## Kész / Jegyzet:

* **Jegyzőkönyv** **rögzítésre** került a felhasznált programokhoz / verziójukhoz / kiegészítőikhez,
* **Git License** hozzáadásra került a repohoz.
* Alap **Readme**.**md** feltöltésre került.

## Felh. programok verziói, kiegészítői:

* **Vue.js | 3.5.12 verzió |** 2024 október 11. dátummal.
* **VITE** v5.4.10

✅ Project name: ... Frontend

✅ Package name: ... frontend

✅ Add TypeScript? ... No / Yes

❌ Add JSX Support? ... No / Yes

✅ Add Vue Router for Single Page Application development? ... No / Yes

✅ Add Pinia for state management? ... No / Yes

✅ Add Vitest for Unit Testing? ... No / Yes

❌ Add an End-to-End Testing Solution? » No

❌ Add ESLint for code quality? ... No / Yes

✅ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes

# Frontend

## / src / components:

* **layouts** – Oldal elrendezések. [Főoldal + Header + Navbar layout-ok]
* **cards** – Kártya típusú komponensek.
* **sections** – Oldalszekciók.

## ’Alias’-ok felvétele:

'@': fileURLToPath(new URL('./src',

'@cards': './src/componensts/cards'

'@layouts':'./src/componenst/layouts',

'@sections': './src/componenst/sections',

## CSS / TailWindCSS

 npm install -D tailwindcss postcss autoprefixer

 npx tailwindcss init

## ****Modulok és Függőségek Telepítése és Használata****

Nest.js:  
A backend alkalmazás fejlesztéséhez a Nest.js keretrendszert használtam. Telepítettem a @nestjs/core, @nestjs/common és @nestjs/cli csomagokat, amelyek alapvetőek a Nest.js működéséhez. A backend oldalon a main.ts fájlban beállítottam, hogy a szerver a Fly.io által megkövetelt 8080 porton fusson.

## **Axios**:

A request csomag elavultsága és sérülékenységei miatt az axios csomagot választottam, mivel modernebb, karbantartott, és több funkcióval is rendelkezik a HTTP-kérések kezeléséhez. Az összes request használatot lecseréltem axios-ra, így biztonságosabb és karbantartottabb lett a kód.

## **Concurrently:**

A concurrently csomag lehetővé tette, hogy a frontend és backend szervereket egyszerre indítsam el helyi fejlesztési környezetben. Ezt egy egyedi dev parancsban állítottam be a gyökér package.json-ban.

## **Vulnerable Module Management:**

Több elavult függőséget (request, tough-cookie, hoek) vizsgáltam meg, és megpróbáltam frissíteni vagy helyettesíteni őket. Az npm audit figyelmeztetéseit elemeztem, és npm audit fix segítségével automatikusan javítottam a kisebb problémákat.

## ****Választások és Indoklás****

* Nest.js és Axios:

A Nest.js jól szervezett, moduláris keretrendszer, amely lehetővé teszi a skálázható alkalmazások építését. Az axios egy hatékony HTTP-kliens, amely modern API-t biztosít a kérések kezelésére, és biztonságosabb, mint a request, amely már elavult és sérülékeny.

* Concurrently:

Lehetővé tette, hogy a frontend és a backend szervereket egyszerre futtassam, így gyorsabban tudtam fejleszteni és tesztelni. A helyi fejlesztési környezet szempontjából nagyon hasznos volt, mivel egyetlen parancs kiadásával el tudtam indítani mindkét szervert.

## ****Fly.io Konfiguráció Frontend és Backend Számára****

## ****Frontend Beállításai****:

A frontend telepítése során figyeltem arra, hogy a **vite** és **tailwindcss** csomagok megfelelően legyenek konfigurálva.

A fly.toml fájlban megadtam a frontend alkalmazás portját, és beállítottam a Fly.io által megkövetelt **8080** port használatát **belső** **szerverként**. Ezen kívül a Dockerfile segítségével optimalizáltam a frontend környezetet a **Vite** **projekt** **buildeléséhez**.

## ****Backend Beállításai**:**

* + A backend számára egy Nest.js szervert állítottam be, amely a Fly.io által megadott környezeti változó alapján indul el a 8080 porton.
  + A Dockerfile-ban meghatároztam a szerver buildelését és futtatását, így biztosítva, hogy a Fly.io megfelelően kezelje a Node.js alapú alkalmazást.
  + A fly.toml-ban a backend internal\_port-ját 8080-ra állítottam, hogy a Fly.io szabványainak megfeleljen.

## ****Tesztelés és Hibakezelés**:**

* + A telepítések után a frontend és backend URL-eket ellenőriztem, és teszteltem a helyes működést. Figyeltem arra, hogy a megfelelő környezeti változókat használjam a PORT-hoz, és kezeltem a Fly.io által dobott 502-es hibákat.

Ezekkel a lépésekkel sikeresen konfiguráltam a frontend és backend alkalmazásokat Fly.io-n, és a helyi fejlesztési környezetben egyszerre tudom futtatni és tesztelni a két szervert a **concurrently** segítségével. A biztonsági auditok és a függőségek frissítése révén minimalizáltam a sérülékenységek kockázatát is.

## Következő lépés(ek):

## Turso Adatbázist összekötni a backenddel.

### Szeretnék elsőnek egy regisztrációs fület létrehozni, ahol bekérjük a :

* + Felhasználónevet,
  + e-mailt,
  + jelszót.

**Első körben ezt kéne letárolni az adatbázisban.**

## Adatbázis szerkezete (Turso-ban):

CREATE TABLE ***user\_registrations***

( ***id*** AUTO\_INCREMENT **PRIMARY** **KEY**,

***username*** VARCHAR(**20**) NOT NULL UNIQUE,

***email*** VARCHAR(**60**) NOT NULL UNIQUE,

***password*** VARCHAR(**16**) NOT NULL

);

## ****LibSQL Kliens Telepítése a Turso Adatbázishoz való hozzáféréshez + Token****

Mivel a Turso SQLite-kompatibilis, a libsql klienst használom mellé, mely támogatja az SQL-lekérdezések futtatását.

***npm install @libsql/client bcryptjs***

* + libsql telepítése a Backendben,
  + bcryptjs telepítése a titkosításhoz!

***SlibSQL\_kliens\_token.env fájl:***

* + Ez tartalmazza a Turso-tól megkapott TOKENT + db URL címét is.
  + eyJhbGciOiJFZERTQSIsInR5cCI6IkpXVCJ9.eyJhIjoicnciLCJpYXQiOjE3MzA1NTQyNzIsImlkIjoiZDljYmMxNGMtOGU4Yy00ZGJjLWE4MDUtNWNjN2QzNGJkY2QwIn0.d59unym\_93SvEY-JbKrA2Vlr9WzbMvKHX1xeieTTiuc1JGBY0-ctm94DAaMdPpOv4n5\_MSVc4JQO7YHd8FFxBA
  + libsql://userregistration-bari1192.turso.io

***Src/database\_services mappa:***

* + database.services.ts-ben token és adatbázis url átadása.