

1. Bevezetés és Célkitűzés

1.1. A probléma felvetése

A modern nagyvárosokban az autós közlekedés egyik legkritikusabb és legstresszesebb pontja a parkolás. A szabad helyek keresése nemcsak jelentős időveszteséget okoz a vezetőknek, de a városi forgalom lassulásához és a károsanyag-kibocsátás növekedéséhez is hozzájárul.

A probléma kettős:

- Információhiány:** Az autósok gyakran nem tudják, hol vannak szabad, ingyenes vagy kedvező áru parkolóhelyek (pl. eldugott utcákban vagy P+R zónákban).
- Adminisztrációs nehézségek:** Gyakori probléma a parkolási idő túllépése, vagy idegen helyen a visszatalálás a leparkolt járműhöz.

A hallgatói feladat egy olyan szoftverrendszer tervezése és megvalósítása volt, amely közösségi összefogással ("crowdsourcing") és modern technológiai megoldásokkal segít enyhíteni ezeken a nehézségeken.

1.2. A megoldás: ParkolóKereső Rendszer

A projekt keretében kifejlesztett szoftver egy **terképalapú, online, reszponzív webalkalmazás (PWA)**. A rendszer nem csupán egy statikus térkép, hanem egy aktív parkolási asszisztens, amely a teljes folyamatot támogatja a kereséstől a parkolás lezárásáig.

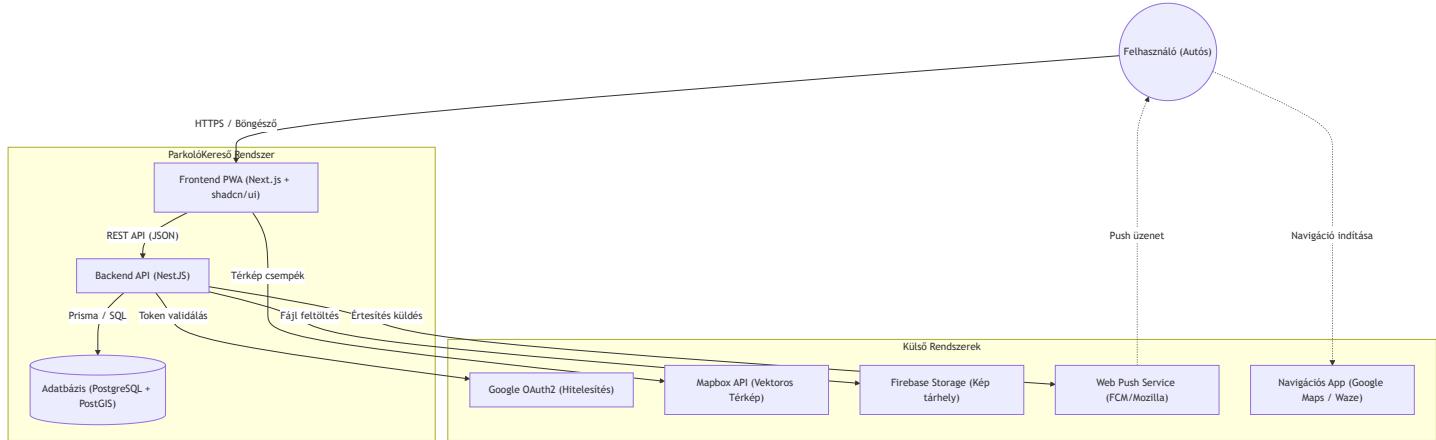
A rendszer főbb képességei:

- Közösségi adatbázis:** A felhasználók rögzíthetik az általuk felfedezett parkolóhelyeket, megadva azok típusát (pl. Ingyenes, Fizetős, P+R), címét és leírását.
- Vizuális információ:** A helyszínekről fényképek tölthetők fel, segítve a tájékozódást.
- Intelligens keresés:** A rendszer támogatja a sugár alapú (radius) keresést a felhasználó GPS pozíciója vagy a térkép középpontja körül.
- Parkolás menedzsment:** A felhasználó egy gombnyomással rögzítheti parkolása kezdetét ("Itt állok"). A rendszer visszaszámítását biztosít, és vizuálisan megjeleníti az autó helyét a térképen a visszataláláshoz.
- Értesítések:** A rendszer push értesítésben figyelmezteti a felhasználót a parkolási idő lejártakor, növelve a kényelmet és elkerülve a büntetéseket.

1.3. Rendszerkörnyezet és Kontextus

Az alkalmazás egy elosztott, 3 rétegű architektúrára épül, amely szorosan integrálódik külső, harmadik fél által biztosított szolgáltatásokkal a térképi megjelenítés, a hitelesítés és az adattárolás terén.

Az alábbi kontextus diagram szemlélteti a rendszer kapcsolatait a külvilággal:



2. Követelményspecifikáció

Ez a fejezet a rendszerrel szemben támasztott funkcionális és minőségi követelményeket, valamint a választott technológiai környezetet részletezi.

2.1. Funkcionális követelmények

A rendszernek az alábbi felhasználói interakciókat (Use Case) kell támogatnia a megvalósítás során.

2.1.1. Felhasználókezelés

- OAuth2 Belépés:** A felhasználó regisztráció nélkül, meglévő Google-fiókjával léphet be. A rendszer átveszi a felhasználó nevét, email címét és profilképét.
- Munkamenet kezelés:** A belépés után a rendszer JWT (JSON Web Token) segítségével azonosítja a felhasználót, így oldalfrissítéskor nem szükséges újra bejelentkezni.

2.1.2. Parkolóhelyek kezelése

- Parkoló rögzítése:** A térképen egy jelölő (marker) elhelyezésével új parkoló hozható létre.
 - Kötelező adatok: Név, Cím, Kategória.
 - Kategóriák: Ingyenes, Fizetős, P+R, Parkolóház, Utcai.
- Képfeltöltés:** A parkolóhoz fénykép csatolható, amelyet a rendszer optimalizálva tárol felhő alapú tárhelyen.
- Parkolók megjelenítése:** A térképen a parkolók kategóriák szerint, markerekkel jelennek meg. A markerre kattintva információs ablak (Popup) nyílik meg a részletekkel.

2.1.3. Keresés és Szűrés

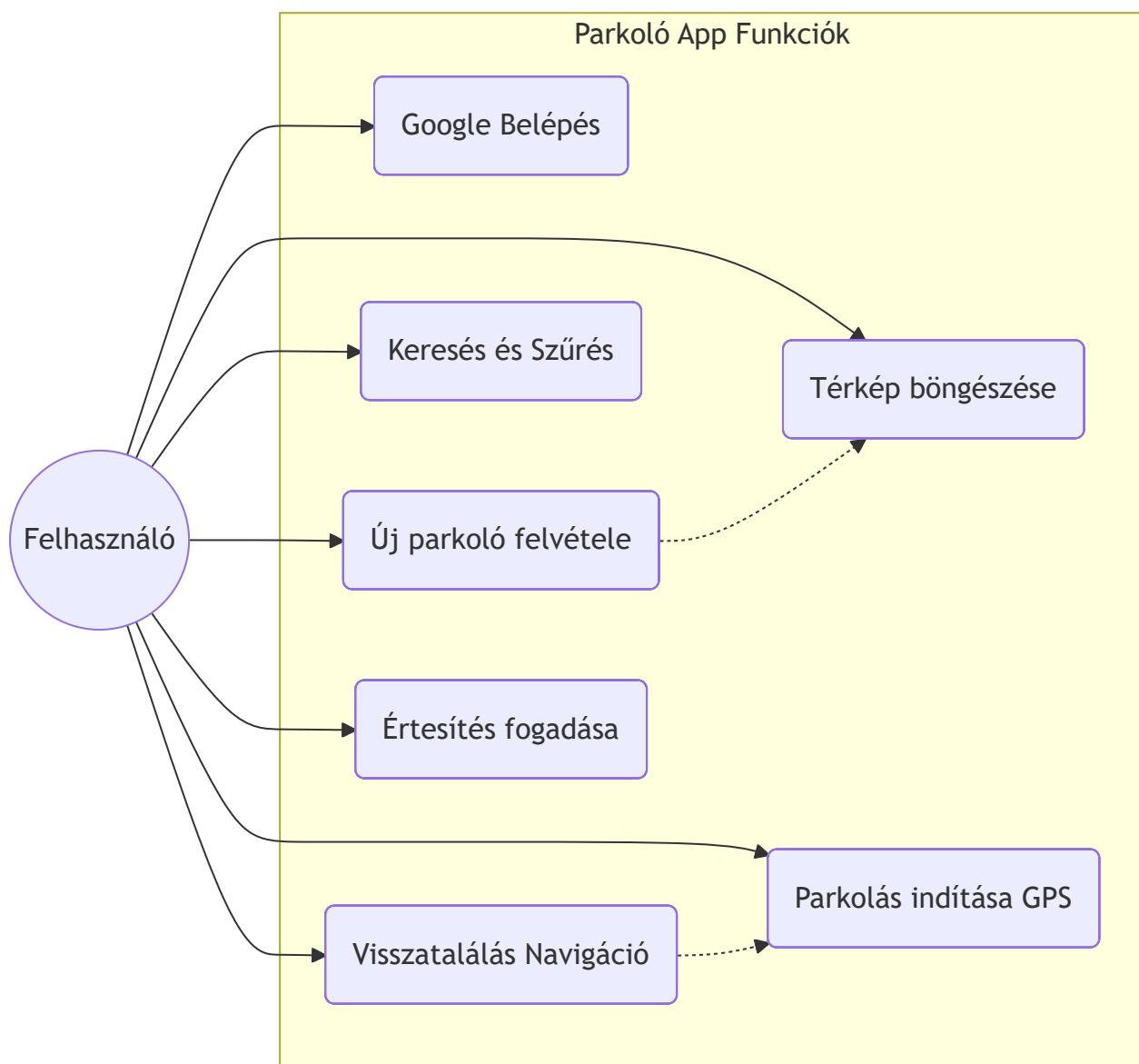
- Sugár alapú keresés (Radius Search):** A felhasználó egy csúszka (Slider) segítségével állíthatja be a keresési távolságot (pl. 500m - 50km). A rendszer csak a megadott körön belüli találatokat listázza.
- Szöveges keresés:** Lehetőség van parkoló nevére vagy címére keresni.
- Kategória szűrés:** A találatok szűkíthetők típus szerint (pl. "Csak Ingyenes").

2.1.4. Parkolási folyamat (Session)

- Parkolás indítása:** A felhasználó a "Leparkoltam itt" gombbal rögzítheti aktuális pozíóját (GPS alapon).
- Lejárat beállítása:** Opcionálisan megadható a parkolójegy lejáratideje.

- **Aktív státusz:** A térképen egy állandó panel ("Overlay") mutatja az eltelt időt és a parkolás helyszínét.
- **Visszatalálás:** A rendszer vizuálisan megkülönbözteti a saját autó pozíóját ("Autó" ikon), és gombnyomásra átirányít a telefon alapértelmezett navigációs alkalmazásába (Google Maps / Waze) az útvonaltervezéshez.
- **Értesítések:** Ha a felhasználó beállított lejárati időt, a rendszer szerver oldali időzítővel (Cron) figyeli azt, és lejáratkor Push Notificationt küld a felhasználó eszközére.

Use Case Diagram



2.2. Nem-funkcionális követelmények

A rendszer minőségi jellemzői a modern webes elvárásoknak megfelelően kerültek meghatározásra:

- **Platformfüggetlenség (PWA):** Az alkalmazás böngészőből futtatható, telepítést nem igényel, de mobil eszközökön "Kezdőképernyőhöz adható" (Add to Home Screen), így natív alkalmazásként

viselkedik.

- **Reszponzivitás:** A felhasználói felület (shadcn/ui , Tailwind CSS) automatikusan igazodik a képernyőmérethez. Mobilon alsó navigációs sávot és érintésbarát felületeket, asztali gépen oldalsávot biztosít.
- **Biztonság:**
 - **HTTPS:** A PWA működéséhez (Service Worker, Geolocation) elengedhetetlen a biztonságos csatorna.
 - **Adatvédelem:** A jelszavakat nem a rendszer, hanem a Google kezeli. A backend API csak hitelesített kéréseket szolgál ki.
- **Teljesítmény:** A térképi megjelenítéshez **Mapbox GL JS** (WebGL) technológiát használunk, amely hardveres gyorsítással biztosítja a sima mozgást és nagyítást nagy adatmennyiség esetén is.

2.3. Futtatási környezet és Technológiai Stack

A rendszer architektúrájának kiválasztásakor a modularitás, a típusbiztonság és a modern fejlesztői élmény (DX) voltak a fő szempontok.

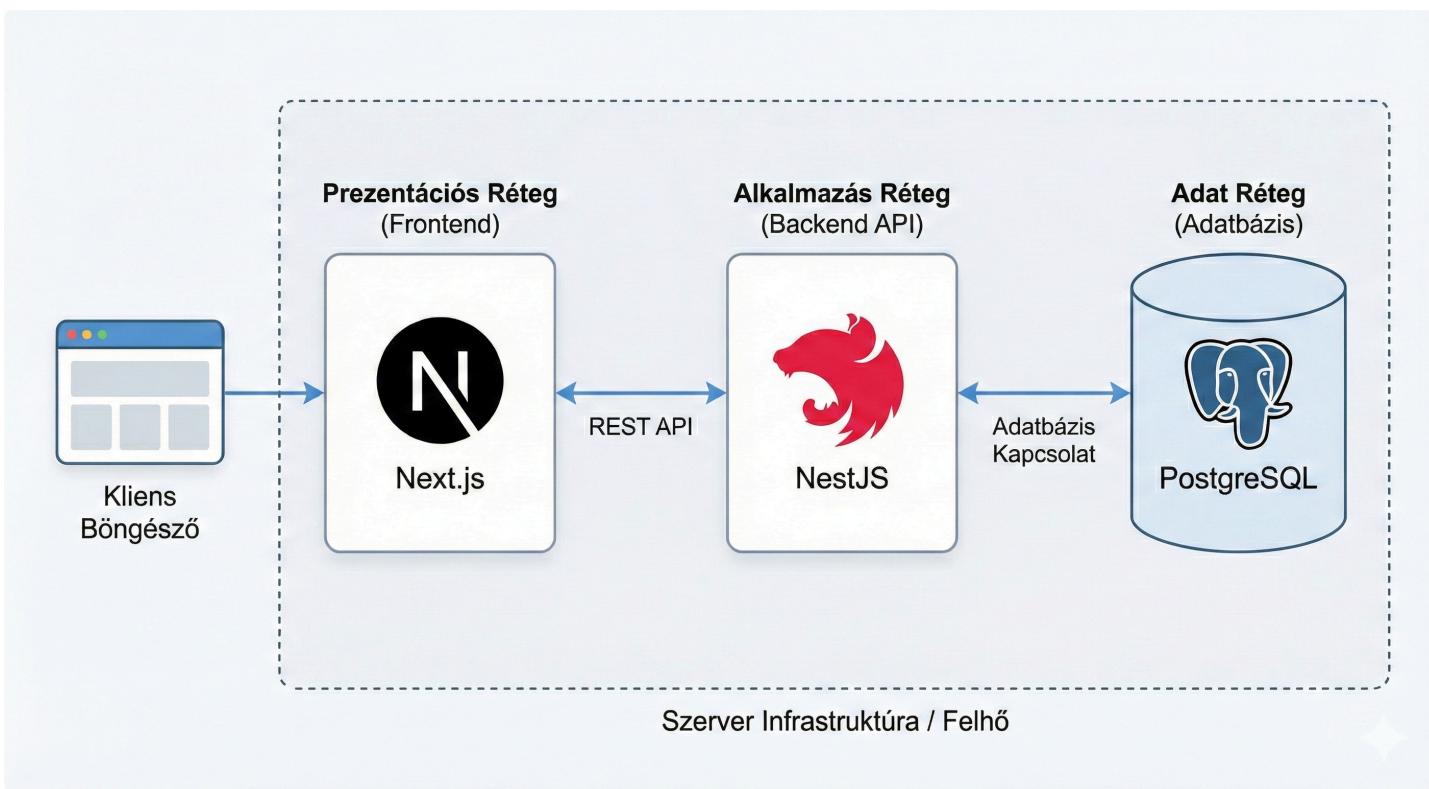
Réteg	Technológia	Indoklás
Frontend	Next.js 16 (React 19)	Szerver oldali renderelés (SSR) és modern App Router struktúra.
Backend	NestJS (Node.js)	Strukturált, moduláris keretrendszer, vállalati szintű architektúra.
Nyelv	TypeScript	Szigorú típusosság a teljes stack-en (Full-stack Type Safety).
Adatbázis	PostgreSQL + PostGIS	Relációs adatbázis fejlett térinformatikai (geospatial) támogatással.
ORM	Prisma	Típusbiztos adatbázis-elérés és migrációkezelés.
Validáció	Zod	Megosztott validációs sémák a Frontend és Backend között (Monorepo).
DevOps	Docker & TurboRepo	Konténerizált futtatás és hatékony monorepo build rendszer.

3. Rendszerterv és Architektúra

Ez a fejezet a rendszer belső felépítését, az adatbázis szerkezetét és a komponensek közötti kommunikációt mutatja be.

3.1. Architekturális áttekintés

A rendszer egy modern, **Monorepo** (TurboRepo) struktúrába szervezett, **3 rétegű (3-tier)** webalkalmazás. A rétegek szigorúan elkülönülnek, kommunikációjuk szabványos interfészeken keresztül történik.



3.1.1. Könyvtárszerkezet (Monorepo)

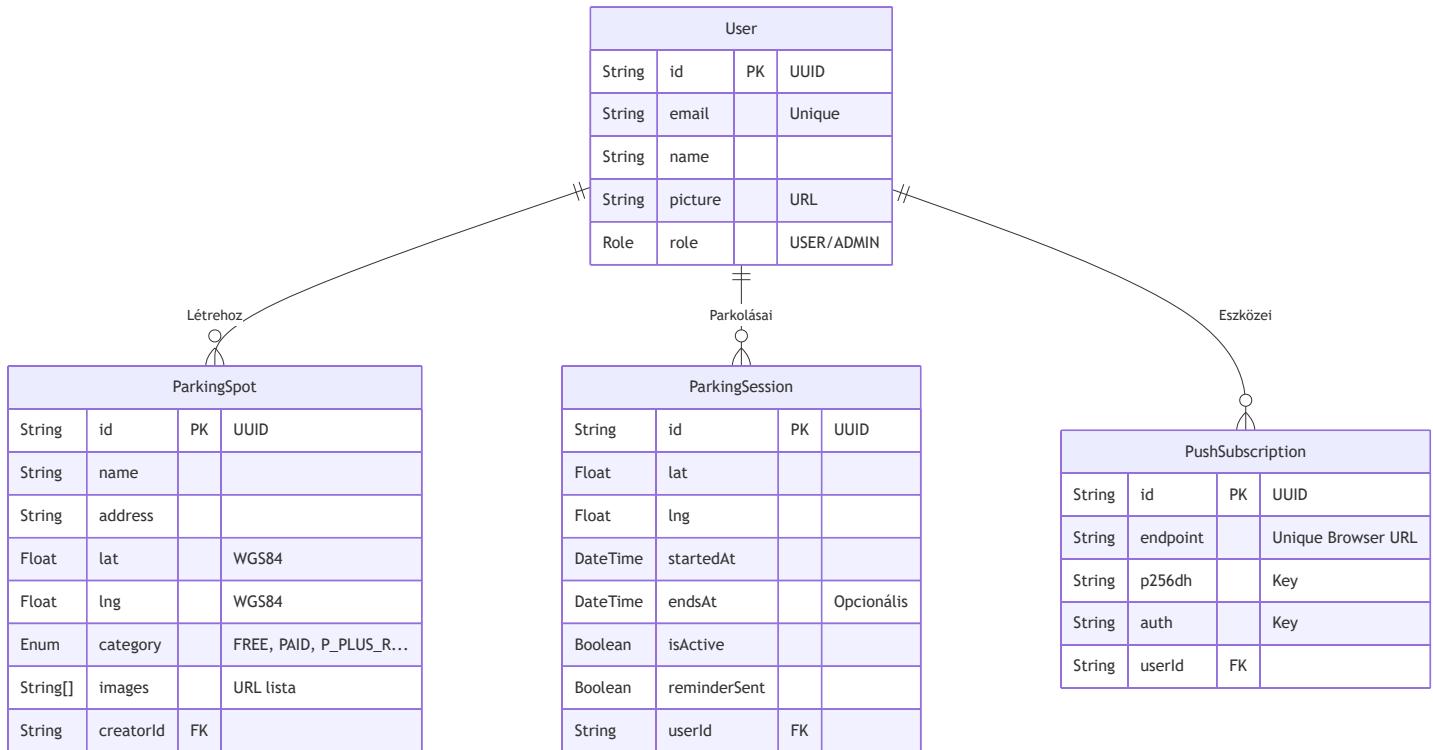
A projekt kódkészlete egyetlen repository-ban található, ami megkönnyíti a verziókezelést és a kódmegosztást.

- **apps/web (Frontend):** A Next.js alapú kliens alkalmazás.
- **apps/api (Backend):** A NestJS alapú szerver alkalmazás.
- **packages/schema (Shared):** A közös Zod validációs sémák és TypeScript típusdefiníciók. Ez biztosítja, hogy a Frontend és Backend közötti adatstruktúrák mindenkorban szinkronban legyenek ("Single Source of Truth").

3.2. Adatbázis terv (Adatmodell)

A rendszer adatait egy **PostgreSQL** relációs adatbázisban tároljuk. Az adatbázis-kezelést és a migrációkat a **Prisma ORM** végzi.

Az alábbi ER (Entity-Relationship) diagram mutatja az entitásokat és kapcsolataikat:



3.2.1. Geolokációs adatok tárolása

Bár a PostgreSQL `geometry` típusokat is támogat (PostGIS), a Prisma ORM-en keresztül a koordinátákat (`lat`, `lng`) lebegőpontos számként (`Float`) tároljuk a kompatibilitás és a könnyebb kezelhetőség érdekében. A térinformatikai számításokat (távolságkeresés) a backend **nyers SQL** (`$queryRaw`) segítségével, a Haversine-formula végzi.

3.3. API Specifikáció

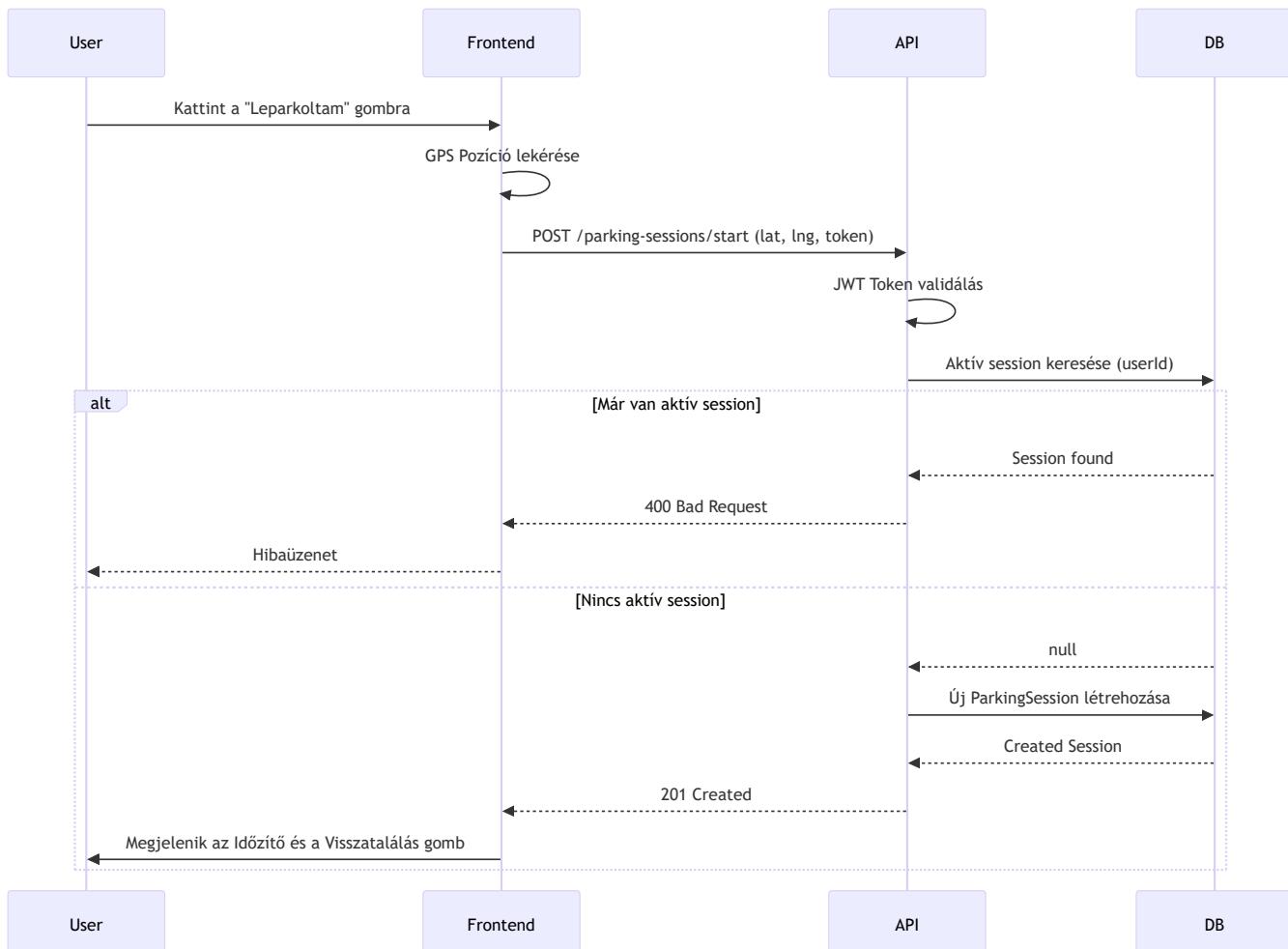
A Backend REST API-t biztosít a Frontend számára. minden végpont (kivéve a publikus adatok lekérését) JWT tokent vár a `Authorization: Bearer <token>` fejlécben.

Főbb végpontok csoportosítva:

Modul	Metódus	Útvonal	Leírás
Auth	GET	/auth/google	Google OAuth2 folyamat indítása.

Modul	Metódus	Útvonal	Leírás
Auth	GET	/auth/google/callback	Google visszatérési pont, JWT generálás.
Parkolók	POST	/parking-spots	Új parkoló létrehozása (Kép URL-ekkel).
Parkolók	GET	/parking-spots	Keresés (Query paraméterek: lat , lng , radius , search).
Session	POST	/parking-sessions/start	Parkolás indítása.
Session	GET	/parking-sessions/active	Aktuális aktív parkolás lekérdezése.
Session	POST	/parking-sessions/end	Parkolás lezárása.
Fájlok	POST	/uploads	Képfeltöltés (Multipart/form-data -> Firebase).
Push	POST	/notifications/subscribe	Böngésző feliratkoztatása értesítésekre.

Szekvencia Diagram: Parkolás Indítása



3.4. Háttérfolyamatok és Értesítések

A rendszer nemcsak felhasználói kérésre reagál, hanem önálló háttérfolyamatokat is futtat.

- **Időzítő (Cron Job):** A backendben a `ParkingSchedulerService` percenként lefut.
 - Lekérdezi azokat az aktív `ParkingSession`-öket, ahol az `endsAt` (lejárat) időpontja a múltban van ÉS `reminderSent` hamis.
 - Ha talál ilyet, meghívja a `NotificationsService`-t.
 - A `web-push` könyvtáron keresztül elküldi az értesítést a felhasználó összes regisztrált eszközére.
 - Az adatbázisban beállítja a `reminderSent = true` értéket (idempotencia).

3.5. Frontend architektúra részletei

A kliens oldal **Next.js App Router**-t használ, amely lehetővé teszi a Szerver (SSR) és Kliens (CSR) komponensek keverését.

- **Állapotkezelés:**
 - **Zustand:** A globális állapotokhoz, mint a bejelentkezett felhasználó tokenje (`useAuthStore`) és a GPS koordináták.
 - **TanStack Query (React Query):** A szerverről érkező adatok (parkolók listája, aktív session) gyorsítótárazására és szinkronizálására. Automatikusan kezeli a "Loading" és "Error" állapotokat.
- **Térkép:** A `react-map-gl` könyvtárat használjuk a Mapbox integrációhoz, amely WebGL alapon, vektorosan rendereli a térképet, támogatva a 3D forgatást és a sötét módot (`next-themes`).

4. Telepítési és Üzemeltetési útmutató

Ez a fejezet lépésről lépésre bemutatja, hogyan állítható be a fejlesztői környezet és hogyan indítható el a rendszer helyi gépen, konténerizáció nélkül.

4.1. Előfeltételek (Prerequisites)

A szoftver futtatásához az alábbi szoftvereknek kell telepítve lenniük a gazdagépen:

- **Node.js (v20+)**: JavaScript futtatókörnyezet.
- **pnpm (v9+)**: Hatékony csomagkezelő (a Monorepo struktúra miatt ajánlott).
 - Telepítés: `npm install -g pnpm`
- **PostgreSQL (v14+)**: Relációs adatbázis szerver.
 - Telepítve és futnia kell a háttérben (vagy elérhetőnek kell lennie egy felhő szolgáltatónál).
 - Létre kell hozni egy üres adatbázist (pl. `parking_db`).

4.2. Konfiguráció és Környezeti változók

A rendszer biztonsági okokból nem tárol titkos kulcsokat a forráskódban. Ezeket környezeti változókban (`.env`) kell megadni.

4.2.1. Backend konfiguráció (`apps/api/.env`)

Hozzon létre egy `.env` fájlt az `apps/api` mappában az alábbi tartalommal:

```

# Szerver beállítások
PORT=3001
NODE_ENV="development"
FRONTEND_URL="http://localhost:3000"

# Adatbázis kapcsolat (PostgreSQL Connection String)
# Formátum: postgresql://USER:PASSWORD@HOST:PORT/DATABASE?schema=public
DATABASE_URL="postgresql://postgres:password@localhost:5432/parking_db?schema=public"

# Biztonság (JWT titkos kulcs - tetszőleges hosszú string)
JWT_SECRET="szupertitkos_fejlesztoi_kulcs"

# Google OAuth2 (Google Cloud Console-ból)
GOOGLE_CLIENT_ID=<GOOGLE_CLIENT_ID>
GOOGLE_CLIENT_SECRET=<GOOGLE_CLIENT_SECRET>
GOOGLE_CALLBACK_URL="http://localhost:3001/auth/google/callback"

# Értesítések (VAPID kulcsok - generálható: npx web-push generate-vapid-keys)
VAPID_PUBLIC_KEY=<VAPID_PUBLIC_KEY>
VAPID_PRIVATE_KEY=<VAPID_PRIVATE_KEY>
VAPID SUBJECT="mailto:admin@example.com"

# Képfeltöltés (Firebase)
FIREBASE_STORAGE_BUCKET=<PROJECT_ID>.firebasestorage.app
GOOGLE_APPLICATION_CREDENTIALS=".firebase-admin-sdk.json"

```

Fontos: A Firebase Service Account kulcsfájlt (`firebase-admin-sdk.json`) másolja be az `apps/api/` gyökérkönyvtárába!

4.2.2. Frontend konfiguráció (`apps/web/.env.local`)

Hozzon létre egy `.env.local` fájlt az `apps/web` mappában:

```

# Backend API elérhetősége
NEXT_PUBLIC_API_URL="http://localhost:3001"

# Térkép (Mapbox Public Token)
NEXT_PUBLIC_MAPBOX_TOKEN="pk.eyJ=...>"

```

4.3. Telepítés és Adatbázis inicializálás

Nyisson egy terminált a projekt gyökérkönyvtárában, és kövesse az alábbi lépéseket:

1. Függőségek telepítése:

Mivel Monorepo-t használunk, ez a parancs telepíti a backend, a frontend és a közös csomagok összes függőségét.

```
pnpm install
```

2. Adatbázis séma szinkronizálása (Migráció):

Ez a lépés létrehozza a szükséges táblákat (User, ParkingSpot, stb.) a megadott PostgreSQL adatbázisban.

```
# Lépjön a backend könyvtárba  
cd apps/api  
  
# Migráció futtatása  
pnpm exec prisma db push  
  
# (Opcionális) Tesztadatok feltöltése  
pnpm exec prisma db seed
```

3. Visszalépés a gyökérbe:

```
cd ../../
```

4.4. A Rendszer Indítása

A fejlesztői környezet indításához (ahol a Frontend és a Backend párhuzamosan fut) használja az alábbi parancsot a gyökérkönyvtárban:

```
pnpm dev
```

A TurboRepo elindítja minden alkalmazást:

- **Backend API:** <http://localhost:3001> (Swagger dokumentáció: /api-docs)
- **Frontend Web App:** <http://localhost:3000>

4.5. Hibaelhárítás

- **Adatbázis hiba:** Győződjön meg róla, hogy a PostgreSQL fut, és a DATABASE_URL helyes felhasználónevet/jelszót tartalmaz.

- **Hiányzó kulcsok:** Ha a térkép nem tölt be, ellenőrizze a `NEXT_PUBLIC_MAPBOX_TOKEN`-t. Ha a képfeltöltés nem működik, ellenőrizze a `firebase-admin-sdk.json` meglétét.
- **Port ütközés:** Ha a 3000-es vagy 3001-es port foglalt, állítsa le a futó folyamatokat, vagy módosítsa a `PORT` változókat az `.env` fájlokban.

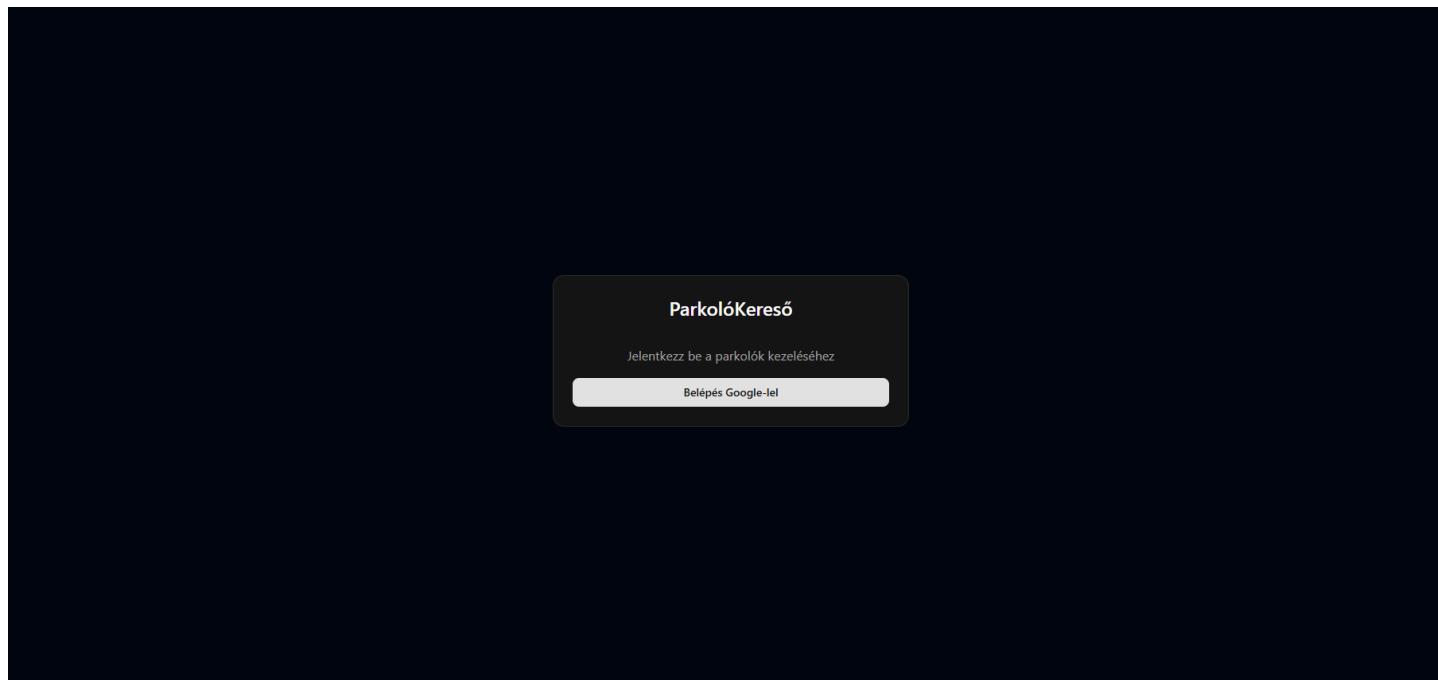
5. Felhasználói kézikönyv és az Alkalmazás bemutatása

Ez a fejezet képernyőképekkel illusztrálva mutatja be az elkészült szoftver főbb funkcióit, lehetővé téve a rendszer megismerését futtatás nélkül is.

5.1. Belépés

Az alkalmazás első megnyitásakor egy letisztult bejelentkező képernyő fogadja a felhasználót. Mivel PWA (Progressive Web App) alkalmazásról van szó, a felület mobil eszközökön is natív élményt nyújt.

- Hitelesítés:** A "Belépés Google-lel" gombra kattintva a felhasználó a Google biztonságos oldalára irányítódik, majd sikeres azonosítás után visszakerül az alkalmazásba.



1. ábra: Bejelentkező képernyő

5.2. Térkép és Keresés

A sikeres belépés után a főoldalra, a térképnézetre jutunk. Itt láthatók a közösség által feltöltött parkolóhelyek.

- Markerek:** A piros tűk jelölik a parkolókat.
- Keresősáv:** A felső sávban név vagy cím alapján kereshetünk.

- Fejlett szűrő:** A szűrő ikonra kattintva lenyílik a panel, ahol beállíthatjuk a **keresési sugarat (Radius)** egy csúszkával, illetve szűrhetünk **kategóriára** (pl. csak Ingyenes vagy P+R).
- Sötét mód (Dark Mode):** A jobb felső sarokban található Nap/Hold ikonnal bármikor válthatunk a világos és sötét téma között. Ez nemcsak a felület színeit, hanem a **terkép stílusát** is azonnal módosítja (Mapbox Dark Style), ami éjszakai vezetésnél különösen hasznos.

Világos Mód (Light)	Sötét Mód (Dark)
2. ábra: Főoldal világos témával	3. ábra: Ugyanaz az oldal sötét témával fejlett szűrővel

5.3. Parkoló részletei és Navigáció

Ha a felhasználó rákattint egy markerre, egy információs ablak (Popup) jelenik meg.

- Információk:** Látható a parkoló neve, címe, kategóriája (színes címkével) és a feltöltött fotó.
- Távolság:** A rendszer kiszámolja és megjeleníti a felhasználó aktuális pozíciójától mért távolságot.
- Navigáció:** A "Tervezés ide" (nyíl) ikonra kattintva a telefon automatikusan megnyitja az alapértelmezett navigációs alkalmazást (Google Maps, Waze vagy Apple Maps) az útvonaltervel.



4. ábra: Parkoló részletei, fotóval és navigációs gombbal

5.4. Új parkoló felvétele

A "Hozzáadás" gombbal (vagy a térképen jobb klikkel) nyitható meg a rögzítő űrlap.

- **Helyszín pontosítása:** A felhasználó egy mozgatható "célkereszt" segítségével pontosíthatja a parkoló helyét a térképen.
- **Képfeltöltés:** Fotó készíthető a kamerával vagy feltölthető a galériából. A kép a háttérben a Firebase felhőbe kerül.
- **Validáció:** Az űrlap csak akkor küldhető be, ha minden kötelező adat (név, kategória) helyesen ki van töltve.

Új parkolóhely

Oszd meg másokkal a felfedezett helyet.

Megnevezés

Cím

Kategória

Helyszín

Fotó feltöltése



Budapest_Országház_(31355012995).jpg

Létrehozás

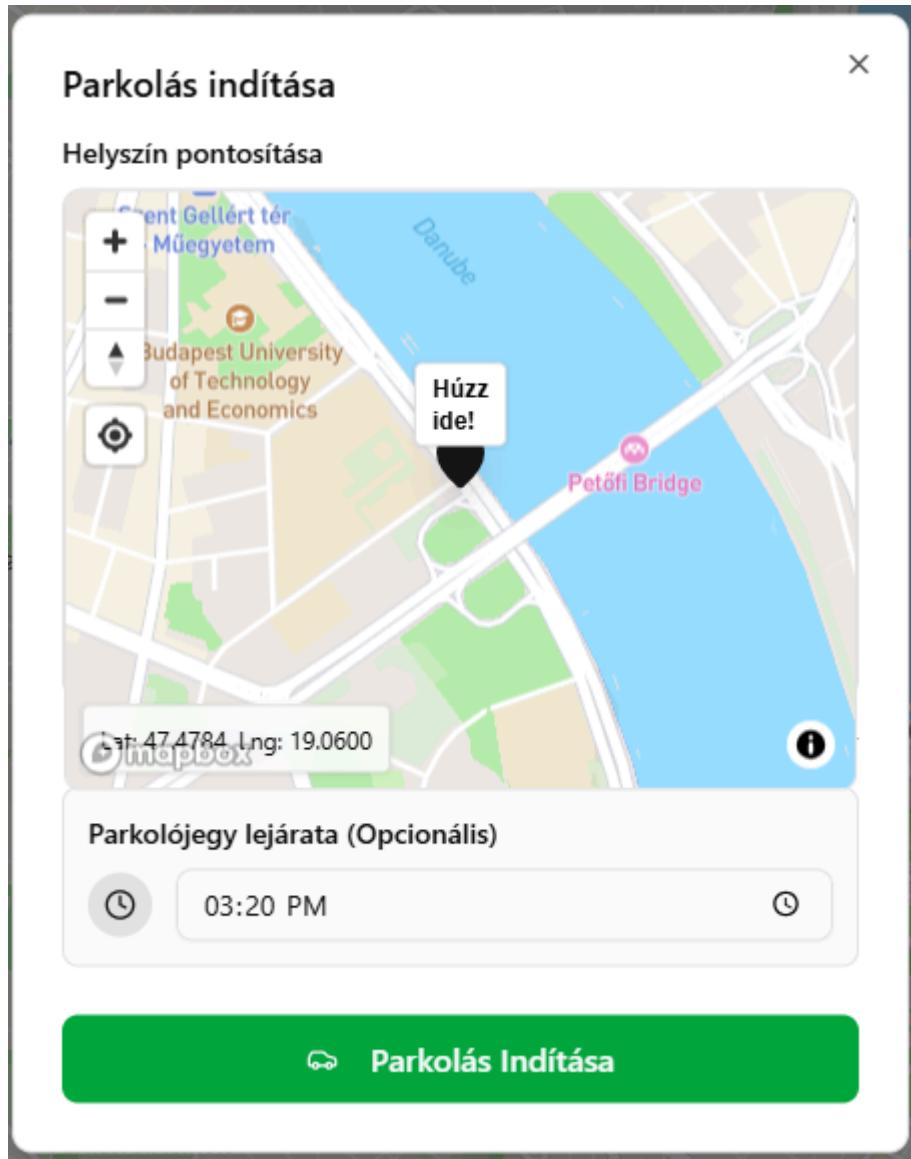
5. ábra: Parkoló hozzáadása űrlap térképes helyválasztóval

5.5. Parkolási folyamat (Session)

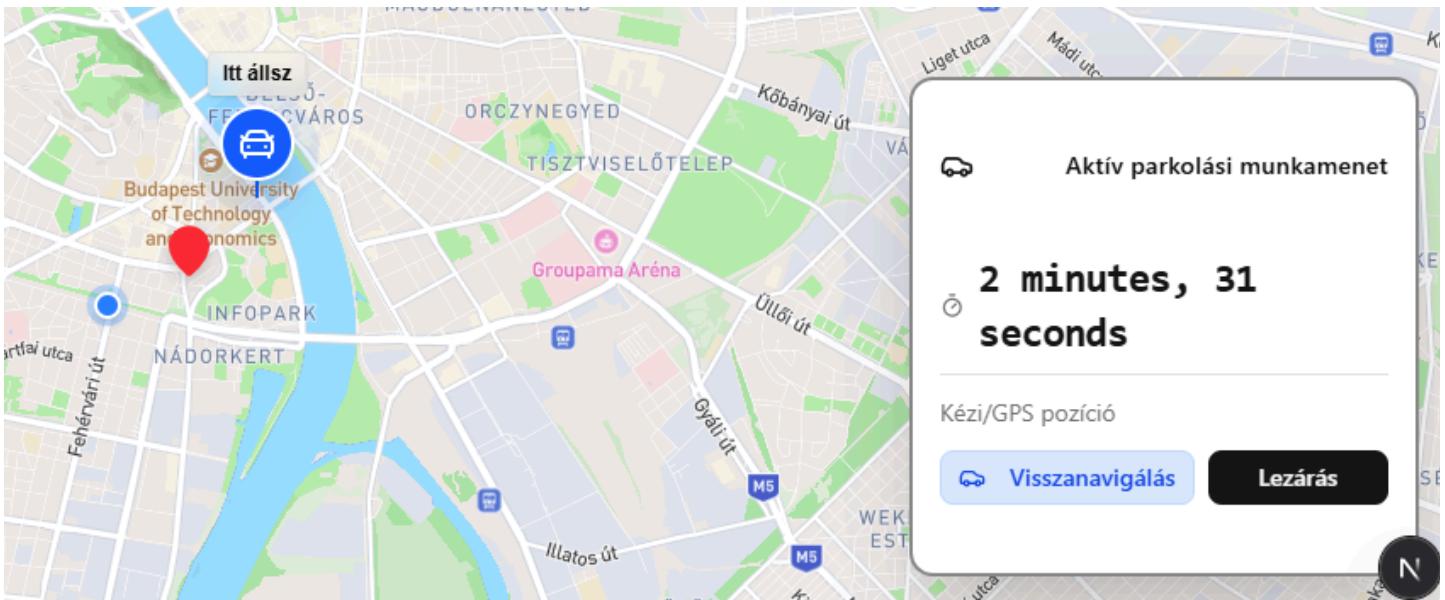
Ez az alkalmazás egyik legfontosabb funkciója.

- Indítás:** A jobb alsó sarokban lévő "Leparkoltam itt" gombbal indítható a parkolás. Opcionálisan beállítható a jegy lejáratú ideje.

- 2. Aktív státusz:** A térkép sarkában megjelenik egy kártya, amely mutatja, mióta parkolunk (vagy mennyi idő van hátra). A térképen egy "Autó" ikon jelzi a kocsi helyét.
- 3. Visszatalálás:** A "Visszatalálás" gomb megnyitja a navigációt a leparkolt autóhoz.
- 4. Értesítés:** Ha lejár az idő, a rendszer **Push Notification**t küld a telefonra (akkor is, ha a képernyő le van zárva).



6. ábra: Parkolásindítása ablak



7. ábra: Folyamatban lévő parkolás időzítővel és az autó pozíciójával

5. Továbbfejlesztési lehetőségek

Bár a rendszer jelenlegi formájában teljesíti a kitűzött MVP (Minimum Viable Product) követelményeket és stabilan működik, számos irányban bővíthető a funkcionális, a felhasználói élmény és az üzleti érték növelése érdekében.

5.1. Közösségi funkciók bővítése

A jelenlegi adatmodell (Review entitás) már előkészíti a lehetőséget az értékelésekre, de a felületen ez még nem jelenik meg.

- Értékelés és Kommentek:** A felhasználók 1-5 csillaggal értékelhetnek a parkolókat (biztonság, tiszta, ár-érték arány), és szöveges visszajelzést írhatnának.
- Jelentési rendszer (Report):** A közösség jelezhetné, ha egy parkoló megszűnt, az árak megváltoztak, vagy a leírás nem felel meg a valóságnak.

5.2. Fizetési rendszer integrációja

Jelenleg az alkalmazás csak információt nyújt a fizetős parkolókról. A következő lépés a tranzakciók kezelése lehetne.

- Mobilparkolás:** Integráció fizetési szolgáltatókkal (pl. Stripe, SimplePay vagy Nemzeti Mobilfizetési Zrt. API), hogy a felhasználók közvetlenül az alkalmazásból indíthassák és fizethessék a parkolást.
- Foglalási rendszer:** Lehetőség biztosítása garázsok vagy magánparkolók előre történő lefoglalására.

5.3. Adminisztrációs felület

A rendszer karbantartásához és a minőségbiztosításhoz szükség lenne egy dedikált Admin Dashboard-ra.

- Moderáció:** A feltöltött képek és új parkolók ellenőrzése a publikálás előtt (vagy utólagos moderáció).
- Felhasználókezelés:** Szabályszegő felhasználók tiltása.
- Statisztikák:** Hőterképek (Heatmaps) generálása a legnépszerűbb parkolási zónákról.

5.4. Valós idejű foglaltságjelzés

A legnagyobb hozzáadott értéket a szabad helyek valós idejű kijelzése jelentené.

- **Crowdsourcing alapú:** A felhasználók jelezhetik, ha egy parkoló "Megtelt".
- **IoT integráció:** Okosváros (Smart City) szenzorok adatainak bekötése API-n keresztül a P+R parkolók vagy parkoloházak telítettségéről.

5.5. Natív mobilalkalmazás

Bár a PWA technológia kiváló élményt nyújt, a natív funkciók (pl. Bluetooth a sorompónyitáshoz, mélyebb rendszerintegráció) érdekében érdemes lehet a Frontend kódállományát **React Native** vagy **Capacitor** segítségével natív iOS és Android alkalmazássá alakítani.

6. Összegzés

A féléves feladat keretében sikeresen megterveztünk és implementáltunk egy modern, skálázható **ParkolóKereső Rendszert**. A fejlesztés során a legkorszerűbb webes technológiákat (Next.js 15, NestJS, TypeScript, Docker) alkalmaztuk, kiemelt figyelmet fordítva a kódminőségre (Monorepo, Zod validáció, ESLint) és a felhasználói élményre (Mapbox, PWA).

A rendszer működőképes, támogatja a teljes parkolási folyamatot a kereséstől a navigáción át az értesítésekig, és szilárd alapot biztosít a későbbi továbbfejlesztésekhez.