Rendszerterv

Munkahelyi nyilvántartási rendszer

Rendszer célja és környezete

Feladatkiírás

A projekt célja egy olyan munkaidő napló webes alkalmazás létrehozása, amely megkönnyíti a projektek menedzselését, és azokról statisztikákat állít elő egy vállalat dolgozói számára.
A rendszerben felvehetőek projektek, amelyeknek van egy kezdési, és lehet egy tervezett befejezési dátuma. Ezek a projektek megtekinthetőek egy naptár nézetben, illetve az alkalmazottak minden eseményükhöz (pl. megbeszélés, feladatok) rögzíthetik, hogy melyik projekttel foglalkoztak. Emellett a dolgozók a naptárba felvehetik a tervezett szabadnapjaikat, amelyeket a közvetlen felettesüknek kell jóváhagynia.
A felettesek megnézhetik a beosztottjaik jóváhagyásra váró szabadnapjait, illetve amikor azok felvételre kerülnek a naptárba, arról E-Mail értesítést is kapnak. Opcionális célunk, hogy E-Mail mellett más értesítési formák is megvalósításra kerüljenek, elsősorban Discord értesítés formájában.

Funkciók

- **Projektek kezelése**

    - Új projekt létrehozása (név, kezdési dátum, tervezett befejezési dátum)
    - Projektek megtekintése (lista és részletes nézet)
    - Résztvevők hozzáadása a projekthez
    - Résztvevők eltávolítása a projektből
    - Projekt lezárása (tényleges befejezési dátum rögzítése)
- **Feladatok kezelése**
    - Feladat létrehozása projekthez rendelten
    - Feladat részleteinek megtekintése
    - Feladat hozzárendelése dolgozóhoz
    - Feladat hozzárendelés módosítása vagy törlése
    - Feladat státuszának változtatása (befejezett/folyamatban)
    - Feladat törlése (csak létrehozó által)
- **Munkaidő nyilvántartás**
    - Munkaidő bejegyzések rögzítése feladatokhoz
    - Rögzített adatok: dátum, óraszám, megjegyzés
    - Munkaidő bejegyzések megtekintése
    - Saját munkaidő bejegyzések törlése
- **Megbeszélések**
    - Megbeszélés létrehozása
    - Résztvevők hozzáadása a megbeszéléshez
    - Kezdési és befejezési időpont megadása
- **Szabadságok kezelése**
    - Szabadság igénylés létrehozása
    - Szabadság jóváhagyása vagy elutasítása (menedzser által)
    - Jóváhagyásra váró szabadságok listázása
- **Adminisztrációs felület (csak menedzsereknek)**
    - Dolgozók munkaidő adminisztrációjának megtekintése
    - Szűrés év és hónap alapján
    - Adminisztrált órák és kötelező órák összehasonlítása
    - Szabadságok figyelembevétele a kötelező órákból
    - Státusz jelzés (elegendő/hiányos/nincs bejegyzés)
- **Naptár nézet**
    - TODO: Fanni
- **Felhasználói rendszer**
    - Bejelentkezés Google OAuth-al
    - Bejelentkezés email és jelszóval
    - Szerepkör alapú jogosultságkezelés (Menedzser, Alkalmazott)

- **Értesítések**
  TODO: Geri
  - Email értesítések szabadság igénylésekről
  - Discord értesítések (opcionális)

Rendszer környezete

A rendszer egy modern webes alkalmazás, amely három fő komponensből áll. A központi alkalmazás egy Nuxt.js alapú full-stack web alkalmazás, amely a felhasználói felületet (Vue.js komponensek), az üzleti logikát és az API végpontokat egyaránt tartalmazza. Az alkalmazás PostgreSQL relációs adatbázist használ az adatok tárolására, amely Prisma ORM segítségével kerül elérésre.

A hitelesítés NextAuth (Auth.js) könyvtárral van megvalósítva, amely támogatja mind a Google OAuth alapú bejelentkezést, mind az email-jelszó páros hitelesítést. A munkamenetek JWT tokenekkel vannak kezelve, biztosítva a biztonságos felhasználói azonosítást. A rendszer szerepkör alapú hozzáférés-vezérlést (RBAC) implementál, három szerepkörrel: Admin, Menedzser és Alkalmazott.

Az értesítési szolgáltatásokat egy Elixir nyelven írt microservice kezeli, amely felelős az email és Discord értesítések kiküldéséért. Emellett egy Python alapú microservice gondoskodik a riportok és statisztikák generálásáról. A három komponens REST API-kon keresztül kommunikál egymással.

A rendszer Docker konténerekben futtatható, megkönnyítve a fejlesztést és a telepítést. A frontend modern, reszponzív felületet biztosít Nuxt UI komponenskönyvtár és TailwindCSS segítségével, támogatva a világos és sötét témákat is.

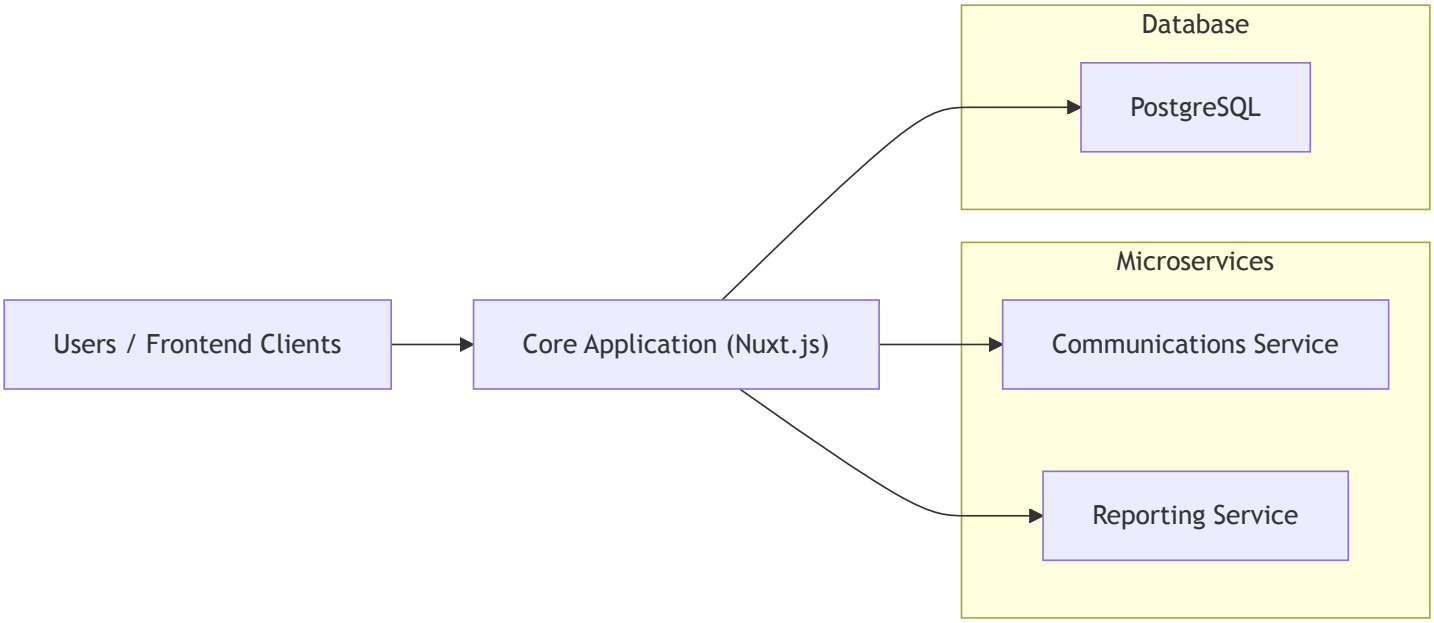
Architektúra

Architektúra áttekintése

Magas szintű architektúra

A rendszer mikroszerviz alapú architektúrát követ, amely három fő komponensből áll:

1. **Core Application** (Nuxt.js) - Központi webes alkalmazás
2. **Communications Service** (Elixir/Phoenix) - Értesítési mikroszerviz
3. **Reporting Service** (Python) - Riportgeneráló mikroszerviz



Architektúrális döntések és indoklásuk

Mikroszerviz architektúra

A rendszer mikroszerviz alapú felépítést követ, amely az alábbi előnyöket nyújtja:

Szeparált felelősségek: Minden mikroszerviz egy konkrét üzleti funkciót lát el (értesítések, riportok), amely egyszerűsíti a karbantartást és fejlesztést.

Technológiai függetlenség: Minden szolgáltatás a feladatához legmegfelelőbb technológiával készülhet:

- **Elixir/Phoenix** az értesítési szolgáltatáshoz - kiváló konkurencia kezelés, hibatűrés
- **Python** a riportgeneráláshoz - gazdag adatelemző és statisztikai könyvtárak (pandas, matplotlib)
- **Nuxt.js** a core alkalmazáshoz - teljes stack fejlesztés egyetlen keretrendszerben

Skálázhatóság: A komponensek egymástól függetlenül skálázhatók. Például az értesítési szolgáltatás külön skálázható nagy terhelés esetén.

Hibatűrés: Egy mikroszerviz hibája nem okozza a teljes rendszer leállását. Ha a riportgeneráló szolgáltatás nem érhető el, a core funkciók továbbra is működnek.

REST API kommunikáció

A mikroszervizek közötti kommunikációhoz REST API-kat választottunk:

Egyszerűség: A REST jól ismert, széles körben támogatott protokoll, amely megkönnyíti az integrációt.

Stateless kommunikáció: Minden kérés független, ami egyszerűsíti a rendszer állapotkezelését.

Platform függetlenség: Bármilyen HTTP kliens képes kommunikálni a szolgáltatásokkal.

Nuxt.js full-stack keretrendszer

A core alkalmazáshoz a Nuxt.js-t választottuk:

Unified codebase: Frontend és backend azonos nyelvben (TypeScript/JavaScript), csökkentve a kontextusváltást.

Server-Side Rendering (SSR): Jobb SEO és gyorsabb kezdeti oldalbetöltés.

File-based routing: Az API végpontok és oldalak könyvtárstruktúrán alapuló routing-ja egyszerűsíti a fejlesztést.

Auto-imports: Automatikus importálás csökkenti a boilerplate kódot.

Type safety: TypeScript támogatás az egész stacken keresztül.

PostgreSQL adatbázis

Relációs adatbázist választottunk az alábbi okok miatt:

ACID tulajdonságok: Tranzakciós biztonság kritikus a munkaidő-nyilvántartásnál.

Komplex kapcsolatok: A rendszerben számos entitás áll kapcsolatban (felhasználók, projektek, feladatok, munkaidő bejegyzések).

Strukturált adatok: Az adatok sémája jól definiált és stabil.

Prisma ORM integráció: Type-safe adatbázis hozzáférés, automatikus migrációk.

JWT alapú autentikáció

A munkamenet kezeléshez JWT tokeneket használunk:

Stateless: A szerver nem tárol session információkat, ami egyszerűsíti a horizontális skálázást.

Mikroszerviz kompatibilitás: A tokenek könnyen validálhatók a különböző szolgáltatásokban.

CredentialsProvider támogatás: Lehetővé teszi mind az OAuth, mind az email-jelszó alapú bejelentkezést.

Biztonságos: A tokenek titkosítva tartalmazzák a felhasználói információkat, rövidített élettartammal.

Rétegek leírása

A Core Application többrétegű architektúrát követ, amely egyértelműen elkülöníti a különböző felelősségeket. Az alkalmazás rétegei alulról felfelé:

Adatbázis réteg (Database Layer)

Ez a legalsó réteg, amely a PostgreSQL adatbázist tartalmazza. A Prisma ORM biztosítja az adatbázis séma kezelését és a migrációkat. Az adatbázis tárolja az összes perzisztens adatot: felhasználókat, projekteket, feladatokat, munkaidő bejegyzéseket és egyéb entitásokat.

Prisma séma (`/prisma/schema.prisma`):

- Típusbiztos adatmodell definíciók
- Kapcsolatok és megszorítások definiálása
- Automatikus migráció generálás

Adatmodell (Entity-Relationship Diagram):

Főbb entitások:

- **User**: Felhasználók, hierarchikus manager-employee kapcsolattal, szerepkör támogatással
- **Account**: OAuth provider fiókok (Google)
- **Session**: Felhasználói munkamenetek (JWT esetén nem használt)
- **Project**: Projektek kezdési és befejezési dátumokkal
- **UserProject**: Összekapcsoló tábla felhasználók és projektek között (many-to-many)
- **Task**: Többcélú entitás - feladatok, megbeszélések, szabadságok, egyéni feladatok
- **TimeEntry**: Munkaidő bejegyzések feladatokhoz rendelve
- **MeetingParticipant**: Megbeszélés résztvevők

Repository réteg (Data Access Layer)

A repository réteg biztosítja az adathozzáférési logikát. Minden entitáshoz tartozik egy repository, amely elrejti a Prisma specifikus implementációt.

Felelősségek:

- CRUD műveletek implementálása
- Adatbázis lekérdezések összeállítása
- Kapcsolódó entitások betöltése (includes, relations)
- Adatbázis hibák kezelése

Főbb repository-k:

- `userRepository.ts` - Felhasználók kezelése
- `projectRepository.ts` - Projektek kezelése
- `taskRepository.ts` - Feladatok kezelése
- `timeEntryRepository.ts` - Munkaidő bejegyzések kezelése
- `userProjectRepository.ts` - Felhasználó-projekt kapcsolatok

Service réteg (Business Logic Layer)

Jelenlegi implementáció:

A Core Application esetében a service réteg **jelenleg nincs külön kiválasztva**, mivel az alkalmazás főként CRUD műveleteket végez, ahol az üzleti logika minimális. A legtöbb endpoint esetében a logika elsősorban **validációból** áll (pl. autentikáció ellenőrzés, jogosultság ellenőrzés, input validálás), amely közvetlenül az API rétegben van implementálva.

Komplex üzleti logika elhelyezése:

A rendszer architektúrájában a jelentős üzleti logika a **mikroszervizekben** található:

- **Communications Service** (Elixir): Értesítési szabályok, email template generálás, Discord integráció logika
- **Reporting Service** (Python): Statisztikai számítások, riportgenerálás, adatelemzés

A Core Application felelőssége ezért elsősorban:

- Adatok perzisztálása és lekérése
- Felhasználói hitelesítés és jogosultságkezelés
- Alapvető input validáció

- Mikroszervizek koordinálása REST API hívásokon keresztül

Jövőbeli fejlesztés: Amennyiben a Core Application-ben több komplex üzleti logika jelenne meg (pl. összetett munkaidő számítások, automatikus projekt értékelések), akkor érdemes lenne a service réteget kiemelni az API handler-ekből a jobb szeparáció és tesztelhetőség érdekében.

API réteg (Route Handlers)

Az API réteg (`/server/api/`) tartalmazza a HTTP végpontokat. Nuxt.js file-based routing rendszert használ, ahol a fájlstruktúra határozza meg az endpoint-okat.

Felelősségek:

- HTTP kérések fogadása
- Autentikáció és autorizáció ellenőrzése
- Request validálás
- Repositoryk hívása
- Response formázás
- HTTP státuszkódok kezelése

Authentication réteg

Az autentikáció NextAuth (Auth.js) könyvtárral van implementálva (`/server/api/auth/[...].ts`).

Támogatott provider-ek:

- Google OAuth 2.0
- Email-jelszó (Credentials Provider)

JWT munkamenet:

- Stateless session kezelés
- Token tartalmazza: user id, email, name, role, image
- Biztonságos titkosítás a NEXTAUTH_SECRET-tel

Prezentációs réteg (Frontend)

A frontend Vue.js komponensekből áll, Nuxt.js keretrendszerben (`/app/`).

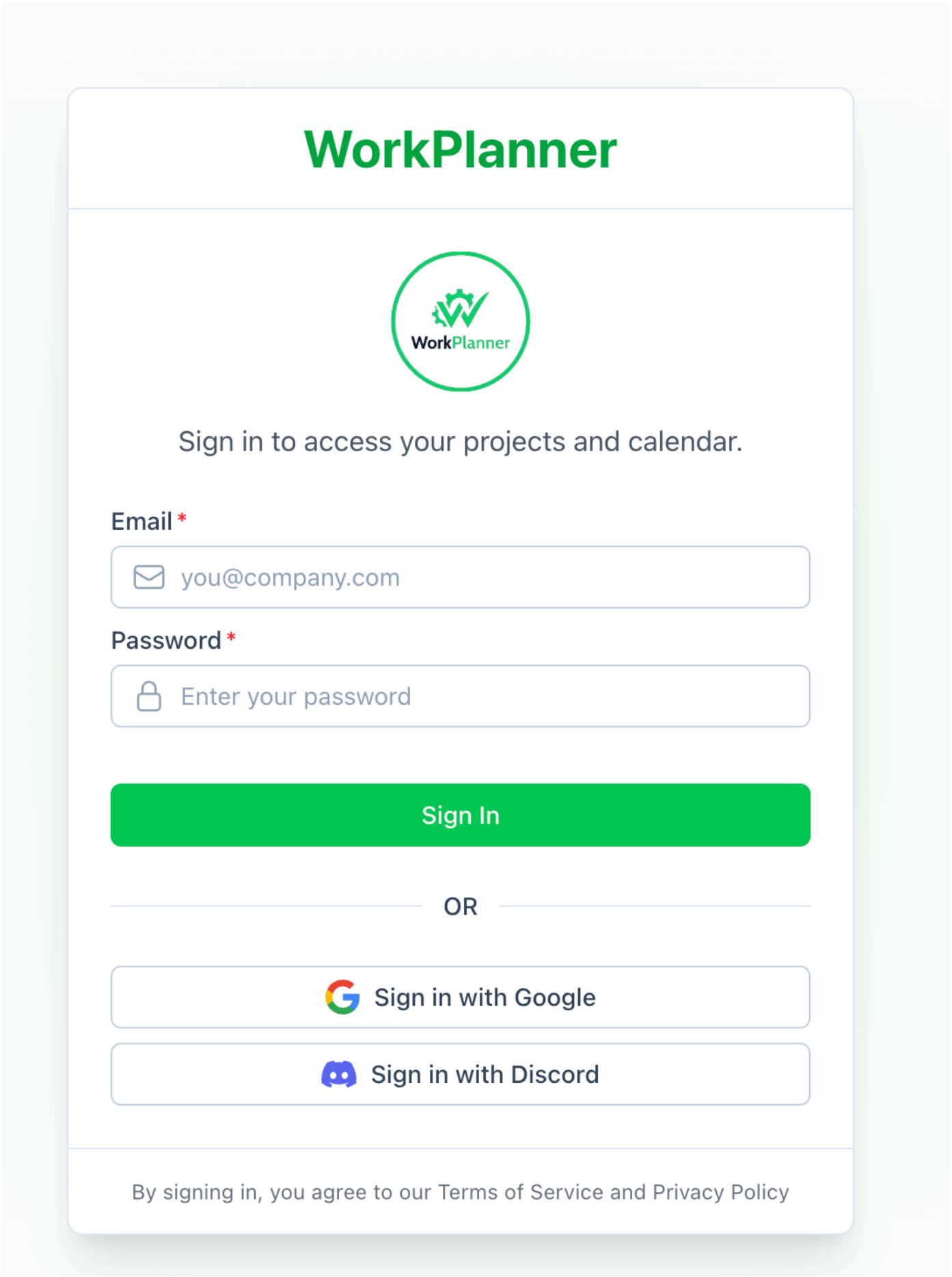
Komponensek (`/app/components/`):

- `ProjectCard.vue` - Projekt megjelenítés

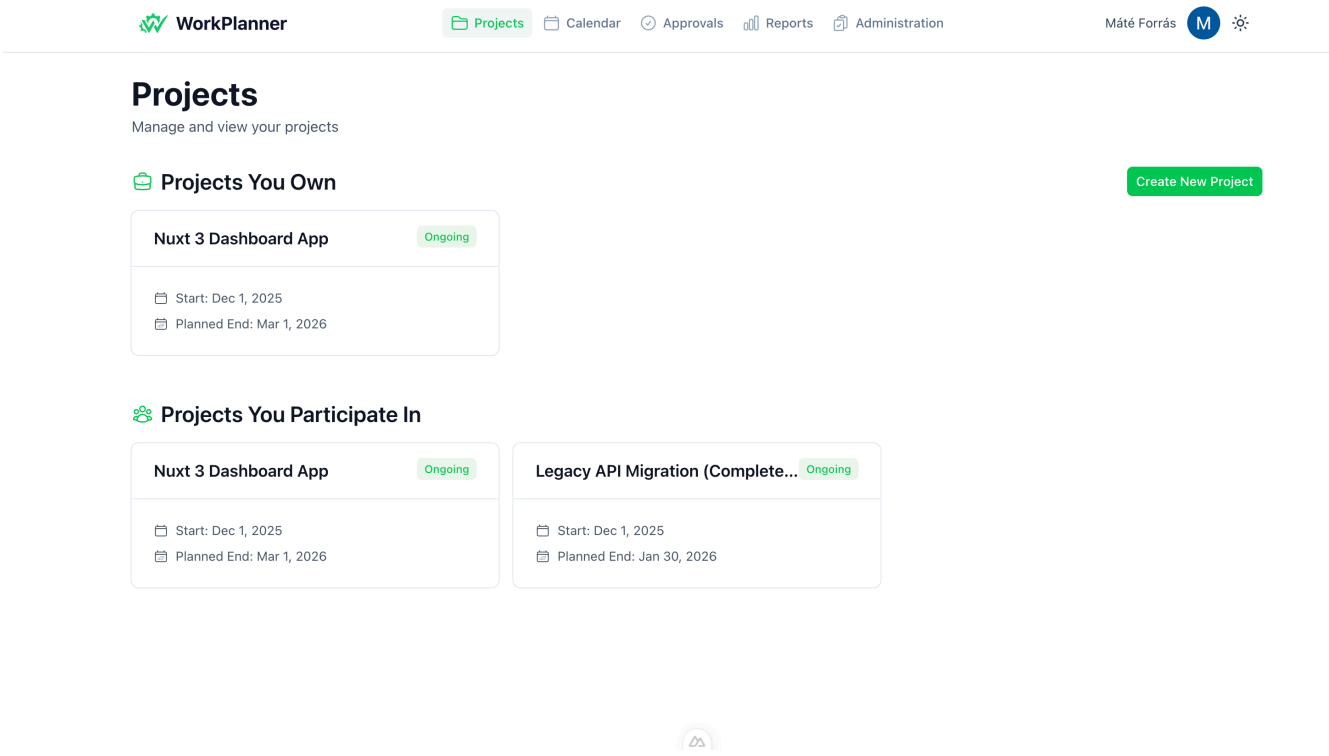
- `TimeAdministrationCard.vue` - Munkaidő adminisztráció lista elem
- `AddTimeEntryModal.vue` - Munkaidő rögzítés modal
- `TaskAssigneeModal.vue` - Feladat hozzárendelés modal
- És további komponensek...

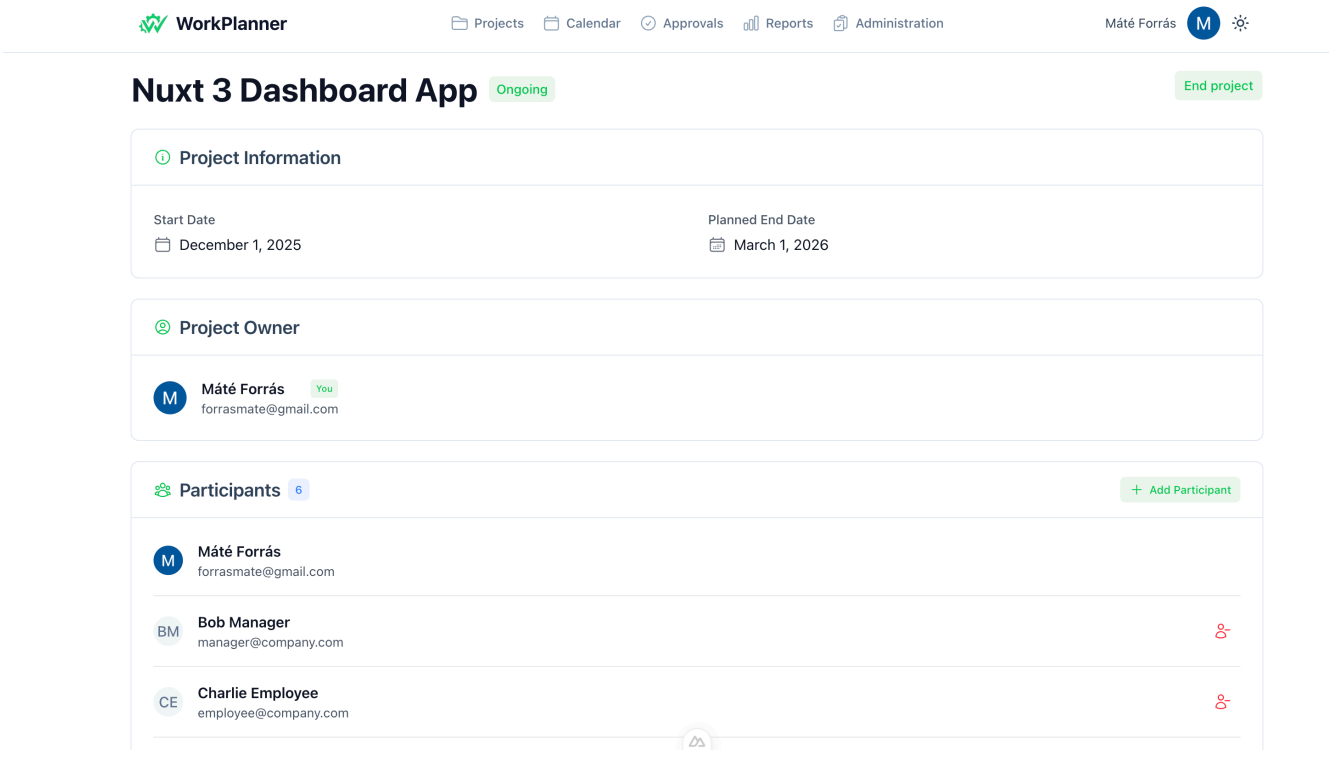
Oldalak (`/app/pages/`):

- `login/` - Bejelentkezés

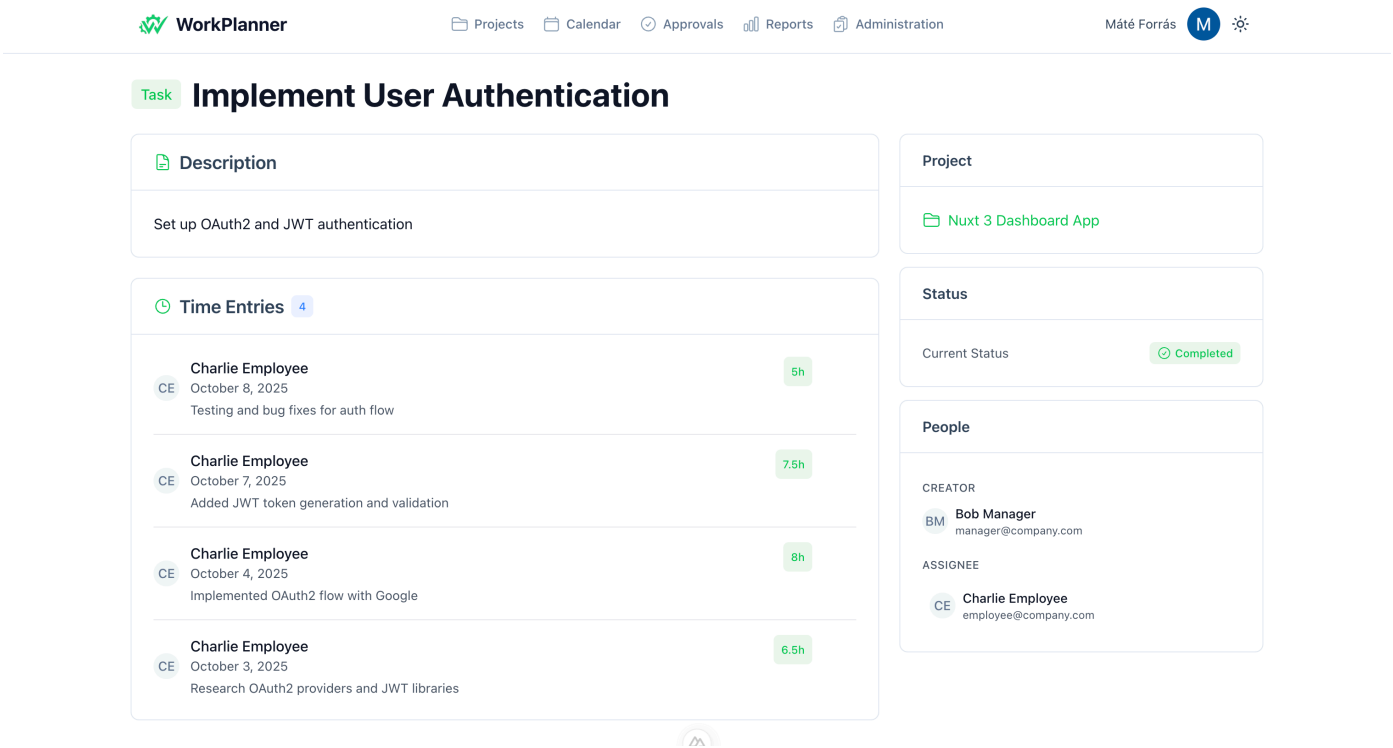


- `projects/` - Projekt menedzsment oldalak

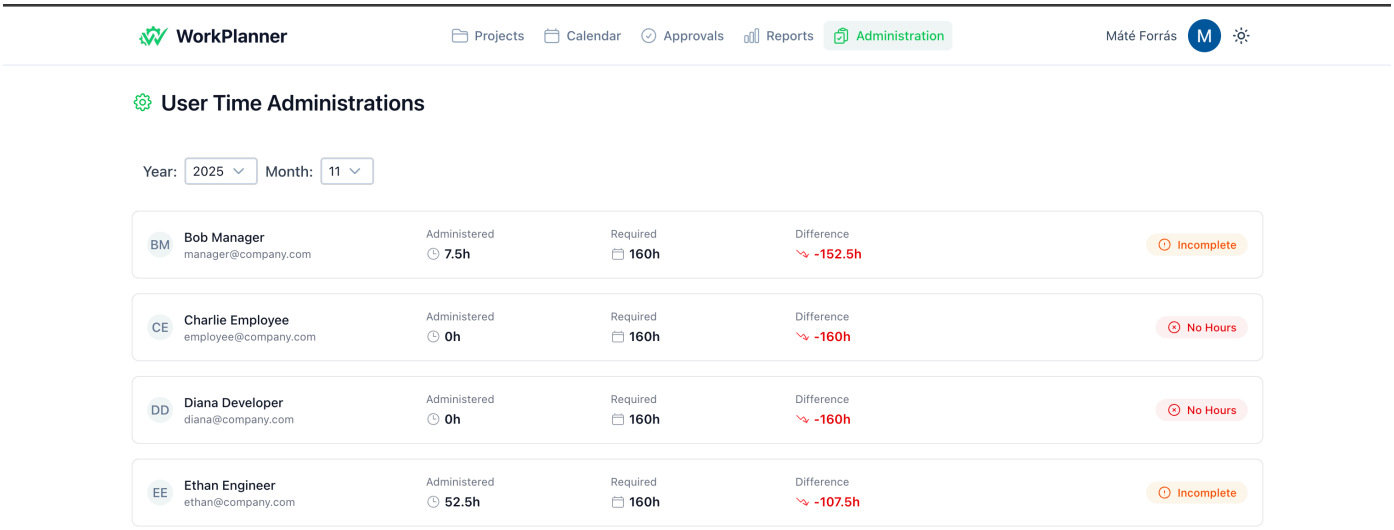




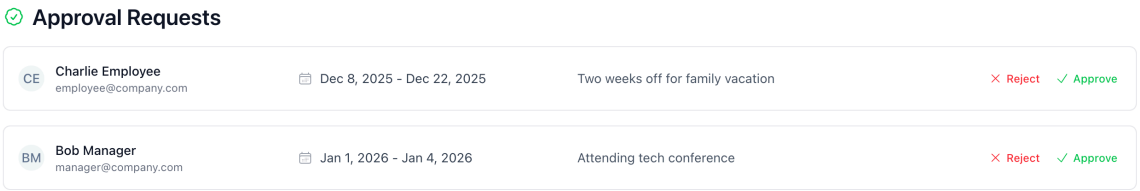
- `tasks/` - Feladat kezelés oldalak



- `administration/` - Adminisztrációs felület



- `approvals/` - Jóváhagyások



- `calendar/` - naptár nézet

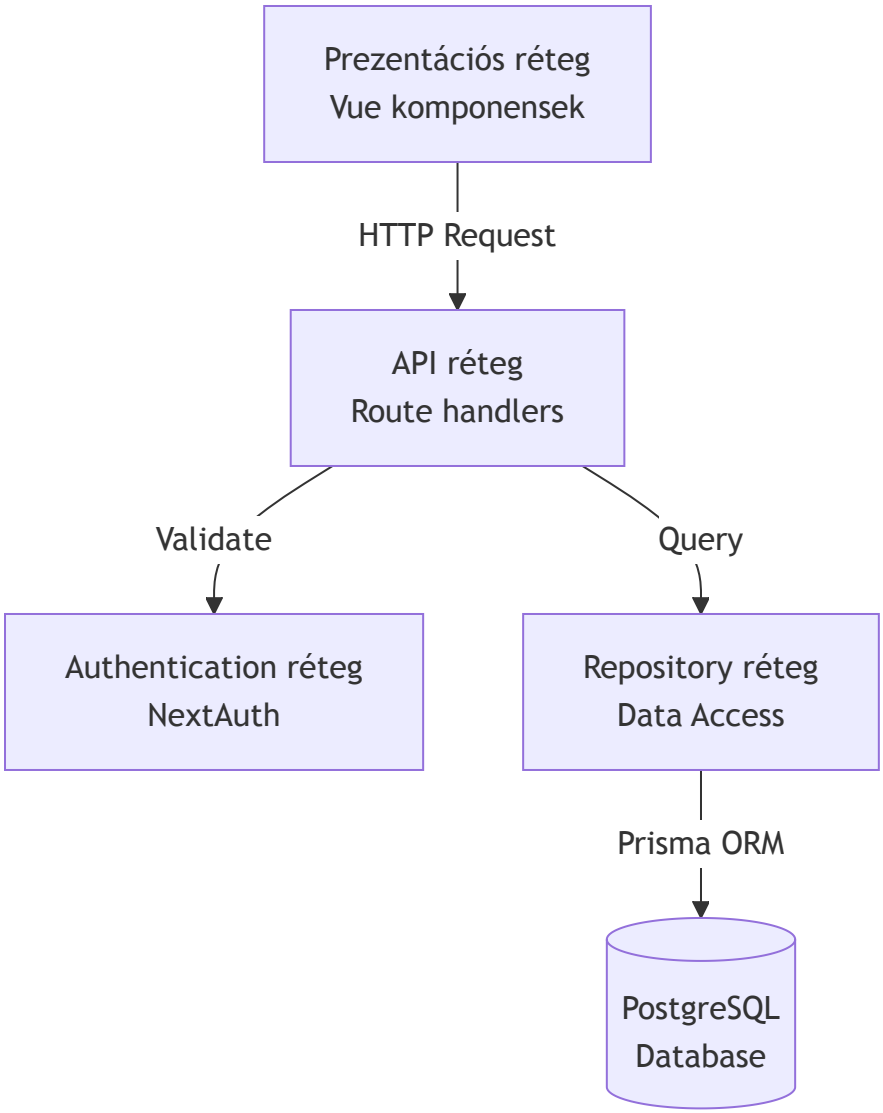
Composable-ök (`/app/composables/`):

- `useUser.ts` - Felhasználói session kezelés
- `localStore.ts` - Local storage kezelés

UI könyvtár: Nuxt UI komponensek TailwindCSS-el

- Sötét/világos téma támogatás
- Reszponzív design
- Formok, modal-ok, card-ok, dropdown-ok

Rétegek közötti kommunikáció



Elvek:

- **Separation of Concerns**: Minden réteg egyetlen felelősséggel rendelkezik
- **Dependency Direction**: Fentről lefelé (UI → API → Service → Repository → DB)
- **Abstraction**: Minden réteg elrejti az implementációs részleteket
- **Testability**: Rétegek külön-külön unit tesztelhetők

Mikroszervizek

A rendszer két különálló mikroszervizzel egészül ki, amelyek a Core Application-től függetlenül futnak és specifikus üzleti funkciókat látnak el.

Communications Service (Elixir/Phoenix)

Az értesítési mikroszerviz Elixir nyelven készült, Phoenix framework-öt használva. Felelős a rendszer összes kimenő kommunikációjáért.

Technológiai választás indoklása:

- **Konkurencia**: Az Elixir/BEAM VM kiválóan kezeli a párhuzamos folyamatokat, ami elengedhetetlen több értesítés egyidejű küldéséhez
- **Hibatűrés**: A "let it crash" filozófia és a supervisor tree-k garantálják, hogy egyedi hibák ne állítsák le a teljes szolgáltatást
- **Teljesítmény**: Kis memórialábnyom és gyors üzenetkezelés nagy terhelés esetén is

Felelősségek:

- Email értesítések küldése (szabadság igénylések, jóváhagyások)
- Discord webhook integráció
- Értesítési template-ek kezelése
- Üzenetsorok kezelése (retry logika hibás küldés esetén)
- Értesítési preferenciák alkalmazása

REST API végpontok:

- `POST /api/notifications/email` - Email küldés
- `POST /api/notifications/discord` - Discord értesítés
- `GET /api/notifications/status/:id` - Értesítés státusz lekérdezés

Megjegyzés: A részletes implementációt a Communications Service fejlesztői dokumentálják.

Reporting Service (Python)

A riportgeneráló mikroszerviz Python nyelven készült, adatelemzésre és statisztikák előállítására optimalizálva.

Technológiai választás indoklása:

- **Adatelemzés**: Gazdag ökoszisztéma (pandas, numpy) komplex statisztikai számításokhoz
- **Riportgenerálás**: Matplotlib, Seaborn visualizációs könyvtárak
- **Egyszerűség**: Gyors prototípus készítés és iteráció

Felelősségek:

- Munkaidő statisztikák generálása (havi, negyedéves, éves bontásban)
- Projekt előrehaladás riportok
- Dolgozói teljesítmény összesítők
- Exportálás különböző formátumokba (PDF, Excel, CSV)
- Adatvisualizáció (grafikonok, diagramok)

REST API végpontok:

- `POST /api/reports/time-summary` - Munkaidő összesítő generálás
- `POST /api/reports/project-progress` - Projekt előrehaladás riport
- `GET /api/reports/:id/download` - Riport letöltés

Megjegyzés: A részletes implementációt a Reporting Service fejlesztői dokumentálják.

Mikroszerviz kommunikáció

A Core Application REST API hívásokkal kommunikál a mikroszervizekkel:

Hibakezelés: Ha egy mikroszerviz nem érhető el, a Core Application továbbra is működik, de a felhasználó figyelmeztetést kap (pl. "Értesítés küldése sikertelen, később újrapróbálkozunk").

Telepítési leírás

Docker compose