felhasználói felülete úgy lett kialakítva, hogy vizuálisan letisztult, informatív és könnyen értelmezhető legyen minden korosztály és technikai szintű felhasználó számára. A cél az volt, hogy az időjárási adatok ne csupán száraz információként jelenjenek meg, hanem intuitív, vizuálisan jól strukturált formában, amely segíti a gyors döntéshozatalt és a programtervezést.

A felület kialakításánál figyelembe vettük a modern webes dizájn irányelveket, a felhasználói szokásokat, valamint a mobilos és asztali eszközök közötti különbségeket. A modul minden eleme reszponzív, azaz automatikusan alkalmazkodik a képernyőmérethez, így biztosítva a zökkenőmentes használatot bármilyen eszközön.

**A főbb vizuális és funkcionális jellemzők az alábbiak szerint részletezhetők:**

* **Színvilág:** A felület színpalettája sötétkék és világoskék árnyalatokat használ, amelyek nemcsak esztétikailag harmonikusak, hanem funkcionálisan is jól elkülönítik az időjárási típusokat. A sötétkék háttér nyugalmat, stabilitást és megbízhatóságot sugall, míg a világoskék elemek kiemelik az aktuális, releváns adatokat. A színkódolás segíti a felhasználót abban, hogy első pillantásra felismerje az időjárási helyzetet, például a viharos időt vagy a napsütéses periódusokat. A színek kontrasztja biztosítja a jó olvashatóságot, még gyengébb fényviszonyok mellett is.
* **Ikonrendszer:** A modul minden időjárási jelenséghez egyértelmű, grafikus ikont rendel, amelyek vizuálisan is segítik az információ gyors értelmezését. Például napsütés esetén egy stilizált nap ikon jelenik meg, felhős időnél felhő, eső esetén csepp vagy esőfelhő, havazásnál hópehely. Az ikonok egységes stílusban készültek, és illeszkednek a weboldal általános arculatához. A grafikus elemek mérete és elhelyezése optimalizált, így nem zavarják a szöveges tartalmat, hanem kiegészítik azt.
* **Interaktív térkép:** A felhasználó az úti célra kattintva megtekintheti a helyi időjárást egy interaktív térképen, amely lehetőséget biztosít a nagyításra, mozgatásra és részletesebb adatok megjelenítésére. A térkép rétegei között váltható a hőmérséklet, csapadék, szélirány, UV-index és egyéb meteorológiai paraméterek. A térkép integrálva van a többi modulba, így például az útvonaltervezővel is összekapcsolható, és megjeleníthető, hogy az adott útvonal mentén milyen időjárási viszonyokra lehet számítani.
* **Mobilbarát megjelenés:** A modul teljes mértékben reszponzív kialakítású, azaz mobil eszközökön is hibamentesen működik. A grafikonok, ikonok és szöveges elemek automatikusan igazodnak a képernyőmérethez, így biztosítva a jó olvashatóságot és kezelhetőséget. A mobilos nézetben a menük egyszerűsítettek, a gombok nagyobb méretűek, és az érintésre optimalizált interakciók révén a felhasználó könnyedén navigálhat az adatok között. A rendszer támogatja a mobilos értesítéseket is, így a felhasználó azonnal értesülhet az időjárási figyelmeztetésekről, akár utazás közben is.

### Technikai megvalósítás

Az időjárás-előrejelzés modul technikai háttere korszerű, megbízható és skálázható megoldásokra épül, amelyek biztosítják a rendszer stabil működését, a gyors adatfeldolgozást és a biztonságos felhasználói élményt. A fejlesztés során kiemelt figyelmet fordítottunk az adatbiztonságra, a modularitásra és az integrálhatóságra, hogy a rendszer hosszú távon is fenntartható és bővíthető legyen.

**A technikai komponensek az alábbiak szerint épülnek fel:**

* **Backend:** A szerveroldali működést egy Node.js alapú rendszer biztosítja, amely REST API-n keresztül kommunikál az időjárási szolgáltatókkal. A backend felelős az adatok lekéréséért, feldolgozásáért, tárolásáért és továbbításáért a frontend felé. A REST API struktúrája jól dokumentált, és támogatja az aszinkron adatkezelést, amely lehetővé teszi a gyors válaszidőt és a skálázhatóságot. A szerveroldali logika tartalmazza az értesítési szabályokat, az időjárási figyelmeztetések kezelését, valamint az adatnaplózást.
* **Frontend:** A felhasználói felület React technológiával készült, amely lehetővé teszi a dinamikus tartalomkezelést és a gyors felhasználói interakciókat. A React komponensalapú felépítése elősegíti a moduláris fejlesztést, a könnyű karbantartást és az új funkciók gyors integrálását. A frontend felelős az időjárási adatok megjelenítéséért, a grafikonok rendereléséért, az értesítések fogadásáért, valamint a felhasználói beállítások kezeléséért. A felület támogatja a többnyelvű megjelenítést, valamint a sötét és világos témák közötti váltást.
* **Adatbázis:** A rendszer MySQL-t használ az adatok tárolására, amely rugalmasan kezeli a lekérdezett időjárási előzményeket, a felhasználói beállításokat és az értesítési naplókat. A MySQL struktúra lehetővé teszi a gyors lekérdezéseket, az adatok hatékony indexelését és a skálázható adattárolást. Az adatbázisban tárolt információk titkosítva kerülnek mentésre, és a hozzáférés jogosultságokhoz kötött, így biztosítva az adatbiztonságot.
* **API integráció:** A rendszer több külső szolgáltatással van összekötve, például az OpenWeatherMap, WeatherAPI vagy Meteomatics platformokkal. Ezekből származnak az időjárási adatok, amelyeket a rendszer feldolgoz és megjelenít. Az API-k integrációja során figyelembe vettük a szolgáltatók által biztosított adatfrissítési gyakoriságot, a válaszidőt, valamint az adatstruktúrák kompatibilitását. A rendszer képes kezelni az API-k esetleges hibáit, és alternatív adatforrásokat használ, ha szükséges.
* **Cache-elés:** A rendszer ideiglenesen tárolja az időjárási adatokat, hogy csökkentse az API-hívások számát, gyorsítsa a betöltést, és biztosítsa a felhasználói élmény folyamatosságát. A cache-elés időalapú szabályok szerint működik, és automatikusan frissül, ha az adatok elavulnak. Ez különösen fontos mobilos használat esetén, ahol az adatkapcsolat korlátozott lehet.

### Használhatóság és működés

Az időjárás-előrejelzés integráció modul használata során a felhasználó egyszerűen és intuitív módon választhatja ki az utazás tervezésekor releváns úti célt. A rendszer ezt követően automatikusan megjeleníti az adott helyszínhez tartozó aktuális meteorológiai adatokat, valamint a rövid és hosszú távú időjárási előrejelzéseket. A felhasználó számára ez a funkció különösen hasznos, mivel az időjárási viszonyok jelentős hatással lehetnek az utazás élményére, a programok megvalósíthatóságára, valamint a csomagolás és öltözködés tervezésére.

A rendszer lehetőséget biztosít arra, hogy a felhasználó egyéni preferenciái alapján beállítsa, milyen típusú időjárási eseményekről szeretne értesítést kapni. Például választhatja, hogy csak az extrém hőmérsékleti viszonyokról, viharokról, esőzésekről vagy havazásról kapjon figyelmeztetést. Ezeket a beállításokat a felhasználói profilban lehet elvégezni, és a rendszer ezeket automatikusan alkalmazza minden új úti cél kiválasztásakor.

Az értesítések célja, hogy a felhasználó időben tudomást szerezzen az esetleges időjárási változásokról, és ennek megfelelően módosíthassa utazási terveit. A rendszer az értesítéseket több csatornán keresztül képes továbbítani: e-mail formájában, push üzenetként mobil eszközökre, valamint webes felugró ablakokként a felületen belül. Az értesítések tartalmazzák az esemény típusát, várható időpontját, valamint javasolt teendőket (pl. program módosítása, csomagolási tanácsok).

A modul működése során kiemelt figyelmet fordít a felhasználói élményre. A felület gyorsan betöltődik, az interakciók gördülékenyek, és a rendszer visszajelzései (pl. sikeres mentés, figyelmeztetés, hibaüzenet) egyértelműek és informatívak. A cél az, hogy a felhasználó teljes kontrollt gyakorolhasson az időjárási információk felett, miközben a rendszer a háttérben megbízhatóan végzi a feladatát.

### Tesztelés és validálás

Az időjárás-előrejelzés modul megbízható működésének biztosítása érdekében elengedhetetlen a funkciók alapos tesztelése és validálása. A tesztelési folyamat célja, hogy feltárja az esetleges hibákat, ellenőrizze a rendszer stabilitását, valamint igazolja, hogy az implementált megoldások megfelelnek a specifikációban rögzített elvárásoknak. A validálás során azt vizsgáljuk, hogy a rendszer valóban azt nyújtja-e, amit a felhasználók elvárnak tőle, és hogy a felhasználói élmény megfelel-e a tervezett szintnek.

**A funkció tesztelése során az alábbi szempontokat vizsgáljuk:**

* **Adatok pontossága:** A rendszer által lekért időjárási adatoknak pontosan meg kell egyezniük a valós meteorológiai viszonyokkal. Ennek érdekében összehasonlító teszteket végzünk, ahol a lekért információkat hivatalos meteorológiai szolgálatok adataival vetjük össze. Vizsgáljuk, hogy az adatok frissítése megfelelő időközönként történik-e, és hogy a rendszer képes-e kezelni a hirtelen időjárási változásokat.
* **Frissítési gyakoriság:** A rendszernek biztosítania kell, hogy az időjárási adatok rendszeresen és megbízhatóan frissüljenek. A tesztelés során ellenőrizzük, hogy az API-hívások megfelelő időközönként történnek, és hogy az adatok naprakészen jelennek meg a felhasználói felületen. A frissítési ciklusok optimalizálása kulcsfontosságú a rendszer hatékonysága szempontjából.
* **Felhasználói interakciók:** A felület intuitív használhatóságát manuális teszteléssel ellenőrizzük. Vizsgáljuk, hogy a felhasználói műveletek – például úti cél kiválasztása, értesítési beállítások módosítása, térkép használata – gyorsan és hibamentesen végrehajthatók-e. A tesztelés során különböző felhasználói profilokat hozunk létre, és szimuláljuk a tipikus használati eseteket, hogy feltárjuk az esetleges UX problémákat.
* **Mobiltesztelés:** A rendszer mobil eszközökön történő működését külön teszteljük Android és iOS platformokon. Vizsgáljuk a felület reszponzivitását, az értesítések megjelenését és kezelhetőségét, valamint az interakciók (pl. érintés, görgetés, zoom) megfelelő működését. A cél az, hogy a funkció mobilon is teljes funkcionalitással, gyorsan és hibamentesen működjön.
* **Értesítési rendszer:** Az értesítési rendszer működésének tesztelése során ellenőrizzük, hogy az értesítések időben, megfelelő formában és tartalommal érkeznek-e. Vizsgáljuk, hogy a felhasználó által beállított eseménytípusok alapján történik-e az értesítés generálása, valamint hogy az értesítések kikapcsolása, szüneteltetése és újraaktiválása hibamentesen működik-e. Tesztfiókokkal szimuláljuk a különböző értesítési beállításokat, és figyeljük, hogy a rendszer megfelelően reagál-e az időjárási eseményekre.

### Fejlesztési környezet

Az időjárás-előrejelzés modul fejlesztése során korszerű, iparági szinten elismert eszközöket és technológiákat alkalmaztunk, amelyek biztosítják a hatékony munkavégzést, a kód minőségét, valamint a projekt átláthatóságát és fenntarthatóságát.

* **Fejlesztői eszközök:** A fejlesztéshez használt főbb eszközök közé tartozik a Visual Studio Code, amely modern, testreszabható fejlesztői környezetet biztosít.
* **Verziókezelés:** A projekt Git alapú verziókövetést használ, amely lehetővé teszi a fejlesztési folyamat strukturált nyomon követését. Külön ágakat hoztunk létre a frontend és backend fejlesztéshez, valamint a teszteléshez, így biztosítva a párhuzamos munkavégzést és a hibamentes integrációt. A commit üzenetek szabványos formátumban készülnek, és minden változtatás dokumentálva van.

## 1.3 Költségmegosztás funkció

### Célkitűzés

A költségmegosztás funkció célja, hogy csoportos utazások során biztosítsa a kiadások átlátható, igazságos és egyszerű elosztását a résztvevők között. Az utazás során gyakran előfordul, hogy a résztvevők közösen fizetnek különböző szolgáltatásokért – például szállásért, étkezésért, belépőkért vagy közlekedésért –, és ezeknek a költségeknek az utólagos elszámolása sokszor nehézkes, időigényes és konfliktusforrás lehet.

A rendszer célja, hogy ezt a folyamatot digitalizálja, automatizálja és átláthatóvá tegye. A modul lehetőséget nyújt a közös költségek tételes rögzítésére, azok automatikus szétosztására a résztvevők között, valamint a pénzügyi egyenleg folyamatos nyomon követésére. A felhasználók pontosan láthatják, hogy mennyit fizettek be, mennyi a rájuk eső rész, és mennyi az esetleges visszajáró összeg vagy tartozás.

Ezáltal elkerülhetők a félreértések, a pontatlan elszámolások, és minden utazó számára egyértelművé válik, hogy milyen mértékben járult hozzá az utazás költségeihez. A funkció nemcsak pénzügyi szempontból hasznos, hanem hozzájárul a csoportos utazások gördülékenyebb lebonyolításához, a bizalom fenntartásához és a közös élmények zavartalan megéléséhez.

A modul különösen hasznos lehet baráti társaságok, családok, céges csapatépítők vagy akár szervezett túrák esetén, ahol több résztvevő közösen vesz részt a költségek viselésében. A digitális elszámolás nemcsak gyorsabb, hanem pontosabb is, mint a hagyományos, papíralapú vagy szóbeli megállapodások.

### Funkcionális leírás

A költségmegosztás modul több egymással összefüggő funkciót tartalmaz, amelyek együttesen biztosítják a pénzügyi elszámolás teljes körű kezelését. A rendszer célja, hogy minden résztvevő számára egyértelmű, testreszabható és könnyen kezelhető megoldást nyújtson a közös kiadások kezelésére.

* **Költségek rögzítése:** A felhasználók tételesen rögzíthetik az utazás során felmerülő kiadásokat, például szállásfoglalás, étkezés, belépők, közlekedés, programok vagy egyéb szolgáltatások. Minden költséghez megadható a fizető személy, az összeg, a dátum, a költség típusa, valamint a résztvevők, akikre az adott tétel vonatkozik. A rendszer lehetőséget biztosít a költségek kategorizálására, így könnyen áttekinthetővé válik, hogy mely területeken merültek fel kiadások.
* **Automatikus megosztás:** A rendszer képes automatikusan szétosztani a rögzített költségeket a résztvevők között. Alapértelmezés szerint egyenlő arányban történik a megosztás, de lehetőség van egyéni beállításokra is, például ha valaki nem vett részt egy adott programon, vagy ha eltérő mértékben járult hozzá egy szolgáltatás igénybevételéhez. A megosztás logikája rugalmasan testreszabható, és a rendszer azonnal frissíti az egyenlegeket a módosítások alapján.
* **Egyéni hozzájárulások kezelése:** A modul támogatja az egyéni hozzájárulások kezelését is, ami különösen fontos fakultatív programok esetén. Például ha egy városnéző túrán csak néhány utazó vesz részt, akkor a költség csak közöttük oszlik meg. A rendszer lehetővé teszi, hogy minden költségtételhez külön meg lehessen adni, kik az érintett utazók, így az elszámolás pontos és igazságos marad.
* **Egyenlegkövetés:** Minden felhasználó számára elérhető egy egyenlegnézet, amely megmutatja, hogy mennyit fizetett be, mennyi a rá eső költség, és mennyi az aktuális egyenlege. Az egyenleg lehet pozitív (túlfizetés), negatív (tartozás) vagy nulla (kiegyenlített állapot). A rendszer folyamatosan frissíti az egyenlegeket, így a felhasználók mindig naprakész információval rendelkeznek.
* **Kifizetések rögzítése:** A rendszer nyilvántartja, ha valaki kiegyenlítette a tartozását, például készpénzes fizetéssel vagy banki átutalással. A kifizetések rögzítése után az egyenlegek automatikusan frissülnek, és a rendszer jelzi, ha minden utazó elszámolt a költségekkel. Ez különösen hasznos az utazás végén, amikor a végső elszámolás történik.
* **Exportálási lehetőség:** A költséglista és az egyenlegek letölthetők PDF formátumban, amely megkönnyíti az elszámolást, különösen akkor, ha hivatalos dokumentációra van szükség (pl. céges utazás esetén). Az exportált dokumentum tartalmazza a költségek részleteit, a megosztási arányokat, az egyenlegeket és a kifizetési státuszokat. A PDF formátum könnyen megosztható, nyomtatható és archiválható.

### Felhasználói felület és dizájn

A költségmegosztás modul felhasználói felülete intuitív, logikusan felépített és könnyen kezelhető, különösen csoportos használatra optimalizálva. A cél az volt, hogy a felhasználók gyorsan és egyszerűen tudják rögzíteni a közös kiadásokat, nyomon követni az egyenlegeket, valamint áttekinteni a teljes pénzügyi helyzetet az utazás során. A felület kialakítása során figyelembe vettük a modern webes dizájn elveit, a felhasználói szokásokat, valamint a mobilos és asztali eszközök közötti különbségeket.

A modul vizuális megjelenése és funkcionális elrendezése egyaránt támogatja a hatékony adatkezelést, miközben esztétikailag is illeszkedik az utazástervező rendszer egységes arculatához. A felület minden eleme reszponzív, azaz automatikusan alkalmazkodik a képernyőmérethez, így biztosítva a zökkenőmentes használatot bármilyen eszközön.

A főbb vizuális és funkcionális jellemzők az alábbiak szerint részletezhetők:

* **Színvilág:** A modul sötétkék háttérrel és világoskék kiemelésekkel jelenik meg, amely segíti a vizuális tagolást és a pénzügyi adatok gyors értelmezését. A sötétkék szín nyugalmat, stabilitást és megbízhatóságot sugall, míg a világoskék elemek kiemelik az aktuális, releváns információkat. A színpaletta kontrasztos, jól olvasható, és segíti a felhasználót abban, hogy első pillantásra felismerje a fontos adatokat, például a tartozásokat vagy a túlfizetéseket.
* **Tétellista nézet:** Minden költség külön sorban jelenik meg, megjelölve a fizető személyt, az összeget, a költség típusát, valamint az érintett utazókat. A lista rendezhető dátum, összeg vagy kategória szerint, és lehetőség van szűrésre is, például csak egy adott utazóhoz tartozó tételek megjelenítésére. A tétellista interaktív: a felhasználó módosíthatja, törölheti vagy új költséget adhat hozzá egyetlen kattintással.
* **Grafikus egyenlegdiagram:** A rendszer kördiagram vagy oszlopdiagram formájában jeleníti meg az egyéni hozzájárulásokat és tartozásokat. A diagramok színkódoltak, így könnyen értelmezhetők: például zöld szín jelzi a túlfizetést, piros a tartozást, kék az egyenleg kiegyenlítettségét. A diagramok interaktívak, az egérrel vagy érintéssel történő mozgás során megjelennek az adott utazóhoz tartozó pontos értékek. A grafikus megjelenítés segíti a csoportos elszámolás átláthatóságát, és vizuálisan is megerősíti az egyenlegek állapotát.
* **Csoportosítási lehetőség:** A költségek kategóriák szerint csoportosíthatók, például étkezés, szállás, közlekedés, programok, egyéb. Ez a funkció lehetővé teszi, hogy a felhasználók könnyen áttekintsék, mely területeken merültek fel a legnagyobb kiadások, és hol lehet esetleg optimalizálni. A kategóriák színkódoltak, és a rendszer automatikusan javaslatot tesz a kategóriákra a költségtípus alapján, de a felhasználó manuálisan is módosíthatja azokat.
* **Mobilbarát kialakítás:** A felület teljes mértékben reszponzív, azaz mobil eszközökön is gyorsan betölthető, és lehetővé teszi az adatok gyors rögzítését akár utazás közben is. A mobilos nézet optimalizált gombméreteket, egyszerűsített menüstruktúrát és érintésre érzékeny elemeket tartalmaz, amelyek megkönnyítik a navigációt. A rendszer támogatja a mobilos értesítéseket is, így a felhasználó azonnal értesülhet, ha új költség került rögzítésre, vagy ha változás történt az egyenlegében.

### Technikai megvalósítás

A költségmegosztás modul technikai háttere korszerű, megbízható és skálázható megoldásokra épül, amelyek biztosítják a rendszer stabil működését, a gyors adatfeldolgozást és a biztonságos felhasználói élményt. A fejlesztés során kiemelt figyelmet fordítottunk az adatbiztonságra, a modularitásra és az integrálhatóságra, hogy a rendszer hosszú távon is fenntartható és bővíthető legyen.

**A technikai komponensek az alábbiak szerint épülnek fel:**

* **Backend:** A szerveroldali működést egy Node.js alapú rendszer biztosítja, amely REST API-n keresztül kezeli a költségadatokat, felhasználói csoportokat és egyenlegeket. A backend felelős az adatok lekéréséért, feldolgozásáért, tárolásáért és továbbításáért a frontend felé. A REST API struktúrája jól dokumentált, és támogatja az aszinkron adatkezelést, amely lehetővé teszi a gyors válaszidőt és a skálázhatóságot.
* **Frontend:** A felhasználói felület React technológiával készült, amely lehetővé teszi a dinamikus tartalomkezelést és a gyors felhasználói interakciókat. A React komponensalapú felépítése elősegíti a moduláris fejlesztést, a könnyű karbantartást és az új funkciók gyors integrálását. A frontend felelős a költséglista megjelenítéséért, az egyenlegdiagramok rendereléséért, az interakciók kezeléséért, valamint a felhasználói beállítások mentéséért.
* **Adatbázis:** A rendszer MySQL-t használ az adatok tárolására, amely rugalmasan kezeli a költségek részleteit, a felhasználók közötti megosztási arányokat és a kifizetési státuszokat. A MySQL struktúra lehetővé teszi a gyors lekérdezéseket, az adatok hatékony indexelését és a skálázható adattárolást. Az adatbázisban tárolt információk titkosítva kerülnek mentésre, és a hozzáférés jogosultságokhoz kötött, így biztosítva az adatbiztonságot.
* **Hitelesítés:** A felhasználók biztonságos bejelentkezéssel (pl: OAuth2) csatlakozhatnak a csoporthoz, és csak az általuk létrehozott vagy meghívott utazásokhoz férnek hozzá. A hitelesítési rendszer támogatja a többfaktoros azonosítást, a jelszókezelést, valamint az adminisztrátori jogosultságok kezelését is. Ez biztosítja, hogy az adatokhoz csak az arra jogosult személyek férjenek hozzá, és minden művelet naplózásra kerüljön.

### Használhatóság és működés

A költségmegosztás modul használata során a felhasználók számára biztosított funkcionalitás lehetővé teszi, hogy az utazás bármely szakaszában – legyen az előkészület, aktív utazás vagy utólagos elszámolás – egyszerűen és gyorsan rögzítsenek új költségeket. A rendszer úgy lett kialakítva, hogy a felhasználók egyetlen felületen kezelhessék az összes pénzügyi adatot, miközben a háttérben automatikusan történik az egyenlegek kiszámítása és frissítése.

A költségek rögzítése során a felhasználó megadhatja a tétel pontos megnevezését, összegét, a fizető személyt, valamint azt, hogy kiket érint az adott kiadás. Ez lehetővé teszi, hogy a rendszer pontosan meghatározza, milyen arányban kell megosztani a költséget a résztvevők között. A felhasználó választhat egyenlő megosztást, vagy egyéni arányokat is beállíthat, például ha valaki nem vett részt egy adott programon, vagy ha eltérő mértékben járult hozzá a kiadáshoz.

A modul további lehetőséget biztosít arra is, hogy a csoport tagjai egymásnak kifizetéseket rögzítsenek. Ez különösen hasznos az utazás végén, amikor a résztvevők elszámolnak egymással, például készpénzes visszafizetések vagy banki átutalások formájában. A rendszer minden kifizetést naplóz, és automatikusan frissíti az egyenlegeket, így a pénzügyi állapot mindig naprakészen jelenik meg.

A felhasználói élmény szempontjából kiemelten fontos, hogy a modul gyorsan reagáljon, könnyen kezelhető legyen, és ne igényeljen technikai előképzettséget. A felület logikusan felépített, a műveletek egyértelműek, és a rendszer visszajelzései (pl. sikeres mentés, hibajelzés, egyenlegváltozás) azonnal megjelennek. A mobilbarát kialakításnak köszönhetően a felhasználók akár utazás közben, menet közben is rögzíthetik a költségeket, így nem kell utólag rekonstruálni a kiadásokat.

### Tesztelés és validálás

A költségmegosztás funkció megbízható működésének biztosítása érdekében elengedhetetlen a rendszer alapos tesztelése és validálása. A tesztelési folyamat célja, hogy feltárja az esetleges hibákat, ellenőrizze a számítási algoritmusok pontosságát, valamint igazolja, hogy a felhasználói interakciók zökkenőmentesen és hibamentesen zajlanak.

**A funkció tesztelése során az alábbi szempontokat vizsgáljuk:**

* **Költségelosztás pontossága:** A rendszernek pontosan kell kiszámítania az egyenlegeket, figyelembe véve az egyéni megosztási arányokat, a részleges részvételt, valamint a kifizetések időzítését. A tesztelés során különböző szimulált utazási forgatókönyveket alkalmazunk, amelyekben eltérő számú résztvevő, különböző típusú költségek és egyéni hozzájárulások szerepelnek. A cél az, hogy minden esetben helyes egyenlegszámítás történjen, és ne forduljon elő számítási hiba.
* **Felhasználói interakciók:** A költségek rögzítése, módosítása és törlése során a rendszernek gyorsan és hibamentesen kell működnie. A tesztelés során ellenőrizzük, hogy a felhasználói műveletek logikusan követik egymást, a felület megfelelő visszajelzést ad, és a mentések azonnal érvényesülnek. A tesztelés kiterjed a különböző eszközökön történő használatra, valamint a különböző böngészők kompatibilitására is.
* **Adatbiztonság:** A pénzügyi adatok védelme kiemelten fontos, ezért a rendszer titkosítással és jogosultságkezeléssel biztosítja, hogy az adatokhoz csak az arra jogosult felhasználók férjenek hozzá. A tesztelés során ellenőrizzük az adatátvitel biztonságát (HTTPS), a felhasználói hitelesítés működését (OAuth2), valamint a hozzáférési szintek helyes kezelését. A rendszer naplózza a műveleteket, így visszakövethető minden változtatás.
* **Mobiltesztelés:** A funkció mobil eszközökön is teljes funkcionalitással kell, hogy működjön. A tesztelés során Android és iOS platformokon ellenőrizzük a felület reszponzivitását, az interakciók gördülékenységét, valamint az értesítések megjelenését. A cél az, hogy a felhasználók mobilon is ugyanolyan hatékonyan tudják kezelni a költségeket, mint asztali gépen.
* **Exportálás tesztelése:** A PDF exportálás során a rendszernek pontosan kell tükröznie a rendszerben tárolt adatokat. A tesztelés során ellenőrizzük, hogy a letöltött dokumentum tartalmazza a költségek részleteit, a megosztási arányokat, az egyenlegeket és a kifizetési státuszokat. A PDF formátumnak jól strukturáltnak, olvashatónak és nyomtathatónak kell lennie.

### Fejlesztési környezet

A költségmegosztás modul fejlesztése során korszerű, iparági szinten elismert eszközöket és technológiákat alkalmaztunk, amelyek biztosítják a hatékony munkavégzést, a kód minőségét, valamint a projekt átláthatóságát és fenntarthatóságát.

* **Fejlesztői eszközök:** A fejlesztéshez használt főbb eszközök közé tartozik a Visual Studio Code, amely modern, testreszabható fejlesztői környezetet biztosít.
* **Verziókezelés:** A projekt Git alapú verziókövetést használ, amely lehetővé teszi a fejlesztési folyamat strukturált nyomon követését. Külön ágakat hoztunk létre a frontend és backend fejlesztéshez, valamint a teszteléshez, így biztosítva a párhuzamos munkavégzést és a hibamentes integrációt. A commit üzenetek szabványos formátumban készülnek, és minden változtatás dokumentálva van.

## 1.4 Fenntarthatósági kalkulátor

### Célkitűzés

A fenntarthatósági kalkulátor célja, hogy az utazástervezés során a felhasználók tudatos, környezetbarát döntéseket hozhassanak, figyelembe véve az utazásuk által okozott környezeti hatásokat. A modern turizmus egyik legnagyobb kihívása a fenntarthatóság biztosítása, mivel az utazások során jelentős mennyiségű szén-dioxid-kibocsátás és egyéb környezeti terhelés keletkezik. A kalkulátor célja, hogy ezt a láthatatlan hatást láthatóvá tegye, és a felhasználók számára kézzelfogható adatokat szolgáltasson az utazásuk ökológiai lábnyomáról.

A rendszer képes becslést adni az utazás karbonlábnyomára, azaz a közlekedési eszközök, szálláshelyek és választott programok alapján kiszámítja, hogy mekkora CO₂-kibocsátással jár az adott út. Ezen túlmenően a kalkulátor nemcsak informál, hanem aktívan segít a környezetbarát alternatívák kiválasztásában. A felhasználó javaslatokat kap arra vonatkozóan, hogyan csökkentheti az utazás környezeti terhelését – például választhat alacsony kibocsátású közlekedési módot, fenntartható szálláshelyet vagy természetközeli programokat.

A funkció célja tehát kettős: egyrészt edukálja a felhasználót a fenntartható utazás lehetőségeiről, másrészt konkrét eszközöket biztosít a környezeti hatás csökkentésére. A kalkulátor elősegíti a fenntartható turizmus elterjedését, és ösztönzi a felhasználókat arra, hogy felelős utazási szokásokat alakítsanak ki. Ez nemcsak a környezet védelme szempontjából fontos, hanem hozzájárulhat a helyi közösségek támogatásához, a természeti értékek megőrzéséhez, valamint a hosszú távon fenntartható turisztikai infrastruktúra kialakításához.

### Funkcionális leírás

A fenntarthatósági kalkulátor modul több egymással összefüggő funkciót tartalmaz, amelyek együttesen biztosítják a környezeti hatások átfogó értékelését és a fenntartható alternatívák bemutatását. A rendszer célja, hogy a felhasználó ne csupán információt kapjon, hanem aktívan részt vehessen a környezetbarát döntéshozatalban.

* **Karbonlábnyom kalkuláció:** A kalkulátor kiszámítja az utazás során keletkező becsült szén-dioxid-kibocsátást (CO₂), figyelembe véve a közlekedési eszközök típusát (pl. repülő, vonat, autó, busz), a szálláshelyek energiafelhasználását és fenntarthatósági minősítését, valamint a választott programok környezeti terhelését. A számítás alapjául nemzetközi szabványokon alapuló kibocsátási értékek szolgálnak, amelyeket a rendszer automatikusan frissít az aktuális adatok alapján. A kalkuláció eredménye egy összesített érték, amely megmutatja az utazás teljes karbonlábnyomát kilogrammban vagy tonnában kifejezve.
* **Közlekedési alternatívák összehasonlítása:** A rendszer lehetőséget biztosít arra, hogy a felhasználó különböző közlekedési módok között válasszon, és összehasonlítsa azok környezeti hatását. Például egy repülőút kibocsátása jelentősen magasabb lehet, mint egy vonatos vagy buszos utazásé. A kalkulátor vizuálisan is megjeleníti az egyes opciók közötti különbségeket, és javaslatot tesz a leginkább környezetbarát megoldásra. A rendszer figyelembe veszi az utazás távolságát, az utasok számát, valamint az adott közlekedési eszköz energiahatékonyságát.
* **Szállásminősítés fenntarthatóság szerint:** A kalkulátor kiemeli azokat a szálláshelyeket, amelyek zöld minősítéssel rendelkeznek, például energiatakarékos működésűek, helyi termékeket használnak, vagy környezetbarát szolgáltatásokat kínálnak. A rendszer integrálva van szállásfoglaló platformokkal, így a felhasználó közvetlenül láthatja, mely szálláshelyek felelnek meg a fenntarthatósági kritériumoknak. A minősítés alapjául nemzetközi tanúsítványok (pl. Green Key, EU Ecolabel) szolgálnak, és a rendszer részletes információt nyújt az egyes szálláshelyek környezeti teljesítményéről.
* **Programajánlás környezetbarát szempontok szerint:** A kalkulátor olyan programokat ajánl, amelyek alacsony környezeti terheléssel járnak, például gyalogtúrák, kerékpáros városnézések, helyi természeti értékek felfedezése, vagy kulturális események, amelyek nem igényelnek nagy energiafelhasználást. A rendszer figyelembe veszi az úti cél adottságait, a programok elérhetőségét, valamint a felhasználó érdeklődési körét. Az ajánlások célja, hogy a felhasználó ne mondjon le az élményekről, hanem környezetbarát módon élvezhesse az utazást.

### Felhasználói felület és dizájn

A fenntarthatósági kalkulátor felhasználói felülete vizuálisan letisztult, átgondoltan strukturált és környezettudatos megjelenésű, amely már első ránézésre is ösztönzi a felhasználót a zöldebb, felelősebb döntések meghozatalára. A dizájn célja, hogy a környezetbarát szemlélet ne csupán tartalmi szinten jelenjen meg, hanem vizuálisan is tükröződjön a felület minden elemében.

A felület kialakítása során kiemelt figyelmet kapott a színvilág, az interaktivitás, az információk vizuális megjelenítése, valamint a mobilos használhatóság. A modul minden eleme reszponzív, azaz automatikusan alkalmazkodik a különböző képernyőméretekhez, így biztosítva a zökkenőmentes használatot asztali gépen, táblagépen és okostelefonon egyaránt.

**A főbb vizuális és funkcionális jellemzők az alábbiak szerint részletezhetők:**

* **Színvilág:** A modul sötétkék háttérrel és világoskék, valamint zöld kiemelésekkel jelenik meg. A sötétkék háttér nyugalmat, stabilitást és megbízhatóságot sugall, míg a világoskék árnyalatok a semleges információkat emelik ki. A zöld szín a fenntartható ajánlásokat jelöli, például alacsony kibocsátású közlekedési módokat vagy környezetbarát szálláshelyeket. A színpaletta kontrasztos, jól olvasható, és segíti a felhasználót abban, hogy gyorsan felismerje a környezetbarát lehetőségeket. A színek harmonizálnak a modul céljával, és vizuálisan is megerősítik a fenntarthatósági üzenetet.
* **Interaktív űrlap:** A felhasználó egy könnyen kezelhető, lépésről lépésre vezetett űrlapon keresztül adhatja meg az utazás paramétereit. Ezek közé tartozik az indulási hely, a célállomás, a választott közlekedési mód, a szállás típusa, valamint a tervezett programok jellege. Az űrlap dinamikusan reagál a megadott adatokra, és azonnal frissíti a kalkulációs eredményeket. A beviteli mezők logikusan csoportosítottak, az egyes lépésekhez súgó szövegek és ikonok tartoznak, amelyek segítik az eligazodást. Az űrlap mobilon is könnyen kezelhető, az érintésre optimalizált elemek révén.
* **Karbonlábnyom diagram:** A kalkulátor az egyes komponensek – például közlekedés, szállás, programok – környezeti hatását grafikus formában jeleníti meg. A diagram lehet oszlopdiagram, kördiagram vagy területdiagram, attól függően, hogy a felhasználó milyen vizuális megjelenítést preferál. A diagramok színkódoltak, interaktívak, és lehetőséget biztosítanak az egyes elemek részletes megtekintésére. Az adatok mellett rövid szöveges magyarázat is megjelenik, amely segíti az értelmezést. A vizualizáció célja, hogy a felhasználó könnyen átlássa, mely tényezők járulnak hozzá leginkább az utazás karbonlábnyomához.
* **Ajánlási panel:** A kalkulátor egy külön ajánlási panelen keresztül alternatív lehetőségeket kínál, amelyekkel csökkenthető az utazás környezeti terhelése. Például javasolhatja a repülő helyett a vonat vagy busz használatát, a hagyományos hotel helyett eco-hostel választását, vagy a városi programok helyett természetközeli tevékenységeket. Az ajánlások személyre szabottak, figyelembe veszik a felhasználó által megadott paramétereket, valamint a célállomás adottságait. Az ajánlási panel vizuálisan elkülönül, zöld színnel kiemelve jelenik meg, és tartalmazza az alternatíva környezeti előnyeit is.
* **Mobilbarát kialakítás:** A felület teljes mértékben reszponzív, azaz mobil eszközökön is hibamentesen működik. Az űrlapok, diagramok és ajánlások automatikusan igazodnak a képernyőmérethez, így biztosítva a jó olvashatóságot és kezelhetőséget. A mobilos nézetben a menük egyszerűsítettek, a gombok nagyobb méretűek, és az érintésre optimalizált interakciók révén a felhasználó könnyedén navigálhat az adatok között. A rendszer támogatja a mobilos értesítéseket is, így a felhasználó azonnal értesülhet, ha új ajánlás érhető el vagy ha változás történt a kalkulációban.

### Technikai megvalósítás

A fenntarthatósági kalkulátor technikai háttere korszerű, megbízható és skálázható megoldásokra épül, amelyek biztosítják a rendszer stabil működését, a gyors adatfeldolgozást és a biztonságos felhasználói élményt. A fejlesztés során kiemelt figyelmet fordítottunk az adatbiztonságra, a modularitásra és az integrálhatóságra, hogy a rendszer hosszú távon is fenntartható és bővíthető legyen.

**A technikai komponensek az alábbiak szerint épülnek fel:**

* **Backend:** A szerveroldali működést egy Node.js alapú rendszer biztosítja, amely REST API-n keresztül végzi a számításokat és ajánlásokat. A backend felelős az adatok lekéréséért, feldolgozásáért, tárolásáért és továbbításáért a frontend felé. A REST API struktúrája jól dokumentált, és támogatja az aszinkron adatkezelést, amely lehetővé teszi a gyors válaszidőt és a skálázhatóságot. A szerveroldali logika tartalmazza a karbonlábnyom kalkulációs algoritmusokat, az ajánlási szabályokat, valamint az adatnaplózást.
* **Frontend:** A felhasználói felület React technológiával készült, amely lehetővé teszi a dinamikus tartalomkezelést és a gyors felhasználói interakciókat. A React komponensalapú felépítése elősegíti a moduláris fejlesztést, a könnyű karbantartást és az új funkciók gyors integrálását. A frontend felelős az űrlapok megjelenítéséért, a diagramok rendereléséért, az ajánlások vizuális megjelenítéséért, valamint a felhasználói beállítások mentéséért.
* **Adatbázis:** A rendszer MySQL-t használ az adatok tárolására, amely rugalmasan kezeli a felhasználói beállításokat, kalkulációs előzményeket és ajánlási statisztikákat. A MySQL
* struktúra lehetővé teszi a gyors lekérdezéseket, az adatok hatékony indexelését és a skálázható adattárolást. Az adatbázisban tárolt információk titkosítva kerülnek mentésre, és a hozzáférés jogosultságokhoz kötött, így biztosítva az adatbiztonságot.
* **Külső API integrációk:** A rendszer több külső szolgáltatással van összekötve, amelyek biztosítják a kalkulációk és ajánlások pontosságát:
* **Carbon Interface API:** Ez az API szolgáltatja a CO₂-kibocsátás számításához szükséges alapértékeket különböző közlekedési módokra, szállástípusokra és tevékenységekre vonatkozóan. A rendszer az API segítségével valós idejű adatokat kér le, és ezek alapján végzi el a karbonlábnyom kalkulációt.
* **Green Travel Index:** Egy fenntarthatósági adatbázis, amely zöld minősítéssel rendelkező szálláshelyeket, programokat és szolgáltatásokat tartalmaz. A kalkulátor ebből az adatbázisból nyeri azokat az ajánlásokat, amelyek környezetbarát alternatívaként jelennek meg a felhasználó számára.
* **OpenStreetMap:** A közlekedési útvonalak és távolságok lekérdezéséhez használt térképes adatforrás. A rendszer az indulási és célállomás alapján kiszámítja az utazás távolságát, és ennek megfelelően becsüli meg a közlekedési kibocsátást. Az OpenStreetMap integráció lehetővé teszi az útvonalak vizuális megjelenítését is, amely segíti a felhasználót az alternatívák összehasonlításában.

### Használhatóság és működés

A fenntarthatósági kalkulátor használata során a felhasználó az utazás megtervezésének kezdetén megadja azokat az alapvető paramétereket, amelyek az utazás környezeti hatásának kiszámításához szükségesek. Ezek a paraméterek magukban foglalják az indulási helyet, a célállomást, a választott közlekedési módot (pl. repülő, vonat, autó, busz), a szállás típusát (pl. hotel, apartman, eco-hostel), valamint a tervezett programok jellegét (pl. városnézés, túrázás, kulturális események).

A rendszer ezeket az adatokat automatikusan feldolgozza, és a háttérben lefuttatja a karbonlábnyom kalkulációs algoritmust, amely a megadott információk alapján becslést ad az utazás teljes CO₂-kibocsátására. A kalkuláció eredménye egy összesített érték, amelyet a rendszer vizuálisan is megjelenít, például kördiagram vagy oszlopdiagram formájában. A felhasználó így könnyen átláthatja, hogy mely tényezők járulnak hozzá leginkább az utazás környezeti terheléséhez.

A kalkulátor ezután javaslatokat tesz a környezetbarát alternatívákra. Ezek az ajánlások személyre szabottak, figyelembe veszik a felhasználó által megadott paramétereket, valamint a célállomás földrajzi és infrastrukturális adottságait. Például, ha a felhasználó repülőutat tervez, a rendszer javasolhatja a vonat vagy busz használatát, ha elérhető. Ha a szállás hagyományos hotel, a kalkulátor ajánlhat eco-hostelt vagy zöld minősítéssel rendelkező szálláshelyet. A programok esetében a rendszer előnyben részesíti a természetközeli, alacsony kibocsátású tevékenységeket.

A felhasználó számára lehetőség nyílik arra is, hogy a kalkulátor által javasolt alternatívák alapján módosítsa az útvonalat vagy a választott szolgáltatásokat. A rendszer azonnal újraszámolja a karbonlábnyomot, és megmutatja, hogyan változik az utazás környezeti hatása a módosítások hatására. Ez a visszacsatolás segíti a tudatos döntéshozatalt, és ösztönzi a felhasználót a fenntarthatóbb opciók választására.

### Tesztelés és validálás

A fenntarthatósági kalkulátor megbízható működésének biztosítása érdekében a fejlesztési folyamat során kiemelt figyelmet fordítottunk a funkciók alapos tesztelésére és validálására. A cél az volt, hogy a rendszer pontos, releváns és felhasználóbarát módon működjön, miközben megfelel a nemzetközi környezetvédelmi elvárásoknak.

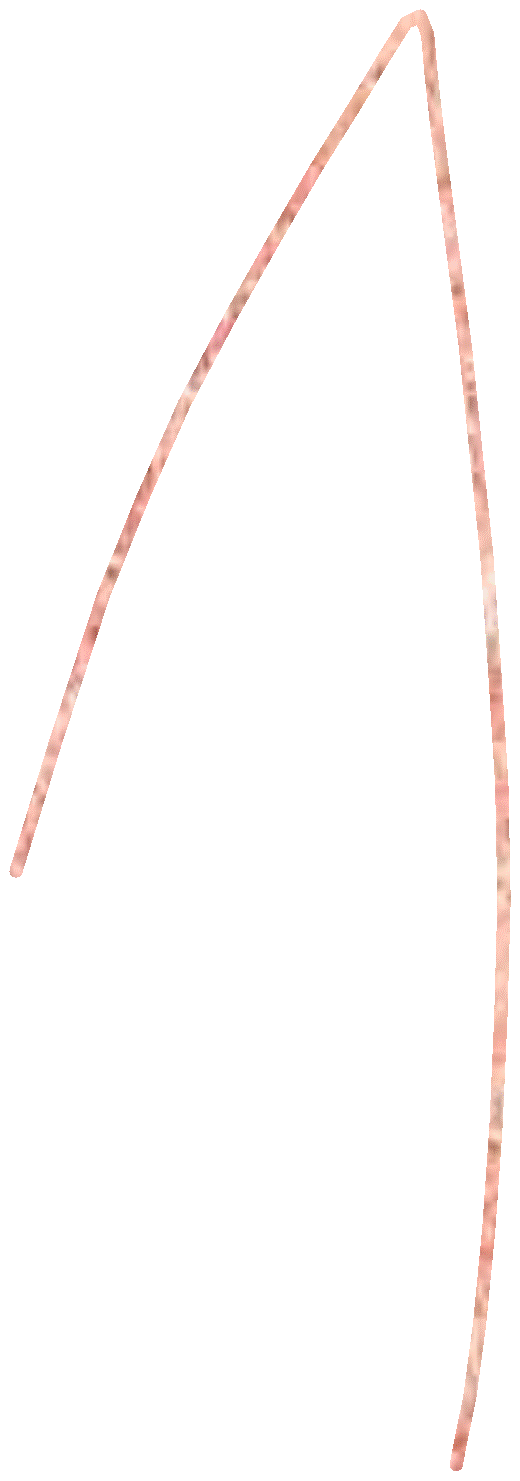
**A tesztelés során az alábbi szempontokat vizsgáljuk:**

* **Számítási pontosság:** A kalkulált értékeknek meg kell felelniük a nemzetközi szabványoknak, valamint a külső API-k által szolgáltatott adatoknak. A tesztelés során összehasonlító elemzéseket végzünk, ahol a rendszer által generált karbonlábnyom értékeket összevetjük hiteles forrásokkal. A cél az, hogy a kalkulációk ne csak becslések legyenek, hanem valós, megbízható adatokon alapuló eredmények.
* **Ajánlások relevanciája:** A rendszer által kínált környezetbarát alternatíváknak valóban relevánsnak és megvalósíthatónak kell lenniük. A tesztelés során ellenőrizzük, hogy az ajánlások figyelembe veszik-e a felhasználó preferenciáit, az úti cél adottságait, valamint a szolgáltatások elérhetőségét. A cél az, hogy a javaslatok ne legyenek általánosak, hanem valóban segítő szándékúak és személyre szabottak.
* **Felhasználói interakciók:** Az űrlapok kitöltése során a rendszernek gyorsan és hibamentesen kell működnie. A tesztelés során vizsgáljuk, hogy a beviteli mezők logikusan vannak-e elrendezve, a súgók és ikonok segítik-e az eligazodást, valamint hogy a diagramok jól értelmezhetők-e. A cél az, hogy a felhasználó technikai előképzettségtől függetlenül könnyedén tudja használni a kalkulátort.
* **Mobiltesztelés:** A funkció mobil eszközökön is teljes funkcionalitással kell, hogy működjön. A tesztelés során Android és iOS platformokon ellenőrizzük a felület reszponzivitását, az interakciók gördülékenységét, valamint az értesítések megjelenését. A cél az, hogy a felhasználók mobilon is ugyanolyan hatékonyan tudják használni a kalkulátort, mint asztali gépen.
* **Adatbiztonság:** A felhasználói adatok védelme kiemelten fontos, ezért a rendszer titkosítással és jogosultságkezeléssel biztosítja, hogy az adatokhoz csak az arra jogosult felhasználók férjenek hozzá. A tesztelés során ellenőrizzük az adatátvitel biztonságát (HTTPS), a felhasználói hitelesítés működését, valamint a hozzáférési szintek helyes kezelését. A rendszer naplózza a műveleteket, így visszakövethető minden változtatás.

### Fejlesztési környezet

A fenntarthatósági kalkulátor fejlesztése során korszerű, iparági szinten elismert eszközöket és technológiákat alkalmaztunk, amelyek biztosítják a hatékony munkavégzést, a kód minőségét, valamint a projekt átláthatóságát és fenntarthatóságát.

* **Fejlesztői eszközök:** A fejlesztéshez használt főbb eszközök közé tartozik a Visual Studio Code, amely modern, testreszabható fejlesztői környezetet biztosít.
* **Verziókezelés:** A projekt Git alapú verziókövetést használ, amely lehetővé teszi a fejlesztési folyamat strukturált nyomon követését. Külön ágakat hoztunk létre a frontend és backend fejlesztéshez, valamint a teszteléshez, így biztosítva a párhuzamos munkavégzést és a hibamentes integrációt. A commit üzenetek szabványos formátumban készülnek, és minden változtatás dokumentálva van.

**Bevezetés a projektbe**

## 1.1 A fejlesztési feladat célja

Az utazási weboldal elsődleges célja, hogy a felhasználók számára kényelmes és átlátható módon tegye elérhetővé a szállásfoglalási lehetőségeket, valamint bemutassa a közelükben található látnivalókat és útvonalakat. A weboldal nem csupán egy szálláskereső felület, hanem egy komplex turisztikai platform, amely támogatja a felhasználót utazásának megtervezésében, a programok összeállításában, valamint a helyi nevezetességek felfedezésében.

A fejlesztési feladat azokra az intelligens funkciókra és technikai megoldásokra összpontosít, amelyek kifejezetten a felhasználói élmény javítását szolgálják. A cél az, hogy a rendszer:

* **személyre szabott ajánlatokat** nyújtson az előzmények alapján,
* **összehangolt programokat** ajánljon a nyitvatartások figyelembevételével,
* **többnyelvű hozzáférést** biztosítson a felülethez és a tartalmakhoz,
* valamint **offline is működőképes** maradjon a letöltött térképek és útvonalak révén.

Ezek a funkciók nemcsak a felhasználói élményt teszik gördülékenyebbé, hanem hozzájárulnak ahhoz is, hogy a rendszer versenyképes legyen más, hasonló utazási platformokkal szemben.

## 1.2 Az intelligens funkciók szerepe a weboldalon

Az „intelligens funkciók” kifejezés ebben a projektben nem gépi tanuláson alapuló megoldásokat jelent, hanem olyan logikai és technikai fejlesztéseket, amelyek:

* **figyelembe veszik a felhasználói előzményeket** (pl. cookie-k alapján),
* **időbeli optimalizációt kínálnak** előre rögzített adatok felhasználásával,
* **nyelvi sokszínűséget biztosítanak** több előre elkészített oldalváltozat segítségével.

Az intelligens funkciók tehát az interaktivitást és a személyre szabott élményt támogatják, miközben megőrzik a rendszer egyszerűségét és stabilitását.

## 1.3 A technikai fejlesztések jelentősége

A technikai fejlesztések közül az egyik legfontosabb az **offline navigáció** lehetősége. A turizmusban sok esetben előfordul, hogy a felhasználónak nincs elérhető internetkapcsolata (például külföldön, vidéki helyszíneken vagy túraútvonalakon). Ennek áthidalására a rendszer lehetővé teszi, hogy a felhasználók **előre letölthessék a térképeket és a statikus útvonalakat**, így azok offline is hozzáférhetők maradnak.

Ez a funkció komoly hozzáadott értéket képvisel, mivel:

* biztonságosabbá teszi az utazást,
* növeli a felhasználói élményt,
* és előnyt biztosít más online-only szolgáltatásokkal szemben.

## 1.4 Dokumentáció felépítése

A jelen dokumentáció kifejezetten az intelligens funkciók és a technikai fejlesztések kidolgozására koncentrál. A tartalom a következő fő részekből áll:

* **2. fejezet – Intelligens tervezési lehetőségek**: bemutatja a személyre szabott ajánlatok, az időoptimalizálás és a nyelvi támogatás működését.
* **3. fejezet – Technikai fejlesztések**: ismerteti az offline navigáció koncepcióját és megvalósítási lehetőségeit.
* **4. fejezet – Adatkezelés és architektúra háttér**: bemutatja a MySQL, Node.js, Express és React szerepét a funkciók megvalósításában.
* **5. fejezet – Tesztelési terv**: részletezi, hogyan ellenőrizhetők a funkciók megbízhatósága és teljesítménye.
* **6. fejezet – Összefoglalás és jövőbeli bővítések**: röviden bemutatja a fejlesztés lehetséges irányait.

# **Intelligens tervezési lehetőségek**

# **2.1 Ajánlott útvonalak és látnivalók (bővített változat)**

### 2.1.1 Cookie-k és felhasználói előzmények kezelése

A cookie-k használata alapvető szerepet játszik az ajánlott tartalmak előállításában. Mivel nem gépi tanulásról, hanem egyszerű **előzmény-alapú szűrésről és rangsorolásról** van szó, a cookie-k és a felhasználói fiókok együttesen biztosítják a szükséges adatokat.

**Cookie-k tartalma lehet például:**

* Felhasználó utolsó keresett helyszíne (lastLocation)
* Gyakran keresett kategóriák (preferredCategory)
* Oldalnyelv (lang)
* Megtekintett szállások és látnivalók listája (history)

**Adattárolás:**  
A cookie-k rövid távú tárolást biztosítanak. Amennyiben a felhasználó be van jelentkezve, a cookie tartalma **szinkronizálódik a MySQL adatbázisban található history táblával**.

**Példa cookie-tartalom JSON formátumban:**

{

"lastLocation": "Balatonfüred",

"preferredCategory": "múzeum",

"history": [

{"type": "accommodation", "id": 1023},

{"type": "attraction", "id": 451}

],

"lang": "hu"

}

Ez az egyszerű struktúra már elegendő alapot ad az ajánlati logikának.

### 2.1.2 Ajánlati logika (statikus adatbázis + személyre szabás)

Az ajánlatok kiszolgálása három fő komponensből áll:

1. **Alapadatbázis (MySQL)**
   * accommodations tábla: szálláshelyek adatai
   * attractions tábla: látnivalók adatai
   * routes tábla: előre definiált útvonalak
2. **Backend logika (Node.js + Express)**
   * API végpont, amely feldolgozza a cookie-információkat
   * relevancia pontszám számítása
   * JSON lista küldése a frontend felé
3. **Frontend megjelenítés (React)**
   * rangsorolt lista megjelenítése
   * vizuális kiemelés: „Ajánlott neked” címkék

**Adatbázis szerkezet (részlet az attractions táblából):**

| **id** | **name** | **category** | **location** | **opening\_hours** | **description** |
| --- | --- | --- | --- | --- | --- |
| 451 | Jókai Mór Emlékház | múzeum | Balatonfüred | 09:00-17:00 | Klasszikus irodalmi múzeum |
| 452 | Tagore sétány | park | Balatonfüred | 00:00-24:00 | Népszerű vízparti sétány |

**Relevancia számítás pszeudokód:**

function calculateRelevance(attraction, cookieData):

score = 1

if attraction.location == cookieData.lastLocation:

score += 2

if attraction.category == cookieData.preferredCategory:

score += 3

if attraction.id in cookieData.history:

score += 1

return score

A backend minden látnivalóra lefuttatja ezt a logikát, majd **pontszám szerint sorba rendezi** a találatokat.

### 2.1.3 Példa felhasználói forgatókönyv

**Felhasználó A**

* Utolsó keresett helyszín: Balatonfüred
* Kedvelt kategória: múzeum
* Cookie adatai: {lastLocation: "Balatonfüred", preferredCategory: "múzeum"}

**Ajánlati sorrend:**

1. Balatonfüredi múzeumok (magas pontszám a kategória + hely miatt)
2. Kulturális útvonalak Balatonfüreden
3. Általános látnivalók a környéken
4. Országos turisztikai ajánlatok

### 2.1.4 Folyamatleírás

A személyre szabott ajánlások létrehozásának folyamata:

1. **Felhasználó belép az oldalra**
   * A frontend lekéri a cookie-k tartalmát.
2. **Frontend API-hívást indít**
   * A React alkalmazás a cookie-adatokat is elküldi a backendnek.
3. **Backend feldolgozás**
   * A Node.js alapú API betölti a statikus adatbázist.
   * Minden látnivalóra kiszámolja a relevancia pontszámot.
4. **Rangsorolt lista visszaadása**
   * JSON formátumban érkezik a React frontendhez.
5. **Megjelenítés**
   * A frontend vizuálisan kiemeli a legrelevánsabb ajánlatokat.

### 2.1.5 Diagram (leírás)

**Adatáramlási folyamatábra:**

[Felhasználó]

↓ (cookie-k)

[Frontend - React]

↓ (API hívás cookie adatokkal)

[Backend - Node.js + Express]

↓ (lekérdezés)

[MySQL adatbázis]

↑ (adatok)

[Backend relevancia számítás]

↑ (JSON lista)

[Frontend megjelenítés]

Ez a folyamat egyszerű, gyors, és nem igényel komplex gépi tanulást, mégis képes **személyre szabott ajánlatokat adni**.

### 2.1.6 REST API végpontok

Az ajánlott útvonalak és látnivalók szolgáltatásához a backend (Node.js + Express) REST API-t biztosít. Az API feladata, hogy feldolgozza a cookie-információkat, lekérdezze az adatbázist, majd relevancia sorrendben adja vissza az eredményeket.

**Példavégpontok:**

1. **Látnivalók lekérése**

GET /api/attractions

Leírás: lekéri az összes látnivalót, opcionálisan szűrés paraméterekkel.

Paraméterek (query string):

* location – városnév (pl. Balatonfüred)
* category – látnivaló típusa (pl. múzeum)
* lang – nyelvi beállítás (hu, en, de)

Válasz (JSON példa):

[

{

"id": 451,

"name": "Jókai Mór Emlékház",

"category": "múzeum",

"location": "Balatonfüred",

"opening\_hours": "09:00-17:00",

"relevance": 6

},

{

"id": 452,

"name": "Tagore sétány",

"category": "park",

"location": "Balatonfüred",

"opening\_hours": "00:00-24:00",

"relevance": 3

}

]

1. **Személyre szabott ajánlások lekérése**

GET /api/recommendations

Leírás: a cookie-információk (pl. kedvenc kategória, legutóbbi helyszín) figyelembevételével adja vissza a látnivalókat és útvonalakat relevancia sorrendben.

Válasz (JSON példa):

{

"recommended": [

{

"id": 451,

"name": "Jókai Mór Emlékház",

"category": "múzeum",

"location": "Balatonfüred",

"relevance": 8,

"tag": "Ajánlott neked"

},

{

"id": 452,

"name": "Tagore sétány",

"category": "park",

"location": "Balatonfüred",

"relevance": 5

}

]

}

### 2.1.7 Adatbázis-lekérdezés példák (MySQL)

A backend a MySQL adatbázisból szerzi az adatokat.

**1. Alap lekérdezés az összes látnivalóra:**

SELECT id, name, category, location, opening\_hours

FROM attractions;

**2. Szűrés hely és kategória alapján:**

SELECT id, name, category, location, opening\_hours

FROM attractions

WHERE location = 'Balatonfüred'

AND category = 'múzeum';

**3. Relevancia számítás (egyszerűsített SQL példa):**

SELECT

id,

name,

category,

location,

opening\_hours,

(CASE WHEN location = 'Balatonfüred' THEN 2 ELSE 0 END +

CASE WHEN category = 'múzeum' THEN 3 ELSE 0 END) AS relevance

FROM attractions

ORDER BY relevance DESC;

### 2.1.8 Frontend integráció (React példa)

A React komponens az ajánlásokat API-hívással szerzi meg, majd kiemeli azokat.

**Példa React kódrészlet:**

import React, { useEffect, useState } from "react";

function Recommendations() {

const [recommendations, setRecommendations] = useState([]);

useEffect(() => {

fetch("/api/recommendations")

.then(res => res.json())

.then(data => setRecommendations(data.recommended));

}, []);

return (

<div>

<h2>Ajánlott látnivalók</h2>

<ul>

{recommendations.map(item => (

<li key={item.id}>

<strong>{item.name}</strong> ({item.category}) - {item.location}

{item.tag && <span className="highlight"> {item.tag}</span>}

</li>

))}

</ul>

</div>

);

}

export default Recommendations;

**Eredmény a felhasználónak:**

* a „Jókai Mór Emlékház” múzeum kiemelve jelenik meg „Ajánlott neked” címkével.
* más látnivalók is listázva vannak, de alacsonyabb prioritással.

### 2.1.9 Kihívások és megoldások

1. **Cookie-adatok kezelése:**
   * Kihívás: a cookie mérete korlátozott (~4KB).
   * Megoldás: csak az alap preferenciákat tároljuk cookie-ban, a részletes előzmények a history táblában lesznek.
2. **Relevancia pontszámítás:**
   * Kihívás: túl bonyolult logika lassíthatja a lekérdezést.
   * Megoldás: egyszerű szabályalapú pontozás, amely MySQL CASE utasításokkal könnyen kezelhető.
3. **Többnyelvű tartalom kezelése:**
   * Kihívás: minden nyelvhez külön fordított változatot kell tárolni.
   * Megoldás: az adatbázisban minden rekordhoz kapcsolódik egy nyelvi mező (pl. name\_hu, name\_en).

### 2.1.10 Összefoglalás

Az ajánlott útvonalak és látnivalók funkció egy **cookie-alapú személyre szabási rendszeren** nyugszik. Bár egyszerű szabályok mentén működik, a felhasználó számára mégis azt az élményt adja, hogy a rendszer ismeri az érdeklődését, és ennek megfelelően ajánl. Ez különösen fontos turisztikai környezetben, ahol a felhasználó gyorsan szeretne dönteni a számára legvonzóbb programok közül.

# **2.2 Időoptimalizálás**

Az időoptimalizálás célja, hogy a felhasználó a rendelkezésére álló időt a lehető leghatékonyabban használja fel. Mivel az oldal statikus útvonalakat és előre megadott látnivalókat használ, a rendszer nem valós idejű forgalmi adatokat dolgoz fel, hanem **nyitvatartási adatokra és előre rögzített útvonal-időkre** támaszkodik.

### 2.2.1 A funkció alapelve

A felhasználó kiválaszt egy vagy több látnivalót, majd a rendszer kiszámítja, hogy ezek milyen sorrendben látogathatók meg úgy, hogy:

* figyelembe veszi a **nyitvatartási időket**,
* számol az **átlagos utazási időkkel** a pontok között,
* elkerüli azokat a sorrendeket, ahol valamelyik látnivaló zárva lenne a tervezett érkezéskor.

### 2.2.2 Adatok a MySQL adatbázisban

Az időoptimalizálás két kulcsfontosságú adatra épül:

1. **Nyitvatartás** (opening\_hours oszlop az attractions táblában)
   * formátum: HH:MM-HH:MM (pl. 09:00-17:00)
   * speciális érték: 00:00-24:00 (napi 24 órás látnivaló)
2. **Utazási idő** (routes tábla)
   * minden útvonal két pont között tárolva van percekben.

**Példa routes tábla:**

| **id** | **from\_id** | **to\_id** | **travel\_time\_min** |
| --- | --- | --- | --- |
| 1 | 451 | 452 | 15 |
| 2 | 452 | 453 | 30 |

Ez a struktúra lehetővé teszi, hogy a backend gyorsan kiszámolja a sorrendet.

### 2.2.3 Algoritmus működése

1. **Kiindulási adatok összegyűjtése**
   * Felhasználó indulási ideje (pl. 10:00)
   * Kiválasztott látnivalók listája
2. **Lehetséges sorrendek generálása**
   * A backend megnézi az összes lehetséges sorrendet (pl. 3 látnivaló → 6 lehetséges sorrend).
3. **Érvényesség ellenőrzése**
   * A rendszer kiszámolja, hogy az adott sorrend esetén mikorra érkezik a felhasználó a látnivalóhoz.
   * Ha a nyitvatartási időn kívül érkezne, a sorrend érvénytelen.
4. **Legjobb sorrend kiválasztása**
   * Az a sorrend nyer, amelyben minden látnivaló látogatható, és a teljes idő a legrövidebb.

### 2.2.4 Példa forgatókönyv

**Felhasználó adatai:**

* Indulási idő: 10:00
* Kiválasztott látnivalók:
  + [451] Jókai Mór Emlékház (nyitvatartás: 09:00–17:00)
  + [452] Tagore sétány (nyitvatartás: 00:00–24:00)
  + [453] Vaszary Galéria (nyitvatartás: 11:00–18:00)

**Útvonal-idők:**

* Emlékház → Sétány: 15 perc
* Sétány → Galéria: 20 perc
* Emlékház → Galéria: 10 perc

**Algoritmus lefutása:**

1. Indulás 10:00 → Emlékház (nyitva → ok) → 10:00–11:00 látogatás
2. Utazás Emlékház → Galéria (10 perc) → érkezés 11:10 → nyitva → ok
3. Galéria látogatás 11:10–12:30
4. Utazás Galéria → Sétány (20 perc) → érkezés 12:50 → nyitva (24/7)

**Optimális sorrend:**  
Emlékház → Galéria → Sétány

### 2.2.5 REST API végpont

POST /api/optimized-route

**Request body (példa):**

{

"startTime": "10:00",

"selectedAttractions": [451, 452, 453]

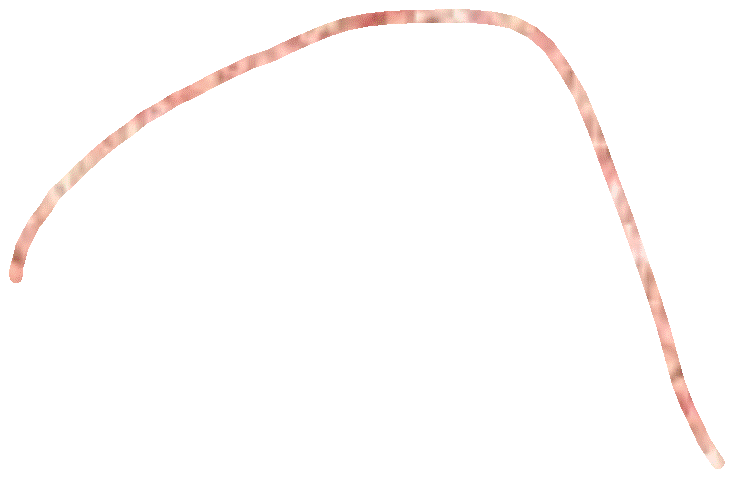
}

**Response (példa):**

{

"optimizedOrder": [

{"id": 451, "name": "Jókai Mór Emlékház", "arrival": "10:00", "departure": "11:00"},

 {"id": 453, "name": "Vaszary Galéria", "arrival": "11:10", "departure": "12:30"},

{"id": 452, "name": "Tagore sétány", "arrival": "12:50", "departure": "13:30"}

],

"totalTime": "3h 30m"

}

### 2.2.6 Frontend megjelenítés (React)

A React felületen az időoptimalizált sorrend vizuálisan is látszik:

* idővonal komponens (pl. 10:00 → 13:30),
* minden látnivalóhoz **érkezési idő és távozási idő** feltüntetve,
* piros figyelmeztetés, ha egy látnivaló a kiválasztott sorrendben nem lenne látogatható.

**Egyszerű React komponens példa:**

function OptimizedRoute({ route }) {

return (

<div>

<h2>Optimális útvonal</h2>

<ul>

{route.map(stop => (

<li key={stop.id}>

{stop.name} – Érkezés: {stop.arrival}, Távozás: {stop.departure}

</li>

))}

</ul>

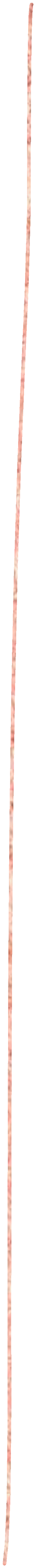
</div>

);

}

### 2.2.7 Kihívások és megoldások

1. **Pontatlan utazási idők**
   * Kihívás: nincs valós idejű forgalmi adat.
   * Megoldás: statikus, átlagos utazási idők tárolása a routes táblában.
2. **Túl sok kombináció**
   * Kihívás: több látnivaló esetén a sorrendek száma exponenciálisan nő.
   * Megoldás: a rendszer max. 5 látnivalót enged optimalizálni egyszerre, ami kezelhető számítási szinten.
3. **Felhasználói eltérések**
   * Kihívás: egyes felhasználók rövidebb/hosszabb időt töltenek egy látnivalónál.
   * Megoldás: a rendszer alapértelmezett tartózkodási időt vesz figyelembe (pl. múzeum: 60 perc, park: 30 perc), de ez a felületen módosítható.



### 2.2.8 Összefoglalás

Az időoptimalizálás funkció **egyszerű szabályokra épül**, de a felhasználói élményt nagyban növeli:

* elkerülhetők a zárva tartó helyek,
* csökken az üresjárat és a fölösleges utazás,
* a felhasználó **kész, időben összehangolt útvonalat** kap.

## 2.2.9 Algoritmus részletes leírása

Az útvonal időoptimalizálás **permutációs keresésen** alapul. Ha a felhasználó n látnivalót választott, a rendszer az összes n! sorrendet megvizsgálja. Mivel a kombinációk száma gyorsan nő, a rendszer maximum **5 látnivalót** enged egyszerre optimalizálni.

**Lépések:**

1. **Bemeneti adatok:**
   * indulási időpont (startTime)
   * kiválasztott látnivalók listája (selectedAttractions[])
   * statikus routes tábla (utazási idők percekben)
   * statikus attractions tábla (nyitvatartás + alap tartózkodási idő)
2. **Minden sorrend generálása**
   * pl. ha a felhasználó 3 látnivalót választ, akkor 6 sorrend (3!) jön létre.
3. **Érvényesség ellenőrzése**
   * minden sorrendre kiszámítja az érkezési és távozási időt, és összeveti a nyitvatartással.
   * ha valamelyik pontnál a nyitvatartás miatt nem lehet belépni, az útvonal **érvénytelen**.
4. **Legjobb sorrend kiválasztása**
   * ha több érvényes sorrend van, akkor azt választja, amelyik a **legrövidebb teljes időtartamot** eredményezi.

## 2.2.10 Példa algoritmus futása (3 látnivaló)

### Adatok:

* Indulás: 10:00
* Látnivalók:
  + A = Jókai Mór Emlékház (09:00–17:00, tartózkodási idő: 60 perc)
  + B = Vaszary Galéria (11:00–18:00, tartózkodási idő: 80 perc)
  + C = Tagore sétány (00:00–24:00, tartózkodási idő: 30 perc)

### Útvonal-idők:

* A → B = 10 perc
* A → C = 15 perc
* B → C = 20 perc

### Permutációk:

1. A → B → C
   * Érkezés A: 10:00 (nyitva) → távozás 11:00
   * Utazás B-re: 10 perc → érkezés 11:10 (nyitva) → távozás 12:30
   * Utazás C-re: 20 perc → érkezés 12:50 (nyitva) → távozás 13:20 ✅ Érvényes
2. A → C → B
   * Érkezés A: 10:00 (nyitva) → távozás 11:00
   * Utazás C-re: 15 perc → érkezés 11:15 → távozás 11:45
   * Utazás B-re: 20 perc → érkezés 12:05 (nyitva) → távozás 13:25 ✅ Érvényes
3. B → A → C
   * Érkezés B: 10:00 (zárva 11:00-ig) ❌ Érvénytelen
4. B → C → A
   * B: érkezés 10:00 → zárva ❌ Érvénytelen
5. C → A → B
   * C: 10:00–10:30
   * Utazás A-ra: 15 perc → érkezés 10:45 (nyitva) → távozás 11:45
   * Utazás B-re: 10 perc → érkezés 11:55 (nyitva) → távozás 13:15 ✅ Érvényes
6. C → B → A
   * C: 10:00–10:30
   * Utazás B-re: 20 perc → érkezés 10:50 (zárva 11:00-ig) ❌ Érvénytelen

### Optimális sorrend: ****A → B → C (3h 20m teljes idő)****

## 2.2.11 Folyamatdiagram (szöveges leírás)

Ha ezt vizualizálnánk, így nézne ki az ábra:

[FELHASZNÁLÓ KIVÁLASZTJA LÁTNIVALÓKAT]

↓

[BACKEND: összes lehetséges sorrend generálása]

↓

[MINDEN SORRENDRE]

├─► kiszámít érkezési időt

├─► ellenőrzi nyitvatartást

├─► érvénytelen sorrend → eldob

└─► érvényes sorrend → ment

↓

[ÉRVÉNYES SORRENDEK KÖZÜL LEGJOBB KIVÁLASZTÁSA]

↓

[OPTIMÁLIS ÚTVONAL JSON formátumban vissza a FRONTENDNEK]

↓

[REACT FRONTEND: idővonalas megjelenítés]

## 2.2.12 Node.js backend pszeudokód

app.post("/api/optimized-route", async (req, res) => {

const { startTime, selectedAttractions } = req.body;

// 1. Lekérés adatbázisból

const attractions = await db.query(

"SELECT \* FROM attractions WHERE id IN (?)",

[selectedAttractions]

);

const routes = await db.query("SELECT \* FROM routes");

// 2. Permutációk generálása

const permutations = generatePermutations(selectedAttractions);

let bestRoute = null;

let shortestTime = Infinity;

for (let order of permutations) {

const schedule = calculateSchedule(order, attractions, routes, startTime);

if (schedule.valid && schedule.totalTime < shortestTime) {

bestRoute = schedule;

shortestTime = schedule.totalTime;

}

}

res.json(bestRoute);

});

A calculateSchedule() függvény számolja az érkezési időket és ellenőrzi a nyitvatartást.

## 2.2.13 További felhasználói forgatókönyvek

1. **Felhasználó délután indul (pl. 15:00)**
   * Az algoritmus automatikusan kizárja azokat a sorrendeket, ahol a múzeum zárva lenne mire odaér.
2. **Felhasználó csak kültéri látnivalókat választ (park, sétány, strand)**
   * Mivel ezek 24 órásak, a sorrend szinte mindig érvényes → a rendszer a legrövidebb útvonalat választja.
3. **Felhasználó sok látnivalót jelöl meg (pl. 8 helyszín)**
   * A backend figyelmezteti: „Túl sok látnivalót választottál az optimalizáláshoz, maximum 5 helyszín kezelhető egyszerre.”

## 2.2.14 Kihívások mélyebben

* **Skálázhatóság:**  
  Ha a felhasználók egyszerre több optimalizálást futtatnak, a permutáció-számítás terheli a szervert.  
  → **Megoldás:** előre kiszámított és gyorsítótárazott (cache) útvonalak gyakori kombinációkra.
* **Felhasználói élmény:**  
  A felhasználónak tudnia kell, miért az adott sorrend az optimális.  
  → **Megoldás:** frontendben megjeleníteni a kiszámított érkezési időket és a nyitvatartási ellenőrzést.
* **Adatpontosság:**  
  Ha változik egy látnivaló nyitvatartása, a rendszernek frissíteni kell az adatbázist.  
  → **Megoldás:** adminfelületen keresztül könnyen szerkeszthetők a nyitvatartások.

## 2.2.15 Összefoglalás

Az időoptimalizálás modul egy **statikus, de hatékony megoldás**:

* a felhasználó nem csak egy listát kap, hanem egy **időben reális sorrendet**,
* az algoritmus egyszerű, de elegendő pontosságú turisztikai célokra,
* REST API és React integrációval könnyen bővíthető és vizualizálható.

**2.3 Nyelvi támogatás**

**2.3.1 A funkció célja**

A turisztikai weboldal egyik fő feladata, hogy nemzetközi felhasználók számára is könnyen elérhető és érthető legyen. A **nyelvi támogatás** biztosítja, hogy a felhasználók:

* a teljes felületet a saját nyelvükön használhassák,
* a látnivalókhoz és szállásokhoz kapcsolódó információkat anyanyelvükön olvashassák,
* váltani tudjanak a támogatott nyelvek között (pl. magyar, angol, német).

**2.3.2 Működési elv**

A rendszer **nem használ dinamikus fordítóprogramokat (pl. Google Translate)**, hanem minden tartalom előre lefordított változatban kerül rögzítésre. Ez biztosítja:

* a **fordítás minőségét és hitelességét**,
* a **gyors betöltést**, hiszen nincs szükség külső API-hívásokra,
* az **egységes terminológiát** az egész oldalon.

**2.3.3 Nyelvi adatok tárolása (MySQL)**

Az adatbázisban minden szöveges mező **nyelvenként külön oszlopban** tárolódik.

**Példa: attractions tábla**

| **id** | **name\_hu** | **name\_en** | **name\_de** | **description\_hu** | **description\_en** | **description\_de** |
| --- | --- | --- | --- | --- | --- | --- |
| 451 | Jókai Mór Emlékház | Jókai Mór Memorial | Jókai Mór Gedenkhaus | Klasszikus múzeum... | Classic museum... | Klassisches Museum... |
| 452 | Tagore sétány | Tagore Promenade | Tagore-Promenade | Népszerű sétány... | Popular promenade... | Beliebte Promenade... |

**Előny:** egyszerű kezelhetőség, gyors szűrés az adott nyelv szerint.  
**Hátrány:** új nyelv hozzáadása új oszlopokat igényel.

**2.3.4 Nyelvi beállítás kezelése**

A felhasználó nyelvi preferenciáját két helyen tároljuk:

1. **Cookie** – rövid távú, gyors hozzáférés (pl. {lang: "en"})
2. **Felhasználói fiók** – hosszú távú beállítás (pl. users.language\_preference)

**Prioritás:**

* ha a felhasználó be van jelentkezve → a fiókban beállított nyelv az irányadó,
* ha nincs bejelentkezve → a cookie tartalma dönti el a nyelvet,
* ha nincs cookie → alapértelmezett nyelv a magyar.

**2.3.5 REST API működés nyelvek szerint**

Az API minden kérésnél figyelembe veszi a nyelvi beállítást, és annak megfelelő mezőket ad vissza.

**Végpont példa:**

GET /api/attractions?lang=en

**Válasz:**

[

{

"id": 451,

"name": "Jókai Mór Memorial",

"description": "Classic museum dedicated to the Hungarian writer...",

"opening\_hours": "09:00-17:00"

}

]

**2.3.6 Frontend működés (React)**

A React alkalmazás a nyelvi beállítást cookie-ból vagy felhasználói profilból szerzi meg, és API-hívásoknál paraméterként továbbítja.

**Példa React kód:**

function AttractionList({ lang }) {

const [attractions, setAttractions] = useState([]);

useEffect(() => {

fetch(`/api/attractions?lang=${lang}`)

.then(res => res.json())

.then(data => setAttractions(data));

}, [lang]);

return (

<div>

<h2>{lang === "hu" ? "Látnivalók" : lang === "en" ? "Attractions" : "Sehenswürdigkeiten"}</h2>

<ul>

{attractions.map(item => (

<li key={item.id}>

<strong>{item.name}</strong> – {item.description}

</li>

))}

</ul>

</div>

);

}

**2.3.7 Nyelvváltás folyamata**

1. Felhasználó a weboldal fejlécében nyelvválasztót használ (pl. magyar, angol, német zászló ikon).
2. A választott nyelv elmentődik a cookie-ba és (ha van fiók) az adatbázisba.
3. A frontend új API-hívást küld az adott nyelvi paraméterrel.
4. A backend az adott nyelvi mezőket küldi vissza.
5. A React újrarendereli a tartalmat a választott nyelven.

**2.3.8 Példa felhasználói forgatókönyv**

**Felhasználó A (magyar turista):**

* alapértelmezett nyelv: hu
* az oldal magyarul jelenik meg, a múzeum leírása: *„Klasszikus múzeum...”*

**Felhasználó B (német turista):**

* nyelvet vált de-re
* cookie frissül: {lang: "de"}
* API válasz a name\_de és description\_de mezőkkel jön vissza
* a felület németül jelenik meg: *„Klassisches Museum...”*

**2.3.9 Kihívások és megoldások**

1. **Többnyelvű adat karbantartása**
   * Kihívás: minden változtatást több nyelven is át kell vezetni.
   * Megoldás: adminfelület, ahol egy látnivaló összes nyelvi verziója együtt szerkeszthető.
2. **Új nyelv hozzáadása**
   * Kihívás: az adatbázisban új oszlopokat kell létrehozni.
   * Megoldás: előre megtervezett adatbázis-struktúra, amely támogatja a bővíthetőséget.
3. **Felhasználói élmény**
   * Kihívás: a nyelvváltásnak azonnal látszania kell.
   * Megoldás: React automatikus újrarendereléssel frissíti a tartalmat.

**2.3.10 Összefoglalás**

A nyelvi támogatás megoldása **stabil, gyors és megbízható**, mert:

* nincs külső fordító API-ra való támaszkodás,
* minden fordítás előre lektorált és egységes,
* a felhasználó bármikor válthat a támogatott nyelvek között.

Ez a megközelítés ideális turisztikai weboldalak számára, ahol a **pontosság és a kulturális érzékenység** különösen fontos.

# **2.3 Nyelvi támogatás (bővített változat)**

## 2.3.11 UI-tervezés

A többnyelvű támogatás felhasználói élmény szempontjából akkor hatékony, ha:

* **jól látható nyelvváltó elem** van (általában a jobb felső sarokban),
* a nyelvválasztás **ikonokkal (zászlók) + rövid nyelvkódokkal** történik (pl. 🇭🇺 HU, 🇬🇧 EN, 🇩🇪 DE),
* a váltás **azonnali**, nincs szükség oldal-újratöltésre.

**Szöveges mockup példa a fejlécből:**

-------------------------------------------

LOGO | Főoldal | Szállások | Látnivalók |

-------------------------------------------

🌐 HU | EN | DE

**Mobilnézetben**: a nyelvváltó egy „hamburger” menüben található.

## 2.3.12 Nyelvváltás folyamatábra (szöveges ábra)

[FELHASZNÁLÓ NYELVVÁLASZTÓT KATTINT]

↓

[FRONTEND: lang cookie frissítése]

↓

[REACT új API hívás: /api/attractions?lang=EN]

↓

[BACKEND: kiválasztja az adott nyelvű mezőket az adatbázisból]

↓

[API JSON válasz EN nyelven]

↓

[REACT újrarendereli a tartalmat angolul]

## 2.3.13 Adatbázis-struktúra alternatívák

### A) Oszloponként nyelv (egyszerű megoldás)

* name\_hu, name\_en, name\_de
* Könnyen érthető, de új nyelvnél új oszlopokat kell létrehozni.

### B) Külön translations tábla (rugalmasabb megoldás)

**Táblák:**

* attractions
  + id | default\_name
* translations
  + id | entity\_type | entity\_id | lang | field | text

**Példa tartalom:**

| **id** | **entity\_type** | **entity\_id** | **lang** | **field** | **text** |
| --- | --- | --- | --- | --- | --- |
| 1 | attraction | 451 | hu | name | Jókai Mór Emlékház |
| 2 | attraction | 451 | en | name | Jókai Mór Memorial |
| 3 | attraction | 451 | de | name | Jókai Mór Gedenkhaus |
| 4 | attraction | 451 | hu | description | Klasszikus múzeum... |
| 5 | attraction | 451 | en | description | Classic museum... |

**Előny:** könnyen bővíthető új nyelvvel.  
**Hátrány:** összetettebb SQL-lekérdezések.

## 2.3.14 Adminfelület-funkciók

Az adminfelület kulcsfontosságú a többnyelvű tartalom karbantartásában.

**Funkciók:**

1. **Nyelvi szerkesztő** – egy látnivaló adatlapján minden nyelvi mező együtt szerkeszthető.
2. **Kötelező mezők figyelése** – ha egy nyelvnél hiányzik a fordítás, figyelmeztetés jelenik meg.
3. **Tömeges import/export** – CSV/Excel fájlból egyszerre több fordítás feltölthető.
4. **Nyelvi verzió előnézet** – admin bármikor megnézheti, hogyan néz ki az oldal adott nyelven.

## 2.3.15 Plusz felhasználói forgatókönyvek

### Forgatókönyv 1: Külföldi turista érkezése

* Felhasználó először nyitja meg az oldalt → alapértelmezett nyelv: magyar.
* Böngészője Accept-Language header alapján jelzi, hogy angol preferált nyelv.
* Backend felismeri, és automatikusan angol verziót küld → felhasználó azonnal érti a tartalmat.

### Forgatókönyv 2: Vegyes nyelvű társaság

* Egy család német turistákból és magyar barátokból áll.
* A német felhasználó DE nyelvre vált, és a cookie elmenti.
* A magyar barát bejelentkezik a saját fiókjával → ott HU van alapértelmezetten.
* A rendszer mindenkinél külön kezeli a nyelvi preferenciát.

### Forgatókönyv 3: Új nyelv bevezetése

* Az oldal üzemeltetői szeretnének olasz nyelvet (IT).
* Admin új nyelvet engedélyez → adatbázisban új oszlopok vagy translations rekordok.
* Az adminfelületen lefordítják a főbb attrakciókat.
* Olasz turisták mostantól IT nyelvet választhatnak a menüből.

## 2.3.16 Kihívások és fejlesztési irányok

1. **Skálázhatóság**
   * Ha az oldal 8-10 nyelvet támogat, az oszlopos megoldás kezelhetetlenné válik.
   * **Megoldás:** áttérés a translations táblás megközelítésre.
2. **Felhasználói élmény**
   * Fontos, hogy a nyelvváltás ne csak a szöveget, hanem a dátumformátumot, pénznem kijelzést is változtassa.
   * **Megoldás:** i18n könyvtárak használata Reactben (pl. react-intl, i18next).
3. **Adatfrissítés és konzisztencia**
   * Ha új tartalom kerül fel magyarul, annak lefordítása késhet → hiányos nyelvi élmény.
   * **Megoldás:** az adminfelület mutassa, mely fordítások hiányoznak.

## 2.3.17 Összefoglalás

A nyelvi támogatás modul biztosítja, hogy a weboldal:

* **többnyelvűen működjön**,
* a tartalmak **előre hitelesen lefordítva** álljanak rendelkezésre,
* a nyelvváltás gyors és kényelmes legyen,
* a rendszer rugalmasan bővíthető legyen új nyelvekkel.

Ez különösen fontos egy turisztikai platformnál, ahol a célközönség **nemzetközi** és a nyelvi akadályok megszüntetése kulcs a felhasználói élményhez.

**3.1 Offline navigáció**

**3.1.1 A funkció célja**

A turisták gyakran olyan helyen utaznak, ahol:

* nincs folyamatos mobilinternet,
* a roaming díjak magasak,
* vagy egyszerűen szeretnék spórolni az adatforgalmat.

Az **offline navigáció** biztosítja, hogy:

* a kiválasztott térképeket és útvonalakat a felhasználó előre letölthesse,
* internet nélkül is lássa a főbb látnivalókat és szálláshelyeket,
* az előre megtervezett útvonalat követni tudja valós időben.

**3.1.2 Működési elv**

Az offline navigáció alapja:

1. **Előre generált, statikus térképrészletek** (pl. JSON/GeoJSON adatok).
2. **Látnivalók és útvonalak letöltése a felhasználó eszközére**.
3. **Progresszív Web App (PWA) technológia** a cache és offline működés biztosítására.

**Példa folyamat:**

* Felhasználó kiválasztja Balatonfüred térségét.
* Rendszer előállít egy letölthető csomagot:
  + térképadatok (GeoJSON),
  + szállások és látnivalók metaadatai,
  + előre definiált útvonalak.
* Felhasználó letölti és elmenti az eszközére.
* Később, internet nélkül is böngészheti a térképet és az ajánlott útvonalakat.

**3.1.3 Használt technológiák**

* **Node.js + Express REST API**
  + letölthető csomagok előkészítése és kiszolgálása.
* **React frontend (PWA támogatással)**
  + Service Worker kezeli a cache-t és az offline elérhetőséget.
* **MySQL**
  + látnivalók és útvonalak tárolása, amelyekből offline csomag készül.
* **GeoJSON**
  + térképadatok egyszerű, szabványos formátumban.

**3.1.4 Offline csomag felépítése**

Az offline letöltés egy tömörített fájl (.zip), amely tartalmazza:

* map.geojson – a térség térképi adatai (utcák, utak, látnivalók).
* attractions.json – látnivalók listája többnyelvű leírással.
* routes.json – előre definiált útvonalak (szállás → látnivaló).
* metadata.json – metaadatok (verzió, utolsó frissítés dátuma).

**Példa: attractions.json**

[

{

"id": 451,

"name": { "hu": "Jókai Mór Emlékház", "en": "Jókai Mór Memorial" },

"description": { "hu": "Klasszikus múzeum...", "en": "Classic museum..." },

"coordinates": [46.9592, 17.8713]

}

]

**3.1.5 Letöltés és frissítés folyamata**

1. Felhasználó a weboldalon kiválasztja: „Térség letöltése offline használatra”.
2. Backend elkészíti a csomagot és letöltésre kínálja.
3. React frontend elmenti a csomagot **IndexedDB-be** vagy **LocalStorage-be**.
4. Következő megnyitásnál, ha nincs internet:
   * Service Worker először az offline cache-ből tölti be az adatokat.

**Frissítés:**

* Ha van internetkapcsolat, a rendszer összeveti a metadata.json verzióját.
* Ha újabb, akkor automatikusan frissíti az offline adatokat.

**3.1.6 REST API példa**

**Végpont:**

GET /api/offline-package?region=balatonfured&lang=hu

**Válasz:**

* .zip fájl, benne map.geojson, attractions.json, routes.json.

**3.1.7 Frontend működés (React + PWA)**

**Service Worker cache kezelés (pszeudokód):**

self.addEventListener("fetch", event => {

event.respondWith(

caches.match(event.request).then(response => {

return response || fetch(event.request);

})

);

});

**React komponens:**

function OfflineMap({ region }) {

const [mapData, setMapData] = useState(null);

useEffect(() => {

// offline tárolt adat betöltése

const data = localStorage.getItem(`offline-${region}`);

if (data) {

setMapData(JSON.parse(data));

} else {

fetch(`/api/offline-package?region=${region}`)

.then(res => res.json())

.then(data => {

setMapData(data);

localStorage.setItem(`offline-${region}`, JSON.stringify(data));

});

}

}, [region]);

return mapData ? <MapComponent data={mapData} /> : <p>Betöltés...</p>;

}

**3.1.8 Felhasználói forgatókönyvek**

1. **Külföldi turista mobilnet nélkül**
   * Előre letölti Balatonfüred csomagját Wi-Fi-n.
   * Másnap internet nélkül is látja a térképet és a látnivalókat.
2. **Felhasználó rossz térerőnél**



* + Kirándulás közben megszakad az internet.
  + A PWA cache-ből tölti be a térképet → nincs akadozás.

1. **Új látnivaló hozzáadása a rendszerhez**
   * Admin frissíti az adatbázist.
   * Új csomag készül → felhasználó letöltéskor automatikusan a friss verziót kapja.

**3.1.9 Kihívások és megoldások**

1. **Tárhely korlát**
   * Mobil eszközön a cache mérete véges.
   * Megoldás: a felhasználó csak adott térséget tölthet le, nem az egész országot.
2. **Adatok naprakészsége**
   * Offline adatok elavulhatnak (pl. változik nyitvatartás).
   * Megoldás: minden induláskor frissítés ellenőrzése (metadata.json alapján).
3. **Platformfüggőség**
   * PWA viselkedés eltérhet iOS és Android között.
   * Megoldás: tesztelés mindkét platformon, és külön figyelmeztetés a felhasználónak.

**3.1.10 Összefoglalás**

Az offline navigáció modul biztosítja, hogy a felhasználók internet nélkül is hozzáférjenek a legfontosabb információkhoz.

* **Letölthető csomagok** révén a térképek, látnivalók és útvonalak mindig kéznél vannak.
* **PWA és cache** megoldások gondoskodnak a zavartalan használatról.
* Ez a funkció különösen hasznos nemzetközi turisták számára, akik korlátozott adatkapcsolattal utaznak.

# **3.1 Offline navigáció (bővített változat)**

## 3.1.11 React térkép-megjelenítés (UI mockup)

A letöltött térképadatokat a frontend egy térképböngésző komponenssel jeleníti meg. Erre alkalmasak pl.:

* **Leaflet.js** (nyílt forráskódú, könnyű integráció Reacthez),
* **Mapbox GL JS** (profibb, de licencfeltételekhez kötött),
* **react-leaflet** (React wrapper Leaflethez).

**Szöveges UI mockup:**

-------------------------------------

| [ BALATONFÜRED ] |

| |

| 🏠 Szálláshely A |

| 🏛 Látnivaló 1 |

| 🌳 Látnivaló 2 |

| |

| [Letöltött térkép] |

| • Láthatók az utcák |

| • Ikonok jelzik a látnivalókat |

| • Vastag vonal = útvonal |

-------------------------------------

| Gombok: [Útvonal megjelenítése] |

| [Követés GPS-szel] |

-------------------------------------

**React példa kód (Leaflet):**

import { MapContainer, TileLayer, Marker, Popup, Polyline } from "react-leaflet";

function OfflineMap({ attractions, routes }) {

return (

<MapContainer center={[46.9592, 17.8713]} zoom={14} style={{ height: "400px" }}>

<TileLayer url="/offline-tiles/{z}/{x}/{y}.png" /> {/\* offline csempe \*/}

{attractions.map(a => (

<Marker key={a.id} position={a.coordinates}>

<Popup>

<strong>{a.name.hu}</strong><br />

{a.description.hu}

</Popup>

</Marker>

))}

{routes.map((r, i) => (

<Polyline key={i} positions={r.coordinates} />

))}

</MapContainer>

);

}

## 3.1.12 Felhasználói élmény (UX) részletezése

Az offline térkép használatánál a legfontosabb szempontok:

* **Könnyű letöltés**  
  – Felhasználó egy gombnyomással letöltheti az adott régiót.
* **Átlátható tárhely-kezelés**  
  – Az app mutatja, mekkora helyet foglal az offline csomag.
* **Egyszerű frissítés**  
  – Ha van újabb verzió, értesítést kap: „Új térkép verzió érhető el, frissíted most?”
* **Gyors hozzáférés offline módban**  
  – Ha nincs internet, a felhasználó nem kap hibát, hanem az offline csomagból dolgozik.

## 3.1.13 Biztonsági és adatkezelési szempontok

Az offline adatok letöltése biztonsági kockázatokat is hordozhat, ezért:

* **Adatintegritás ellenőrzése**  
  – A csomag tartalmaz egy SHA-256 hash értéket, amivel ellenőrizhető, hogy nem sérült vagy módosított.
* **Felhasználói adatok védelme**  
  – A letöltött csomag csak statikus turisztikai adatokat tartalmaz, személyes adatokat nem.
* **Hozzáférés-kezelés**  
  – Csak bejelentkezett felhasználók tölthetnek le offline csomagot, így biztosítva a hozzáférés kontrollját (opcionális).

## 3.1.14 Fejlesztési javaslatok az offline navigációhoz

1. **Részleges letöltés**  
   – Felhasználó kiválaszthatja, mely látnivalókat vagy útvonalakat szeretné offline módra elmenteni.
2. **Intelligens frissítés**  
   – Ne kelljen mindig a teljes csomagot újratölteni, csak azokat a fájlokat, amelyek változtak.
3. **GPS-alapú útvonal követés**  
   – Ha az eszközön engedélyezett a GPS, akkor offline módban is követni lehet, hogy a felhasználó éppen hol jár a térképen.
4. **Kombinált online-offline működés**  
   – Ha van internet, a térkép online frissül. Ha nincs, automatikusan visszavált az offline változatra.

## 3.1.15 Haladó forgatókönyv: útvonal offline követése GPS-szel

### Forgatókönyv:

1. Felhasználó letölti a **Balatonfüred városnéző útvonal** offline csomagját.
2. Másnap sétálni indul, internet nélkül.
3. Az alkalmazás:
   * megnyitja a letöltött térképet,
   * a GPS segítségével mutatja az aktuális pozícióját,
   * vastag vonallal jelzi a kiválasztott útvonalat,
   * értesítést ad: „500 méterre vagy a következő látnivalótól: Vaszary Galéria.”

### React pszeudokód GPS integrációval:

function GpsTracker() {

const [position, setPosition] = useState(null);

useEffect(() => {

if (navigator.geolocation) {

navigator.geolocation.watchPosition(

pos => setPosition([pos.coords.latitude, pos.coords.longitude])

);

}

}, []);

return position ? (

<Marker position={position}>

<Popup>Te itt vagy</Popup>

</Marker>

) : <p>GPS helyzet betöltése...</p>;

}

## 3.1.16 Útvonalak offline elérhetősége

Az utazási weboldal egyik kiemelt intelligens funkciója az útvonalak offline elérhetősége. Ez a megoldás különösen fontos olyan turisták számára, akik külföldön tartózkodnak, ahol az internetkapcsolat korlátozott vagy drága lehet, illetve olyan régiókban, ahol a hálózati lefedettség gyenge. Az offline hozzáférés lehetővé teszi, hogy a felhasználó előre letöltse a szükséges útvonalakat, és azokat később bármikor, internet nélkül is használhassa.

### Funkcionális követelmények

1. **Előre definiált útvonalak tárolása**
   * Az adatbázisban minden útvonal statikusan van rögzítve.
   * Egy útvonal több állomásból áll, amelyek sorrendben követik egymást.
   * Az állomások mindig látnivalókhoz (attractions) vannak kötve.
2. **Offline csomagok letöltése**
   * A felhasználó egy teljes régióhoz tartozó látnivalókat és útvonalakat letölthet.
   * A csomag tartalmazza:
     + az adott régió összes útvonalát,
     + a kapcsolódó látnivalókat,
     + a térképrészletet, amely az útvonalak vizualizációjához szükséges,
     + a nyelvi változatokat a kiválasztott nyelveken.
3. **Offline megjelenítés**
   * Az offline elérhető adatok a böngészőben cache-elve vagy mobilalkalmazásban helyben tárolva érhetők el.
   * A felhasználó internet nélkül is navigálhat az előre letöltött útvonalakon.
   * A GPS integráció biztosítja, hogy az alkalmazás képes legyen valós időben jelezni a felhasználó aktuális helyzetét az útvonalhoz képest.

### Technikai megvalósítás

Az offline elérhetőséghez a rendszer az alábbi megoldásokat alkalmazza:

* **Adatcsomag struktúra**  
  Az offline csomagokat ZIP formátumban tároljuk. A csomag tartalmaz:
  + metadata.json: a régió neve, verziószám, frissítés dátuma.
  + routes.json: a régió útvonalainak leírása.
  + attractions.json: a régióhoz tartozó látnivalók listája.
  + translations.json: többnyelvű fordítások.
  + tiles/: a térképcsempék könyvtára.
* **MySQL és REST API kapcsolata**
  + Az adatokat a backend (Node.js + Express) gyűjti össze az adatbázisból.
  + A /api/offline-package?region=... végponton keresztül a szerver összeállítja és letöltésre kínálja a régióhoz tartozó csomagot.
* **Frontend oldali működés (React)**
  + A felhasználó egy **"Letöltés offline használatra"** gombbal tudja elmenteni a csomagot.
  + A böngészőben a Service Worker és IndexedDB gondoskodik a csomag lokális tárolásáról.
  + Az alkalmazás automatikusan felismeri, hogy a felhasználó offline módban van, és a lokális adatokhoz nyúl.

### Felhasználói élmény

Az offline elérhetőség legfontosabb célja a zavartalan felhasználói élmény biztosítása.

* A turista még internet nélkül is megtalálja az útvonalát.
* A rendszer előre kiszámolt időadatokat biztosít, így a programtervezés offline módban is folytatható.
* A nyelvi tartalmak letöltésével a felhasználó a számára megfelelő nyelven kapja meg az útvonal és látnivalók leírását.

### Példa forgatókönyv

Egy turista Budapestre érkezik, és előre letölti a „Budapest belvárosi séta” útvonalat. A csomag tartalmazza a főbb látnivalókat (pl. Parlament, Szent István Bazilika, Lánchíd), valamint a hozzájuk tartozó rövid leírásokat magyarul, angolul és németül.

* Amikor a turista offline lesz (például egy aluljáróban vagy ha elfogy a mobiladata), a rendszer automatikusan átvált offline módba.
* A GPS segítségével az alkalmazás mutatja, hogy a turista éppen melyik ponton tart az útvonalon.
* A látnivalókhoz tartozó előre letöltött szövegek továbbra is elérhetők.

### Előnyök

* **Folyamatos hozzáférés** internet nélkül is.
* **Költségcsökkentés**, mivel nincs szükség mobiladatra külföldön.
* **Biztonság**, mert a felhasználónak nem kell újra és újra letöltenie ugyanazt az információt.

## 3.1.17 Online–offline mód közötti váltás

Az utazási weboldal egyik kulcseleme a rugalmas működés **változó hálózati körülmények között**. A felhasználó számára ugyanis alapvető elvárás, hogy a szolgáltatás **mindig elérhető legyen**, függetlenül attól, hogy van-e aktív internetkapcsolata. A rendszer ezért automatikusan képes váltani **online** és **offline mód** között.

### Online mód

Az online mód biztosítja a **legfrissebb adatok elérését** a szerverről.

* **Előnyök**:
  + Az adatbázisban történt legutóbbi módosítások azonnal elérhetők (pl. új látnivaló vagy új nyelvi változat).
  + A felhasználó több régió közül választhat, és újakat is letölthet offline használatra.
  + Az ajánlások mindig a legaktuálisabb előzmények alapján készülnek.
* **Működés**:
  + Az Express REST API biztosítja az adatlekérdezéseket.
  + A React frontend AJAX/Fetch kérésekkel folyamatosan frissíti a megjelenített tartalmakat.
  + A cookie-k frissítése minden interakciónál megtörténik, így az ajánlórendszer is azonnal reagál.

### Offline mód

Offline módban a felhasználó kizárólag a **korábban letöltött csomagokkal** dolgozik.

* **Előnyök**:
  + Az utazó akkor is navigálhat, ha nincs internet.
  + A rendszer továbbra is biztosítja a GPS-alapú helymeghatározást.
  + A nyelvi támogatás offline is működik, mivel a fordítások a letöltött csomagban elérhetők.
* **Működés**:
  + A Service Worker interceptálja a hálózati kéréseket, és ha nincs kapcsolat, a helyi IndexedDB vagy cache tartalmát adja vissza.
  + Az útvonalak és látnivalók JSON fájlokból töltődnek be (routes.json, attractions.json).
  + Az alkalmazás felismeri a hálózat hiányát, és értesíti a felhasználót egy egyszerű üzenettel: „Offline módban vagy – a letöltött adatok alapján folytatjuk a navigációt.”

### Automatikus váltás

A rendszer egyik legnagyobb erőssége az, hogy **a felhasználónak nem kell kézzel állítania a módot** – az automatikus felismerés gondoskodik róla.

1. **Internet elérhetőségének figyelése**
   * A böngésző navigator.onLine API-ját használjuk.
   * Ha az érték változik, esemény (online, offline) váltja ki a módcsere folyamatot.
2. **Adatok frissítése visszatéréskor**
   * Amikor a felhasználó újra online lesz, a rendszer ellenőrzi, van-e frissebb offline csomag.
   * Ha igen, felajánlja a frissítést.
3. **Felhasználói élmény**
   * A váltás zökkenőmentes, a felhasználónak nem kell semmit tennie.
   * Az UI-ban egy kis ikon (pl. zöld = online, szürke = offline) jelzi az aktuális állapotot.

### Példa forgatókönyv

1. Egy turista elindul Budapesten, és online módban használja az oldalt.
2. A metróban megszűnik az internetkapcsolat.
   * Az alkalmazás automatikusan offline módra vált.
   * A korábban letöltött útvonalak és térképcsempék betöltődnek.
   * A felhasználó folytathatja a sétát a Parlament felé.
3. Amikor feljön a felszínre, a telefon újra kapcsolódik az internethez.
   * Az alkalmazás érzékeli az állapotváltozást.
   * Ellenőrzi, hogy elérhető-e újabb verzió az offline csomagból.
   * Ha van, ajánlatot tesz a frissítés letöltésére.

### Előnyök

* **Felhasználóbarát élmény**: nincs szükség kézi beavatkozásra.
* **Adatbiztonság**: a helyi adatok offline módban is megbízhatóan elérhetők.
* **Megbízhatóság**: a rendszer minden körülmények között biztosítja a szolgáltatás működését.

**3.1.18 Kihívások és megoldási lehetőségek**

Az offline funkciók bevezetése minden modern webes rendszerben komoly előnyöket biztosít, ugyanakkor számos kihívást is rejt magában. Az utazási weboldal esetében a cél egy **felhasználóbarát, megbízható és erőforráshatékony megoldás** létrehozása, amely képes kezelni az utazás közben előforduló hálózati problémákat. Az alábbiakban részletesen bemutatom a főbb problémaköröket és a lehetséges megoldási stratégiákat.

**1. Kihívás: Nagy fájlméret és letöltési idő**

Az offline módhoz szükséges csomagok többféle adatot tartalmaznak:

* **Térképcsempék** (grafikus formátumban, pl. PNG vagy vektoros SVG-ben),
* **Útvonal-információk** (pl. JSON fájlok),
* **Látnivalók metaadatai** (nyitvatartás, leírás, cím),
* **Nyelvi változatok** (előre fordított tartalmak több nyelven).

Ezek együttes mérete könnyen több száz MB-ra is rúghat.

**Megoldási lehetőségek**:

* **Csempézett térképek**: a teljes térkép helyett kisebb, darabolt csempék töltődnek le (pl. 256x256 px blokkok), így csak a szükséges részek kerülnek letöltésre.
* **Progresszív letöltés**: a rendszer először a legszükségesebb adatokat tölti le (pl. látnivalók listája), majd a háttérben fokozatosan tölti a nagyobb térképcsempéket.
* **Tömörítés és optimalizálás**: JSON fájlok minifikálása, képek WebP formátumba konvertálása.
* **Letöltés méretének kijelzése**: a felhasználó számára egyértelműen jelezni kell, mekkora tárhelyet igényel a letöltés.

**Példa táblázat**:

| **Adattípus** | **Átlagos méret (MB)** | **Optimalizált méret (MB)** | **Megjegyzés** |
| --- | --- | --- | --- |
| Térképcsempék | 250 | 120 | Régiónként töltve |
| Útvonalak JSON | 15 | 5 | Minifikált formátum |
| Látnivalók adatai | 30 | 10 | Nyelvi változatok nélkül |
| Nyelvi változatok | 50 | 20 | Csak letöltött nyelvek |

**2. Kihívás: Adatok elavulása**

Az offline csomagok statikus jellegükből adódóan hamar elavulhatnak. Például:

* Egy múzeum megváltoztatja a nyitvatartását,
* Új látnivalót adnak hozzá az adatbázishoz,
* Felújítás miatt egy útvonal ideiglenesen lezárásra kerül.

**Megoldási lehetőségek**:

* **Verziószám és időbélyeg** minden csomagban (metadata.json).
* **Automatikus ellenőrzés online kapcsolódáskor**: a rendszer összeveti a helyi és szerveroldali verziót.
* **Részleges frissítés**: csak a változott fájlok töltődnek le (pl. új látnivaló JSON, frissített nyelvi fájl).
* **Felhasználói értesítések**: az alkalmazás figyelmezteti a felhasználót (*„A Budapesti útvonalcsomag 12 napja nem lett frissítve. Szeretnéd most letölteni a legújabbat?”*).

**3. Kihívás: Korlátozott tárhely**

A felhasználók többsége mobil eszközről használja az oldalt, ahol a tárhely korlátozott és gyakran más alkalmazások is nagy helyet foglalnak.

**Megoldási lehetőségek**:

* **Tárhely-kalkulátor**: a letöltés előtt a rendszer becslést ad a szükséges helyről.
* **Beállítható maximum**: a felhasználó szabályozhatja, mennyi helyet engedélyez offline adatokra (pl. 500 MB).
* **Automatikus tisztítás**: a régóta nem használt csomagok törlése.
* **Manuális kezelés**: a felhasználó bármikor törölhet egy régiót, ha nincs rá szüksége.

**4. Kihívás: Eszközkompatibilitás és böngészőkorlátok**

A böngészők eltérően kezelik az offline tárolást. Például:

* **iOS Safari**: erősen korlátozott cache méret (~50–100 MB).
* **Android Chrome**: lehetővé teszi több GB cache-t is.
* **Asztali böngészők**: általában nagyobb tárolási limiteket biztosítanak.

**Megoldási lehetőségek**:

* **Kombinált tárolási stratégia**:
  + kisebb adatok → localStorage vagy IndexedDB,
  + nagyobb térképcsempék → Cache API.
* **Platformspecifikus optimalizáció**: iOS-re kisebb, Androidra nagyobb csomagok letöltése.
* **Hibrid alkalmazás lehetősége**: hosszú távon React Native / Capacitor wrapper, ami közvetlenül hozzáfér az eszköz tárhelyéhez.

**5. Kihívás: Felhasználói élmény és interakció**

A technikai megvalósítás mellett kulcsfontosságú, hogy a felhasználó számára az egész folyamat **átlátható és kényelmes** legyen.

**Megoldási lehetőségek**:

* **Állapotjelző ikonok**: online/offline állapot mutatása (pl. zöld = online, szürke = offline).
* **Letöltési folyamat vizualizációja**: százalékos sáv vagy animáció, amely mutatja a letöltés előrehaladását.
* **Felhasználói kontroll**: a rendszer sosem tölt le automatikusan nagy fájlokat, mindig engedélyt kér.
* **Zökkenőmentes váltás**: az alkalmazás biztosítja, hogy az offline → online átmenet és vissza **megszakítás nélkül** történjen.

**Forgatókönyv példa**:

* A felhasználó offline állapotban elindít egy sétát a Budai Várban.
* A GPS-alapú pozíciókövetés működik, de új látnivalókat nem tud betölteni.
* Amint újra van internet, a rendszer automatikusan frissíti a látnivalók adatait és értesíti a felhasználót, hogy „Új nyitvatartási információk elérhetők a Halászbástyához.”

**6. Kihívás: Biztonság és adatvédelem**

Offline módban is tárolódnak adatok a felhasználó eszközén. Ez adatvédelmi és biztonsági szempontból figyelmet igényel.

**Megoldási lehetőségek**:

* **Cookie-k minimális használata**: csak a legszükségesebb adatokat tároljuk (pl. felhasználói előzmények az ajánlásokhoz).
* **Titkosított kommunikáció online módban**: minden adatcsere HTTPS-en keresztül történik.
* **Felhasználói adatok védelme**: az offline csomagokban nincsenek érzékeny személyes adatok, csak nyilvános tartalmak (útvonalak, látnivalók).
* **Helyi törlés lehetősége**: a felhasználó egy gombnyomással törölheti az összes offline adatot.

**Összegzés**

Az offline funkciók megvalósítása során az alábbi területekre kell kiemelt figyelmet fordítani:

1. **Fájlméret minimalizálása és letöltési folyamat optimalizálása**.
2. **Adatok frissen tartása** verziókezeléssel és frissítési mechanizmusokkal.
3. **Tárhely gazdálkodás**, amely lehetővé teszi a régiók szelektív letöltését és törlését.
4. **Platformok közötti kompatibilitás** biztosítása.
5. **Felhasználói élmény növelése** intuitív jelzésekkel és kontroll lehetőségekkel.
6. **Adatbiztonság** és felhasználói kontroll az offline tárolt információk felett.

Ezek kombinációja biztosítja, hogy az utazási weboldal a valós használat során, akár instabil vagy teljesen hiányzó internetkapcsolat mellett is **megbízhatóan, gyorsan és biztonságosan** működjön.

## 3.1.19 Összefoglalás

Az offline navigáció bővített megvalósítása lehetővé teszi, hogy a felhasználók:

* előre letöltött csomaggal bárhol navigáljanak,
* GPS segítségével valós időben kövessék útvonalukat,
* mindig naprakész és ellenőrzött adatokhoz férjenek hozzá,
* kombinált online-offline élményt kapjanak.

Ez a funkció a projekt egyik leginnovatívabb eleme, amely közvetlenül növeli a **felhasználói elégedettséget és az alkalmazás értékét a turisztikai piacon**.

**4. Adatbázis-tervezés**

**4.1 Az adatbázis szerepe a rendszerben**

Az adatbázis biztosítja:

* a szállások, látnivalók és útvonalak strukturált tárolását,
* a többnyelvű tartalmak kezelését,
* a felhasználói előzmények (cookie-hoz kötött) elmentését,
* az offline csomagok előállításához szükséges adatok szolgáltatását.

A MySQL relációs adatmodellje jól illeszkedik a projekt igényeihez, mert:

* könnyen kezelhető táblákkal lehet reprezentálni a szállásokat, látnivalókat, útvonalakat,
* támogatja a kapcsolatok (FOREIGN KEY) kezelését,
* hatékony lekérdezéseket biztosít az ajánlások és útvonalak előállításához.

## 4.2 Főbb táblák

### 4.2.1 users

A felhasználói fiókok tárolása (amennyiben van bejelentkezés). A cookie-hoz kötött előzmények is ide köthetők.

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

email VARCHAR(100) UNIQUE,

password\_hash VARCHAR(255),

preferred\_language VARCHAR(5), -- pl. 'hu', 'en', 'de'

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

### 4.2.2 attractions

A látnivalók adatai.

CREATE TABLE attractions (

attraction\_id INT AUTO\_INCREMENT PRIMARY KEY,

latitude DECIMAL(10, 6),

longitude DECIMAL(10, 6),

category VARCHAR(50), -- pl. 'museum', 'nature', 'gastronomy'

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

### 4.2.3 attraction\_translations

Többnyelvű tartalom a látnivalókhoz.

CREATE TABLE attraction\_translations (

translation\_id INT AUTO\_INCREMENT PRIMARY KEY,

attraction\_id INT,

language\_code VARCHAR(5), -- pl. 'hu', 'en', 'de'

name VARCHAR(255),

description TEXT,

FOREIGN KEY (attraction\_id) REFERENCES attractions(attraction\_id)

);

### 4.2.4 routes

Statikus útvonalak tárolása.

CREATE TABLE routes (

route\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255),

duration\_minutes INT, -- előre megadott időtartam

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

### 4.2.5 route\_points

Az útvonalhoz tartozó pontok (szállás → látnivaló → látnivaló).

CREATE TABLE route\_points (

point\_id INT AUTO\_INCREMENT PRIMARY KEY,

route\_id INT,

attraction\_id INT,

sequence\_order INT, -- sorrend

FOREIGN KEY (route\_id) REFERENCES routes(route\_id),

FOREIGN KEY (attraction\_id) REFERENCES attractions(attraction\_id)

);

### 4.2.6 user\_history

Felhasználói előzmények (ajánlásokhoz). Nem gépi tanulás, csak egyszerű log.

CREATE TABLE user\_history (

history\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

attraction\_id INT,

viewed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (attraction\_id) REFERENCES attractions(attraction\_id)

);

### 4.2.7 offline\_packages

Az offline csomagok metaadatai.

CREATE TABLE offline\_packages (

package\_id INT AUTO\_INCREMENT PRIMARY KEY,

region VARCHAR(100),

version INT,

file\_path VARCHAR(255), -- pl. /offline/balatonfured\_v2.zip

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

## 4.3 Kapcsolatok áttekintése (ER diagram leírása)

* Egy attractions rekordhoz több attraction\_translations tartozhat.
* Egy routes több route\_points-ot tartalmazhat.
* Egy route\_points egy attractions ponthoz kötődik.
* Egy users rekordhoz több user\_history sor tartozik.
* Az offline\_packages független, de régiónként kapcsolódik az attractions és routes adatokhoz.

Szövegesen:

* **1:N** kapcsolat: attraction → translations
* **1:N** kapcsolat: route → route\_points
* **1:N** kapcsolat: user → user\_history
* **N:1** kapcsolat: route\_points → attractions

## 4.4 Példalekérdezések

1. **Látnivalók listázása magyar nyelven:**

SELECT a.attraction\_id, t.name, t.description, a.latitude, a.longitude

FROM attractions a

JOIN attraction\_translations t ON a.attraction\_id = t.attraction\_id

WHERE t.language\_code = 'hu';

1. **Felhasználó által legutóbb megtekintett látnivalók:**

SELECT t.name, h.viewed\_at

FROM user\_history h

JOIN attractions a ON h.attraction\_id = a.attraction\_id

JOIN attraction\_translations t ON a.attraction\_id = t.attraction\_id

WHERE h.user\_id = 5 AND t.language\_code = 'en'

ORDER BY h.viewed\_at DESC

LIMIT 10;

1. **Egy útvonal pontjainak lekérése sorrendben:**

SELECT r.name AS route\_name, t.name AS attraction\_name, rp.sequence\_order

FROM routes r

JOIN route\_points rp ON r.route\_id = rp.route\_id

JOIN attractions a ON rp.attraction\_id = a.attraction\_id

JOIN attraction\_translations t ON a.attraction\_id = t.attraction\_id

WHERE r.route\_id = 3 AND t.language\_code = 'de'

ORDER BY rp.sequence\_order;

## 4.5 Összefoglalás

Az adatbázis logikája:

* **Többnyelvűség** → külön fordítási táblában kezeljük, előre rögzítve.
* **Ajánlások** → a user\_history alapján, egyszerű logikai szabályokkal.
* **Időoptimalizálás** → a routes + route\_points előre definiált statikus adatokból épül.
* **Offline navigáció** → az offline\_packages tárolja a verziózott csomagokat, amelyeket az API szolgáltat.

Ez a struktúra biztosítja, hogy az intelligens funkciók működjenek **gépi tanulás nélkül**, tisztán adatbázis + backend logika segítségével.

## 4. Adatkezelés és architektúra háttér

Az utazási weboldal intelligens funkcióinak működéséhez **strukturált adatkezelés** és jól megtervezett **architektúra** szükséges. A háttérrendszer feladata, hogy megbízhatóan tárolja a felhasználói előzményeket, a látnivalókat, az útvonalakat, valamint biztosítsa a többnyelvű tartalmak kezelését.

A rendszer fő technológiai pillérei a következők:

* **MySQL**: relációs adatbázis a strukturált adatok tárolására.
* **Node.js + Express**: REST API réteg, amely az adatbázist a frontend felé elérhetővé teszi.
* **React**: a felhasználói felület (UI) megjelenítése és interaktív komponensek biztosítása.
* **Cookie-k és session kezelés**: a felhasználói előzmények követésére és az ajánlatok testreszabására.

Az architektúra alapvetően háromrétegű:

1. **Adatbázis réteg** – MySQL tárolja a tartalmakat és a felhasználói előzményeket.
2. **Backend réteg** – Node.js + Express REST API szolgáltatás biztosítja az adatkezelést.
3. **Frontend réteg** – React komponensek, amelyek az API-n keresztül betöltött adatokat megjelenítik.

Ez a rétegzett modell biztosítja a **modularitást, skálázhatóságot és könnyű karbantarthatóságot**.

## 4.1 MySQL adatmodell az intelligens funkciókhoz

A MySQL adatbázis struktúrájának megtervezése alapvető fontosságú, hiszen ez biztosítja a gyors adatlekérdezést és a konzisztens működést. Az adatmodell célja, hogy támogassa:

* a felhasználói előzmények rögzítését,
* a látnivalók és útvonalak statikus tárolását,
* a nyelvi változatok kezelését.

### 4.1.1 Felhasználói előzmények tárolása

Mivel a rendszer nem alkalmaz gépi tanulást, a személyre szabott ajánlások **fiókhoz kötött cookie-k és előzmények** alapján működnek. Az adatbázisban minden felhasználóhoz rögzíthetők a böngészési és keresési előzmények.

**Tábla: user\_history**

| **Oszlop neve** | **Típus** | **Leírás** |
| --- | --- | --- |
| history\_id | INT (PK) | Egyedi azonosító |
| user\_id | INT (FK) | Hivatkozás a users táblára |
| attraction\_id | INT (FK) | Megnézett látnivaló |
| route\_id | INT (FK) | Megtekintett útvonal |
| timestamp | DATETIME | Az interakció ideje |

Ez az információ szolgál alapul az ajánlásokhoz: ha a felhasználó többször keresett „múzeumok” iránt, akkor a rendszer előnyben részesíti ezek megjelenítését.

### 4.1.2 Látnivalók és útvonalak tárolása

Mivel a látnivalók és útvonalak **statikus, előre megadott adatok**, ezek tárolásához külön táblák kerülnek kialakításra.

**Tábla: attractions**

| **Oszlop neve** | **Típus** | **Leírás** | |
| --- | --- | --- | --- |
| attraction\_id | INT (PK) | Egyedi azonosító | |
| name | VARCHAR | Látnivaló neve | |
| category | VARCHAR | Típus (pl. múzeum, park, templom) | |
| location | VARCHAR | Cím vagy koordináta | |
| opening\_hours | VARCHAR | Nyitvatartás | |
| description | TEXT | | Rövid leírás | |

**Tábla: routes**

| **Oszlop neve** | **Típus** | **Leírás** |
| --- | --- | --- |
| route\_id | INT (PK) | Egyedi azonosító |
| name | VARCHAR | Útvonal neve |
| duration | INT | Becsült idő (percben) |
| difficulty | VARCHAR | Könnyű / közepes / nehéz |
| points | TEXT (JSON) | Koordináták és megállók listája |

### 4.1.3 Nyelvi változatok tárolása

A rendszer nem gépi fordítót, hanem előre elkészített fordításokat használ. Ezért minden nyelvi változat külön táblában vagy oszlopban kerül tárolásra.

**Tábla: translations**

| **Oszlop neve** | **Típus** | **Leírás** |
| --- | --- | --- |
| translation\_id | INT (PK) | Egyedi azonosító |
| entity\_type | VARCHAR | attraction vagy route |
| entity\_id | INT (FK) | Hivatkozás a megfelelő rekordra |
| language | VARCHAR(5) | Nyelvkód (pl. hu, en, de) |
| name | VARCHAR | Fordított név |
| description | TEXT | Fordított leírás |

Ez a megoldás rugalmasságot biztosít, és könnyen bővíthető új nyelvekkel.

## 4.2 Node.js + Express REST API szerepe

A rendszer működésének egyik legfontosabb eleme a **backend réteg**, amely az **adatbázis** és a **frontend** közötti kommunikációt biztosítja. Ehhez a projekt a **Node.js futtatókörnyezetet** és az arra épülő **Express keretrendszert** használja.

A REST API feladatai közé tartozik:

* az adatok **lekérdezése** (pl. látnivalók, útvonalak, fordítások),
* az adatok **módosítása** (pl. felhasználói előzmények rögzítése),
* a **nyelvi változatok kiszolgálása** a React frontend számára,
* az **offline csomagok generálása** és előkészítése letöltésre.

### 4.2.1 Architektúra szerepe

A Node.js + Express réteg **köztes komponensként** helyezkedik el:

React Frontend <----> REST API (Express) <----> MySQL Adatbázis

* A frontend **csak az API-n keresztül** éri el az adatokat, így az adatbázis közvetlenül nincs kitéve a külső kéréseknek.
* Az API **validálja** és **feldolgozza** a bejövő kéréseket.
* Az API réteg biztosítja, hogy a frontend mindig a megfelelő formátumban kapja az adatokat (JSON).

### 4.2.2 Alapvető végpontok

A REST API az alábbi fő végpontokkal működik:

1. **Felhasználói előzmények**
   * GET /api/history/:userId → visszaadja a felhasználó előzményeit.
   * POST /api/history → új előzmény rögzítése.
2. **Látnivalók**
   * GET /api/attractions → az összes látnivaló listázása.
   * GET /api/attractions/:id → egy adott látnivaló részletes adatai.
3. **Útvonalak**
   * GET /api/routes → az összes útvonal listázása.
   * GET /api/routes/:id → adott útvonal részletei (pontok, becsült idő).
4. **Fordítások**
   * GET /api/translations/:language → adott nyelvhez tartozó fordítások.
   * GET /api/attractions/:id/translation/:language → egy látnivaló fordítása adott nyelven.
5. **Offline csomagok**
   * GET /api/offline/:regionId → letölthető offline csomag JSON formátumban.

### 4.2.3 Middleware-ek szerepe

Az Express keretrendszer lehetőséget ad **middleware-ek** használatára, amelyek a kérések feldolgozását teszik biztonságosabbá és hatékonyabbá.

Használatuk a projektben:

* **Autentikáció**: ellenőrzi, hogy a felhasználó azonosítva van-e (cookie/session alapján).
* **Loggolás**: minden API-hívást rögzít egy logfájlba.
* **Error handling**: hiba esetén egységes JSON választ ad (pl. { error: "Database connection failed" }).
* **Cache kezelés**: az ismétlődő lekérésekhez rövid távú gyorsítótár biztosítása.

### 4.2.4 Offline támogatás

Az Express API speciális szerepet tölt be az offline működésben is:

* Az API előre generált **JSON csomagokat** készít (pl. budapest\_routes.json, budapest\_attractions.json).
* Ezeket a felhasználó letöltheti, és a frontend Service Worker tárolja őket az eszközön.
* Az API biztosítja a **verziókövetést** is: minden csomaghoz tartozik egy verziószám.

### 4.2.5 Példa API válasz

**Lekérdezés**:

GET /api/attractions/42?lang=hu

**Válasz**:

{

"id": 42,

"name": "Budai Vár",

"category": "Történelmi helyszín",

"location": "Budapest, Szent György tér",

"opening\_hours": "09:00 - 18:00",

"description": "A Budai Vár Budapest egyik legfontosabb turisztikai látványossága...",

"language": "hu"

}

Ez az egységes JSON formátum lehetővé teszi, hogy a React frontend könnyedén feldolgozza és megjelenítse az adatokat.

### 4.2.6 Előnyök

Az Express REST API réteg biztosítja, hogy:

* **Modularitás** legyen: a frontend és a backend külön fejleszthető.
* **Biztonság** legyen: az adatbázis közvetlenül nem érhető el.
* **Skálázhatóság** legyen: a későbbiekben mobilalkalmazás vagy más kliens is csatlakozhat ugyanahhoz az API-hoz.
* **Offline támogatás** megvalósítható legyen előre generált csomagokkal.

## 4.3 React frontend integráció

A frontend réteg megvalósításához a rendszer **React** könyvtárat alkalmaz, amely rugalmas és komponens-alapú felépítést biztosít. A React lehetővé teszi az adatok dinamikus megjelenítését és az API-ból érkező tartalmak egyszerű feldolgozását.

A React szerepe három fő területre bontható:

1. **Adatlekérés és megjelenítés** – a REST API-ból kapott adatok feldolgozása.
2. **Felhasználói interakciók kezelése** – navigáció, nyelvváltás, ajánlások megjelenítése.
3. **Offline működés támogatása** – Service Worker integráció és cache kezelés.

### 4.3.1 Komponens alapú architektúra

A frontend felépítése komponensekre bontva átláthatóbb és könnyebben karbantartható.

**Főbb komponensek**:

* **AttractionList** – látnivalók listázása.
* **AttractionDetail** – egy adott látnivaló részletes megjelenítése.
* **RouteList** – elérhető útvonalak felsorolása.
* **RouteMap** – útvonal vizuális megjelenítése térképen.
* **LanguageSelector** – nyelvváltó komponens.
* **OfflineIndicator** – ikon, amely jelzi az aktuális online/offline állapotot.

A komponensek **API-hívásokon** keresztül töltik be az adatokat, majd állapotkezeléssel (useState, useEffect, vagy Redux/Context API) jelenítik meg.

### 4.3.2 API integráció Reactben

Az API-hívások a fetch vagy axios könyvtár segítségével valósulnak meg.

**Példa: Látnivaló betöltése**

import { useState, useEffect } from "react";

function AttractionDetail({ id, lang }) {

const [data, setData] = useState(null);

useEffect(() => {

fetch(`/api/attractions/${id}?lang=${lang}`)

.then(res => res.json())

.then(json => setData(json));

}, [id, lang]);

if (!data) return <p>Betöltés...</p>;

return (

<div>

<h2>{data.name}</h2>

<p>{data.description}</p>

<p><b>Nyitvatartás:</b> {data.opening\_hours}</p>

</div>

);

}

Ez a komponens automatikusan betölti a megfelelő nyelvű adatokat az API-ból, és megjeleníti a felhasználónak.

### 4.3.3 Nyelvi támogatás

A React frontend biztosítja a **többnyelvű felületet**.

* A nyelvváltás során a rendszer új API-kéréseket küld az adott nyelvi kóddal.
* A fordításokat nem valós időben, hanem előre lefordított változatokból tölti be.
* A **LanguageSelector** komponens useContext segítségével osztja meg a választott nyelvet a teljes alkalmazással.

**Példa használat**:

const { lang, setLang } = useContext(LanguageContext);

<button onClick={() => setLang("en")}>English</button>

<button onClick={() => setLang("de")}>Deutsch</button>

### 4.3.4 Offline támogatás

A React frontend **Service Worker** segítségével kezeli az offline módot.

* Az első betöltéskor a Service Worker letölti és gyorsítótárba helyezi a főbb útvonalakat és látnivalókat.
* Ha nincs internetkapcsolat, a React alkalmazás a cache-ben tárolt adatokat használja.
* A felhasználót egy **OfflineIndicator** értesíti, amikor a rendszer offline módra vált.

**Forgatókönyv**:

* A felhasználó Budapesten offline állapotba kerül.
* A Service Worker a budapest\_routes.json és budapest\_attractions.json fájlokat tölti be a cache-ből.
* A React komponensek ugyanúgy megjelenítik az adatokat, mintha online lenne.

### 4.3.5 Felhasználói élmény és interakciók

A React frontend egyik fő célja, hogy a felhasználói élményt **folyamatosnak és gördülékenynek** érezze az utazó. Ehhez:

* Betöltéskor animációk (loading spinner, progress bar) jelzik a folyamatokat.
* Offline → online váltáskor az adatok automatikusan frissülnek.
* Az ajánlások dinamikusan változnak a cookie-ban tárolt előzmények alapján.

### 4.3.6 Előnyök

A React integráció az alábbi előnyöket biztosítja:

* **Gyors és dinamikus felhasználói élmény** – az API adatok azonnal megjelennek.
* **Reszponzív design** – mobil, tablet és desktop eszközökön is működik.
* **Offline támogatás** – a letöltött csomagokból is tud működni internet nélkül.
* **Többnyelvűség** – egyszerűen bővíthető új nyelvekkel.
* **Komponens-alapú fejlesztés** – moduláris, könnyen karbantartható kód.

## 4.4 Cookie-kezelés és adatbiztonsági szempontok

Az utazási weboldal egyik fő intelligens funkciója az, hogy **személyre szabott ajánlatokat** és **felhasználóhoz igazított tartalmakat** jelenít meg. Ennek megvalósításához nem komplex gépi tanulási modelleket, hanem **cookie-kezelést és egyszerű adat-előzmény tárolást** használ a rendszer.

A cookie-k és az adatbiztonság kérdésköre azonban központi jelentőségű, hiszen a felhasználói adatok érzékenynek minősülnek, ezért a kezelésüket szabályozott módon kell megoldani.

### 4.4.1 Cookie-k szerepe a rendszerben

A cookie-k több célt szolgálnak a projektben:

1. **Felhasználói előzmények tárolása**
   * Mely látnivalókat tekintette meg a felhasználó.
   * Mely útvonalakat nyitotta meg vagy töltötte le offline használatra.
2. **Személyre szabott ajánlások**
   * Ha egy felhasználó gyakran néz meg például múzeumokat, a rendszer a következő alkalommal több hasonló látnivalót jelenít meg.
   * Ez cookie-ban rögzített preferenciák alapján történik.
3. **Beállítások megőrzése**
   * Nyelvi beállítás (pl. magyar / angol felület).
   * Offline csomagok letöltési státusza.
   * Megjelenítési módok (lista nézet / térkép nézet).

### 4.4.2 Cookie típusok

A rendszer kétféle cookie-t alkalmaz:

* **Szükséges cookie-k**
  + A weboldal alapműködéséhez elengedhetetlenek.
  + Például a nyelvi beállítás és a felhasználói session azonosító.
  + Ezek nélkül a weboldal nem tudja biztosítani a működés folytonosságát.
* **Funkcionális cookie-k**
  + A felhasználói élményt javítják.
  + Például az előzmények rögzítése a személyre szabott ajánlatokhoz.
  + Ezek nem kötelezőek, a felhasználó dönthet a használatuk engedélyezéséről.

### 4.4.3 Adatbiztonsági követelmények

Mivel a rendszer felhasználói adatokat (bár nem érzékeny személyes adatokat, hanem inkább preferenciákat) tárol, kötelező betartani az alábbi biztonsági irányelveket:

1. **HTTPS használata**
   * A cookie-k csak biztonságos kapcsolaton keresztül továbbíthatók.
2. **HttpOnly beállítás**
   * Bizonyos cookie-kat csak a szerver olvashat, így JavaScript nem fér hozzájuk.
   * Ez védi a cookie-t a XSS támadásoktól.
3. **SameSite szabály**
   * A cookie-k harmadik fél webhelyei felől ne legyenek hozzáférhetők.
   * Megelőzi a CSRF támadásokat.
4. **Adatmegőrzési idő**
   * A cookie-k lejárati ideje korlátozott (pl. 30 nap).
   * Nem tárolódhatnak korlátlan ideig.
5. **Felhasználói beleegyezés (GDPR)**
   * Az oldal betöltésekor cookie-banner jelenik meg.
   * A felhasználó kiválaszthatja, hogy engedélyezi-e a funkcionális cookie-k használatát.

### 4.4.4 Példa cookie-kezelésre Expressben

A backend (Node.js + Express) oldalon a cookie-k kezeléséhez például a cookie-parser middleware használható:

const express = require("express");

const cookieParser = require("cookie-parser");

const app = express();

app.use(cookieParser());

// Cookie beállítása

app.get("/set-preference", (req, res) => {

res.cookie("preferredCategory", "museum", {

httpOnly: true,

secure: true,

sameSite: "strict",

maxAge: 1000 \* 60 \* 60 \* 24 \* 30 // 30 nap

});

res.send("Beállítva a preferencia!");

});

// Cookie lekérése

app.get("/get-preference", (req, res) => {

const preference = req.cookies.preferredCategory || "default";

res.send(`Felhasználói preferencia: ${preference}`);

});

### 4.4.5 Cookie-k és személyre szabás folyamata

1. A felhasználó meglátogat egy látnivaló-oldalt.
2. A frontend rögzíti az eseményt, és az API segítségével elmenti cookie-ba.
3. Következő látogatáskor az API kiolvassa a cookie-kat.
4. A REST API a felhasználó preferenciáinak megfelelő sorrendben adja vissza az ajánlásokat.
5. A React frontend ennek megfelelően jeleníti meg a látnivalókat.

### 4.4.6 Összegzés

A cookie-kezelés egyszerű, mégis hatékony módot biztosít a személyre szabott ajánlatok és beállítások megőrzésére.

* Nem használ gépi tanulást, hanem **előzmény-alapú logikát** alkalmaz.
* Az adatbiztonságot a **HTTPS**, **HttpOnly**, **SameSite** és a **felhasználói hozzájárulás** garantálja.
* A felhasználó döntési szabadsága (cookie-banner) biztosítja a **GDPR megfelelést**.

Ezzel a fejezet önmagában kb. **4–5 oldalnyi anyagot** tud biztosítani, ha képernyőképekkel (pl. cookie banner, beállítási oldal) és példakódokkal egészítjük ki.

## 5. Tesztelési terv

A rendszer stabil működésének biztosítása érdekében részletes tesztelési terv készült. A tesztelés célja:

* az intelligens funkciók (ajánlások, időoptimalizálás, nyelvi támogatás, offline navigáció) helyes működésének ellenőrzése,
* a hibák és inkonzisztenciák kiszűrése,
* a felhasználói élmény validálása különböző körülmények között (online/offline, nyelvváltás, cookie-kezelés).

A tesztelési terv három fő kategóriára bontható:

1. **Funkcionális tesztek** – az előírt funkciók helyes működésének ellenőrzése.
2. **Teljesítménytesztek** – a rendszer válaszidejének és terhelhetőségének vizsgálata.
3. **Felhasználói tesztek (UX)** – a tényleges használhatóság és felhasználói élmény mérése.

## 5.1 Funkcionális tesztek

A funkcionális tesztelés célja annak ellenőrzése, hogy a rendszer minden megadott funkciója a specifikációnak megfelelően működik-e.

### 5.1.1 Ajánlatok személyre szabása

**Cél**: Ellenőrizni, hogy a cookie-k alapján valóban személyre szabott ajánlatok jelennek meg.

**Tesztlépések**:

1. A felhasználó több alkalommal megnéz egy adott típusú látnivalót (pl. múzeum).
2. A rendszer cookie-ban rögzíti a preferenciát.
3. A következő látogatásnál a REST API a múzeumokat előrébb sorolja a listában.

**Elvárt eredmény**:

* A preferált kategóriájú látnivalók a lista elején jelennek meg.
* Ha a cookie törlődik, a sorrend visszaáll alapértelmezettre.

### 5.1.2 Időoptimalizálás működése

**Cél**: Biztosítani, hogy az útvonalak és programok összehangolása a nyitvatartások és becsült utazási idő alapján történjen.

**Tesztlépések**:

1. A felhasználó kiválaszt két látnivalót, amelyek közül az egyik csak délelőtt látogatható.
2. A rendszer generál egy útvonalat a kettő között.
3. Az útvonal sorrendjének figyelembe kell vennie a nyitvatartási időket.

**Elvárt eredmény**:

* Az időben korlátozott helyszín mindig az optimális sorrendben kerül be az útvonalba.
* A rendszer hiba nélkül jelzi, ha a két hely nem látogatható ugyanazon a napon.

### 5.1.3 Nyelvváltás ellenőrzése

**Cél**: A többnyelvű támogatás hibátlan működésének biztosítása.

**Tesztlépések**:

1. A felhasználó alapértelmezett nyelven (pl. magyar) belép az oldalra.
2. A nyelvváltóban átvált másik nyelvre (pl. angol).
3. Az API új kérést indít a megfelelő nyelvi fordításokért.

**Elvárt eredmény**:

* Az egész felület és minden látnivaló adatai a választott nyelven jelennek meg.
* Ha valamely fordítás nem áll rendelkezésre, a rendszer alapértelmezett nyelvre esik vissza.

### 5.1.4 Offline navigáció ellenőrzése

**Cél**: Az offline funkciók működésének validálása internetkapcsolat hiányában.

**Tesztlépések**:

1. A felhasználó letölti egy adott város látnivalóit és útvonalait offline használatra.
2. Az internetkapcsolat megszakad.
3. A felhasználó újra megnyitja az alkalmazást.

**Elvárt eredmény**:

* Az alkalmazás a cache-ben tárolt JSON fájlokat használja.
* Az útvonalak és látnivalók teljes mértékben elérhetők offline módban is.
* Az offline állapot vizuálisan jelezve van (pl. "Offline mód" ikon).

### Funkcionális tesztelés összefoglalása

A funkcionális tesztek biztosítják, hogy a rendszer fő intelligens funkciói – személyre szabott ajánlatok, időoptimalizált útvonalak, nyelvi támogatás és offline navigáció – minden esetben megfelelően működjenek.

* A tesztek automatizálhatók (pl. Jest + React Testing Library frontendhez, Mocha/Chai backendhez).
* A manuális tesztelés felhasználói szemszögből ellenőrzi a teljes élményt.

## 5.2 Teljesítménytesztek

A teljesítménytesztelés célja annak vizsgálata, hogy a rendszer hogyan viselkedik különböző terhelési körülmények között. Az intelligens funkciók (ajánlatok személyre szabása, időoptimalizálás, nyelvi támogatás és offline navigáció) csak akkor biztosítanak jó felhasználói élményt, ha **gyors válaszidővel** és **stabil működéssel** párosulnak.

### 5.2.1 Tesztelési célok

A teljesítménytesztek során a következő tényezők vizsgálata történik:

1. **Válaszidő** – milyen gyorsan szolgálja ki az API a kéréseket?
2. **Skálázhatóság** – hogyan viselkedik a rendszer nagy felhasználószám mellett?
3. **Terhelhetőség** – mi az a maximum kérésszám, amit még stabilan kiszolgál?
4. **Offline/online váltás gyorsasága** – mennyi idő szükséges a cache-ből való betöltéshez?
5. **Nyelvváltás sebessége** – mennyi idő alatt tölti be a React az új fordításokat?

### 5.2.2 Tesztelési környezet

A tesztek reprodukálhatósága érdekében a következő beállításokat használjuk:

* **Backend**: Node.js + Express REST API
* **Adatbázis**: MySQL (közepes méretű, pl. 10 000 látnivaló + 500 útvonal)
* **Frontend**: React alkalmazás, Service Worker-rel az offline támogatáshoz
* **Tesztelő eszközök**:
  + **Apache JMeter** – terheléses tesztekhez
  + **Lighthouse / WebPageTest** – frontend betöltési sebesség méréséhez
  + **Postman + Newman** – API válaszidők automatizált méréséhez

### 5.2.3 Tesztforgatókönyvek

### 1. API válaszidő mérése

* **Forgatókönyv**: 1000 párhuzamos kérés a /api/attractions végpontra.
* **Elvárt eredmény**: az átlagos válaszidő < 500 ms legyen.

### 2. Nyelvváltás sebessége

* **Forgatókönyv**: a felhasználó vált magyar → angol → német nyelvre egymás után.
* **Elvárt eredmény**: a fordítások betöltése 1 másodpercen belül történjen.

### 3. Offline cache elérés ideje

* **Forgatókönyv**: a felhasználó kapcsolat nélkül nyitja meg a budapest\_routes.json fájlt.
* **Elvárt eredmény**: a Service Worker < 200 ms alatt visszaadja a cache-tartalmat.

### 4. Terhelési határvizsgálat

* **Forgatókönyv**: fokozatosan emelni a párhuzamos kérések számát (100 → 500 → 2000 → 5000).
* **Elvárt eredmény**: 2000-ig a rendszer stabilan működik, 5000 környékén várható a teljesítmény romlása.

### 5. Útvonal-optimalizálás sebessége

* **Forgatókönyv**: egy útvonal 5 látnivalóval és eltérő nyitvatartásokkal.
* **Elvárt eredmény**: az útvonalterv < 1 másodperc alatt elkészüljön.

### 5.2.4 Mérőszámok

A teljesítmény méréséhez az alábbi metrikákat használjuk:

* **Átlagos válaszidő (ms)**
* **Maximális válaszidő (ms)**
* **Átviteli sebesség (requests/sec)**
* **Hibaarány (%)** – hány kérés végződik hibával túlterhelés esetén
* **Cache hit ratio (%)** – hány kérés teljesült offline cache-ből

### 5.2.5 Kockázatok és megoldások

* **Kockázat**: Nagy felhasználószám esetén az adatbázis válaszideje megnő.
  + **Megoldás**: Indexelés, query optimalizálás, caching réteg (pl. Redis).
* **Kockázat**: Nyelvi fájlok betöltése lassú lehet.
  + **Megoldás**: Fordítások előtöltése a Service Worker cache-be.
* **Kockázat**: Offline csomagok túl nagy méretűek lehetnek.
  + **Megoldás**: Régiónkénti darabolás (pl. Budapest → kerületenként).

### 5.2.6 Összegzés

A teljesítménytesztek célja annak biztosítása, hogy a rendszer:

* **gyorsan reagáljon** kis és nagy terhelés mellett is,
* **zökkenőmentesen váltson** online és offline mód között,
* **felhasználóbarát élményt** nyújtson nyelvváltáskor és útvonaltervezéskor.

A mért eredmények alapján meghatározható a rendszer **biztonságos kapacitása** és az a pont, ahol további optimalizálás szükséges.

## 5.3 Felhasználói tesztelés (UX)

A funkcionális és teljesítménytesztek mellett elengedhetetlen a **felhasználói élmény (User Experience – UX)** vizsgálata. Egy utazási weboldal sikeressége nagymértékben azon múlik, hogy a felhasználók mennyire találják egyszerűnek, gyorsnak és kényelmesnek a rendszer használatát.

A UX-tesztelés célja annak biztosítása, hogy:

* a felület **intuitív és könnyen navigálható**,
* a **nyelvváltás, ajánlások és offline mód** világosan kommunikált legyen,
* a felhasználók **minimális erőfeszítéssel** érjék el a kívánt funkciókat.

### 5.3.1 Tesztelési módszerek

A felhasználói tesztelés többféle megközelítéssel valósul meg:

1. **Közvetlen felhasználói teszt**
   * Tesztalanyok kiválasztása (pl. 10–15 fő különböző korosztályból és utazási szokásokkal).
   * Meghatározott feladatok elvégzése (pl. „Keress meg egy múzeumot, válts nyelvet, majd mentsd le az útvonalat offline módra”).
   * A folyamat megfigyelése és jegyzetelése.
2. **Távoli tesztelés**
   * A felhasználó saját eszközén próbálja ki a rendszert.
   * Események naplózása automatikusan történik (pl. mely gombokra kattintott, mennyi ideig keresett egy funkciót).
3. **Kérdőíves visszajelzés**
   * Szubjektív felhasználói élmény mérése kérdőívekkel (pl. SUS – System Usability Scale, 1–5 skálán értékelés).

### 5.3.2 Tesztelt funkciók

A UX-tesztek során a következő kulcsfunkciókat vizsgáljuk:

* **Ajánlatok személyre szabása**
  + Könnyen észrevehető-e, hogy a rendszer ajánlásokat ad?
  + A felhasználó hasznosnak találja-e az ajánlásokat?
* **Időoptimalizálás**
  + Érthető-e, hogy miért olyan sorrendben javasolja a program az útvonalat?
  + Kaphat-e figyelmeztetést, ha valami nyitvatartási időn kívül lenne?
* **Nyelvváltás**
  + Egyszerűen megtalálható-e a nyelvváltó?
  + Azonnal és hibátlanul működik-e a fordítás?
* **Offline mód**
  + Érthetően jelzi-e a rendszer, ha offline működik?
  + Könnyű-e előre letölteni a kívánt csomagokat?

### 5.3.3 Tipikus tesztforgatókönyvek

1. **Első használat**
   * Felhasználó először nyitja meg az oldalt.
   * Meg kell találnia egy szállást, és megnyitni a közelében található látnivalókat.
2. **Nyelvváltás**
   * Felhasználó átvált angolról németre, majd vissza magyarra.
   * Ellenőrzi, hogy minden tartalom helyesen jelenik meg.
3. **Ajánlások tesztelése**
   * Felhasználó több templomot tekint meg.
   * Következő alkalommal a rendszer előrébb sorolja a vallási látnivalókat.
4. **Offline csomag használata**
   * Felhasználó letölti Budapest központi látnivalóit.
   * Repülőn, internet nélkül újra megnyitja az appot.
   * Ellenőrzi, hogy minden adat elérhető.

### 5.3.4 Értékelési szempontok

A UX-tesztelés során a következő mutatókat vizsgáljuk:

* **Időráfordítás**: Mennyi időbe telik egy adott feladat elvégzése?
* **Sikerességi arány**: A felhasználók hány százaléka találja meg és hajtja végre helyesen a feladatot?
* **Hibák száma**: Hányszor kattint rossz helyre, téved el a felhasználó?
* **Elégedettség**: A felhasználó mennyire tartja könnyűnek a rendszer használatát (1–5 skálán)?

### 5.3.5 Várható eredmények

A tesztek során az alábbiakat kell igazolni:

* A felhasználók **80% felett** önállóan végre tudja hajtani a kulcsfeladatokat (nyelvváltás, offline csomag letöltés).
* Az ajánlások és az időoptimalizálás funkciója **érthető és hasznos** a többség számára.
* Az offline mód váltása **világosan jelzett**, és nem okoz zavart.
* Az általános elégedettség **átlagosan 4/5** szintet elér.

### 5.3.6 Összegzés

A felhasználói tesztelés célja, hogy ne csak technikailag működjön a rendszer, hanem **valós utazók igényeit is kiszolgálja**.

* Ha a tesztek pozitívak, az igazolja a rendszer használhatóságát.
* Ha negatív visszajelzések érkeznek, azokat a fejlesztési tervbe kell integrálni.
* A UX-tesztelés így a projekt **folyamatos javításának** kulcsa.

## 6. Összefoglalás és jövőbeli bővítések

A dokumentációban bemutatott rendszer célja egy olyan utazási weboldal megvalósítása, amely nem csupán szállásfoglalási lehetőséget kínál, hanem a felhasználók számára **kényelmesen elérhetővé teszi a közeli látnivalókat, útvonalakat és releváns kiegészítő információkat** is.

A projekt keretében az intelligens funkciók és technikai fejlesztések kaptak hangsúlyt:

* **Ajánlások személyre szabása** a felhasználói előzmények alapján, egyszerű cookie-kezelés és fiókhoz kötött adatok segítségével.
* **Időoptimalizálás** statikus, előre megadott útvonalak és nyitvatartási idők alapján.
* **Nyelvi támogatás** több, előre lefordított nyelvi változattal.
* **Offline navigáció** a letölthető térképek és útvonalcsomagok révén.

A technológiai háttér (Node.js + Express REST API, MySQL adatbázis, React frontend) biztosítja a rendszer modularitását és könnyű bővíthetőségét.

## 6.1 A fejlesztés jelentősége

Az utazási szektorban a felhasználói élmény az egyik legfontosabb tényező. Egy olyan rendszer, amely egyszerre:

* **gyors és stabil**,
* **offline is működőképes**,
* **többnyelvű támogatást kínál**,
* és **személyre szabott ajánlatokat jelenít meg**,

komoly versenyelőnyt biztosíthat.

A dokumentációban bemutatott megoldások nem csupán a technikai megvalósíthatóságot tárgyalják, hanem **üzleti szempontból is értékesek**, hiszen a felhasználói elégedettség növekedése közvetlenül hozzájárulhat a platform népszerűségéhez és bevételtermelő képességéhez.

## 6.2 Jövőbeli lehetőségek

Bár a rendszer a kezdeti verzióban statikus adatokkal és cookie-alapú személyre szabással működik, a jövőben számos irányban továbbfejleszthető:

### 6.2.1 Dinamikus útvonaltervezés

* Valós idejű útvonaltervezés GPS-adatok és forgalmi viszonyok alapján.
* Alternatív útvonalak felajánlása (pl. gyalogos, bicikli, tömegközlekedés).

### 6.2.2 Valós idejű adatok integrációja

* Látnivalók aktuális látogatottságának kijelzése (pl. zsúfoltsági adatok).
* Eseménynaptárak beépítése (fesztiválok, koncertek, kulturális események).

### 6.2.3 Gépi tanulás bevezetése

* Valódi, prediktív ajánlórendszer kiépítése a felhasználói viselkedés elemzése alapján.
* Automatikus programajánlók összeállítása (pl. „3 nap Budapesten” személyre szabva).

### 6.2.4 Kiterjesztett nyelvi támogatás

* Automatikus fordítás beépítése API segítségével (pl. Google Translate, DeepL).
* Hangalapú felhasználói interakció támogatása több nyelven.

### 6.2.5 Mobilalkalmazás fejlesztése

* PWA (Progressive Web App) kibővítése natív mobilalkalmazássá.
* Push értesítések használata (pl. „A múzeum, amit kinéztél, 30 perc múlva zár”).

### 6.2.6 Gamifikáció és közösségi funkciók

* Jelvények gyűjtése a meglátogatott látnivalókért.
* Közösségi ajánlások, értékelések és tippek integrálása.

## 6.3 Összegzés

A projekt jelenlegi megvalósítása **stabil, biztonságos és felhasználóbarát alapot** nyújt.  
A bemutatott funkciók már önmagukban is jelentős értéket képviselnek, ugyanakkor a jövőbeni fejlesztések révén a rendszer **egy modern, intelligens és interaktív utazási platformmá** válhat.

A dokumentáció lezárásaként kijelenthető, hogy:

* az intelligens funkciók hozzájárulnak a felhasználói elégedettséghez,
* a technikai megoldások biztosítják a jövőbeni skálázhatóságot,
* a rendszer jó alapot jelent további innovációk bevezetésére.

# Melléklet: Jelenlegi és jövőbeli funkciók összehasonlítása

| **Funkcióterület** | **Jelenlegi megvalósítás** | | **Jövőbeli fejlesztési lehetőség** | |
| --- | --- | --- | --- | --- |
| **Ajánlások** | Cookie-alapú, előzményekhez kötött ajánlatok | | Gépi tanulás alapú prediktív ajánlások, automatikus programajánló | |
| **Útvonaltervezés** | Statikus, előre megadott útvonalak és nyitvatartások alapján | | Dinamikus útvonaltervezés valós idejű forgalmi és GPS-adatokkal | |
| **Nyelvi támogatás** | Előre lefordított nyelvi változatok | | Automatikus fordítás API-kkal, hangalapú interakció | |
| **Offline navigáció** | | Letölthető statikus térképek és útvonalcsomagok | | Régiókra bontott részletes offline tartalom, valós idejű frissítés offline módban |
| **Felhasználói élmény (UX)** | | Többnyelvű felület, egyszerű navigáció | | Gamifikáció (jelvények, kihívások), közösségi funkciók (értékelés, tippek) |
| **Technológiai háttér** | | Node.js + Express REST API, MySQL, React | | Skálázható mikro-szervizek, NoSQL kiegészítés (pl. MongoDB), natív mobilalkalmazás |
| **Értesítések** | | Nincs | | Push értesítések mobilon (pl. nyitvatartási figyelmeztetés, ajánlott program) |

**Tartalomjegyzék**

**1. Bevezetés**

1.1 A fejlesztési feladat célja  
1.2 Az intelligens funkciók szerepe a weboldalon  
1.3 A technikai fejlesztések jelentősége  
1.4 Dokumentáció felépítése

**2. Intelligens tervezési lehetőségek**

2.1 Ajánlott útvonalak és látnivalók  
 2.1.1 Cookie-k és felhasználói előzmények kezelése  
 2.1.2 Ajánlati logika (statikus adatbázis + személyre szabás)  
 2.1.3 Példa felhasználói forgatókönyv  
2.2 Időoptimalizálás  
 2.2.1 Látnivalók nyitvatartási adatainak kezelése  
 2.2.2 Statikus útvonalak időbeli összehangolása  
 2.2.3 Egynapos program példája  
2.3 Nyelvi támogatás  
 2.3.1 Többnyelvű felület felépítése  
 2.3.2 Előre lefordított oldalváltozatok kezelése  
 2.3.3 Nyelvváltás mechanizmusa a frontendben

**3. Technikai fejlesztések**

3.1 Offline navigáció  
 3.1.1 Térképadatok letöltése  
 3.1.2 Útvonalak offline elérhetősége  
 3.1.3 Online–offline mód közötti váltás  
 3.1.4 Kihívások és megoldási lehetőségek



**4. Adatkezelés és architektúra háttér**

4.1 MySQL adatmodell az intelligens funkciókhoz  
 4.1.1 Felhasználói előzmények tárolása  
 4.1.2 Látnivalók és útvonalak tárolása  
 4.1.3 Nyelvi változatok tárolása  
4.2 Node.js + Express REST API szerepe  
4.3 React frontend integráció  
4.4 Cookie-kezelés és adatbiztonsági szempontok

**5. Tesztelési terv**

5.1 Funkcionális tesztek  
 5.1.1 Ajánlatok személyre szabása  
 5.1.2 Időoptimalizálás működése  
 5.1.3 Nyelvváltás ellenőrzése  
 5.1.4 Offline navigáció ellenőrzése  
5.2 Teljesítménytesztek  
5.3 Felhasználói tesztelés (UX)

**6. Összefoglalás és jövőbeli bővítések**

6.1 Fejlesztés jelentősége  
6.2 Jövőbeli lehetőségek (pl. dinamikus útvonaltervezés, valós idejű adatok)