Szakdolgozat

Miskolci Egyetem

A Vue, az Angular és a React Javascript keretrendszerek összehasonlítása

Készítette:

Bujnóczki Bence

Programtervező informatikus

Témavezető:

Agárdi Anita

Miskolc, 2023

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Szakdolgozó Neve (B00P5Y) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: kulcsszavak, hasonlók

A szakdolgozat címe: A Vue, az Angular és a React Javascript keretrendszerek összehasonlítása

A feladat részletezése:

A feladat során a három jelenleg legnépszerűbb frontend keretrendszerek kerülnek bemutatásra. Az összehasonlítás fő szempontjai: teljesítmény, fájlméret, és szintaxisbeli különbségek. A feladathoz három webes alkalmazás készül, amelyek működése megegyezik és a segítségükkel mérhetőek és kimutathatóak a keretrendszerek közötti különbségek. A program a nap pozícióját jelzi a megadott időben és helyen. Ezen kívül olyan teszteket tartalmaz, amivel a teljesítmény mérhető.

Témavezető: Agárdi Anita (tanársegéd)

A feladat kiadásának ideje: 2021. szeptember 27.

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Bujnóczki Bence**; Neptun-kód: `B00P5Y` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *A Vue, az Angular és a React Javascript keretrendszerek összehasonlítása* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

A szakdolgozatom a három jelenleg legnépszerűbb frontend keretrendszer összehasonlításával foglalkozik. A projekt fókuszában a Vue, Angular és React állnak, melyek az elmúlt években a legnépszerűbb frontend technológiák lettek.

Mindhárom keretrendrendszerben elkészítettem ugyanazt a programot, hogy egyenlő feltételek mellett tudjam összehasonlítani a keretrendszereket. A program témája a Nap, mint égitest adatainak ábrázolása szövegesen és grafikusan is. Az ábrázolt adatok közé tartozik a napfelkelte és naplemente időpontja, nap hossza és egy grafikon is, ami a Nap pályáját jelöli. A kódot úgy próbáltam elkészíteni, hogy a leggyakoribb frontendes problémák be legyenek mutatva. Mint például az eseménykezelés, hálózati kérés, lista renderelés, komponensek közötti kommunikáció és beállítások eltárolása a böngésző memóriájában.

A frontend keretrendszerek olyan eszközök, amelyek segítségével a webalkalmazások fejlesztése egyszerűbbé és hatékonyabbá válik. A keretrendszerek célja, hogy elősegítsék az újrafelhasználható, jól strukturált, karbantartható és kiterjeszthető kódbázis kialakítását. A következőkben néhány olyan okot sorolok fel, hogy miért használnak frontend keretrendszereket a fejlesztők:

- Komponens alapú struktúra: A keretrendszerek lehetővé teszik a fejlesztők számára, hogy a webalkalmazásokat különálló komponensekre bontsák, és ezeket újra felhasználják más részekben vagy más alkalmazásokban is. Ez nagyban megkönnyíti a kód karbantartását, fejlesztését és bővítését.

- Jobb teljesítmény: A frontend keretrendszerek sok esetben jobb teljesítményt biztosítanak, mint a hagyományos JavaScript megoldások. Ezek az eszközök optimalizálják a DOM manipulációt, minimalizálják a felesleges újra renderelést, és segítenek megelőzni a lassú, blokkoló felhasználói felületet.

- Hatékonyabb adatkezelés: A frontend keretrendszerek segítségével könnyen lehet kezelni a dinamikus adatokat, amelyek folyamatosan változnak az alkalmazás életciklusa során. A keretrendszerek biztosítanak különféle funkciókat az adatok állapotának nyomon követéséhez és a különféle adatmanipulációkhoz.

- Nagyobb közösségi támogatás: A frontend keretrendszereknek általában nagy közössége van, amely folyamatosan fejleszti és karbantartja azokat. A közösségi támogatásnak köszönhetően a fejlesztők számára könnyebbé válik az új funkciók bevezetése, a hibajavítás és az általános támogatás.

- Alkalmazáskeretekkel való integráció: A frontend keretrendszerek sok esetben integrálhatók más alkalmazáskeretekkel, például a backend keretrendszerekkel, a mobilalkalmazás keretrendszerekkel és az adatbázisokkal. Ez megkönnyíti a teljes alkalmazás fejlesztését és integrációját.

A három keretrendszer mind nagyon népszerű, és különböző előnyökkel és hátrányokkal rendelkezik. Az összehasonlításom során feltárom az egyes keretrendszerek különbségeit és hasonlóságait, hogy segítségül szolgáljak azoknak, akik úgy döntenek, hogy ezek közül választanak.

Összességében a szakdolgozat egy nagyszerű lehetőséget kínált számomra, hogy bővítsem a tudásomat és új technológiákat ismerjek meg. A programozás során felfedezhetem a keretrendszerek specifikus tulajdonságait. A példaprogram saját ötletem, amit már régóta szerettem volna megvalósítani. Ügyeltem arra, hogy az alkalmazás önmagában is értelmes, használható legyen, ne csak az összehasonlítást szolgálja.

2. fejezet

Koncepció

2.1. Felhasznált technológiák

A dolgozatomban három frontend keretrendszert használtam. Mindegyikhez az ajánlott plugineket és eszközöket használtam fel.

2.1.1. Single-page application

A single-page application (SPA) egy olyan webalkalmazás, ami úgy kommunikál a felhasználóval, hogy az oldal tartalmát dinamikusan változtatja. Így nem szükséges a teljes oldal újratöltése, elegendő a változtatandó adatokat lekérni a szervertől. Ettől az oldal sokkal gyorsabbá válik, kevesebb a hálózaton átvitt adat mennyisége és az alkalmazás sokkal inkább natív appnak érződik.

Mivel nem töltődik újra az oldal, ezért az oldalváltásokat egy frontenden futó routernak kell kezelnie. Mindhárom keretrendszer rendelkezik ilyennel.

2.1.2. JavaScript / TypeScript

A JavaScript egy dinamikus, interpretált programozási nyelv, amelyet a webes alkalmazások kliensoldali és szerveroldali fejlesztésére használnak.

A TypeScript egy nyílt forráskódú nyelv, amely kiterjeszti a JavaScriptet típusokkal. A TypeScript azonban nem önálló nyelv, hanem JavaScriptre fordul le. Segít a fejlesztőknek a hibák korai észlelésében, növeli a kódbázis biztonságát és olvashatóságát. Ezen előnyök miatt döntöttem a használata mellett.

2.1.3. Node és NPM

A Node.js egy szerveroldali JavaScript futtatókörnyezet, amely lehetővé teszi a JavaScript programozási nyelv használatát szerveroldali alkalmazások fejlesztéséhez. A Node.js-t a frontendes kód csomagolására használtam fel.

Az npm (Node Package Manager) az alapértelmezett csomagkezelő a Node.js-hez, amely lehetővé teszi a különböző Node.js modulok és függőségek telepítését és kezelését.

Az npm legfontosabb parancsai:

- npm install: Ezzel a paranccsal telepíthetünk egy vagy több csomagot a projektünkhöz. A telepített csomagok automatikusan hozzáadódnak a package.json fájlhoz és a node_modules mappába.

- npm uninstall: Ezzel a paranccsal eltávolíthatjuk a telepített csomagokat a projektünkből.
- npm run: Ezzel a paranccsal futtathatjuk az alkalmazásunkat, illetve saját scripteket is definiálhatunk a package.json fájlban. A vue-s alkalmazás az serve scripttel, az anuglar-os és react-es pedig a start scripttel indítható el fejlesztői módban.

2.1.4. Chart.js

A Chart.js [4] egy JavaScript-ben írt grafikonrajzoló könyvtár. Használata egyszerű és jelenleg is aktívan fejlesztik. Az adatpontok megváltozása esetén animálja az átmenetet. Az egyszerű használat és a jó teljesítmény miatt döntöttem a használata mellett.

2.1.5. SunCalc

A SunCalc [17] egy kisméretű könyvtár a nap és a hold pozíciójának kiszámítására. A számításhoz a földrajzi koordinátákra és az időpontra van szüksége. Ezekből az adatokból kiszámítja az adott pillanatban és helyen a nap (vagy a hold) horizonttal bezárt szögét, valamint azimutját. Ezenkívül az adott napra vonatkozóan kiszámítja a napkelte, a naplemente, a különböző szürkületi időpontokat a delelés és a nadír időpontját.

2.1.6. Vue

A Vue [20] egy JavaScript keretrendszer amely segítségével felhasználói felületek készíthetőek. HTML, CSS és JavaScript-re épül. Egy deklaratív és komponens alapú programozási modellt biztosít, amely segítségével hatékonyan és gyorsan fejleszthetők alkalmazások. Egy egyszerű alkalmazást az alábbi módon hozhatunk létre. A `createApp` függvény készíti el az alkalmazás példány, a `mount` pedig a weboldal egy megadott eleméhez csatolja. A Vue és hozzá kapcsolodó csomagok fejlesztését egy nemzetközi csapat végzi.

```
import { createApp } from 'vue'

createApp({
    data() {
        return {
          count: 0
        }
    }
}).mount('#app')
```

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Vue CLI

A Vue CLI [18] egy parancssori eszköz. Segítségével új projektek hozhatóak létre templateből, beállítja a webpack fordítást és különböző pluginekkel funkciókat ad hozzá a

projekthez. Jelenleg már a create-vue csomag az ajánlott projekt létrehozási módszer, azonban a dolgozatom írásának kezdetén még a vue-cli volt.

Példa projekt létrehozásra:

```
1     vue create hello-world
```

Vue Router

A Vue Router [21] az alapértelemezett router vue alkalmazásokhoz.

Pinia

A Pinia [11] a vue3-hoz ajánlott store csomag. Segítségével az alkalmazás globális állapota egy központi helyen tárolható és elérhető a különböző komponensekből és oldalakról. A más store-okhoz képest (pl.: Vuex, Redux) egyszerűbb a szerkezete. Composition API-szerű szintaxist használ és nem szükséges minden változtatáshoz külön mutationt létrehozni. A Pinia és a Vuex különbségei:

- Nincsenek mutation-ök. Általában a mutation-ök ismétlődőek és felesleges kódot eredményeztek. Korábban a devtools miatt is volt rájuk szükség, az állapotváltozások követése miatt, azonban a vue 3-ban már enélkül is megoldható az állapot követése.
- A TypeScript teljes mértékben támogatott, működik az autocomplete
- A storeok (modulok) automatikusan dinamikus működésúek.
- Nincsenek explicit store modulok, de importálással használhatóak együtt a különböző részek.

2.1.7. React

A React [12] egy nyílt forráskódú JavaScript könyvtár, amely felhasználói felületek fejlesztésére szolgál. Az alkalmazásokat komponensek építik fel, amelyeket újra fel lehet használni és dinamikusan változtatni lehet őket az adatok alapján. A templatekhez JSX-et használ. A JSX (JavaScript XML) egy olyan kódolási nyelv, amely lehetővé teszi a HTML-szerű kódszerkezetek írását JavaScript kódban. Fejlesztője a Facebook (Meta).

```
1 const root = ReactDOM
2     .createRoot(document.documentElement('root'));
3 root.render(<h1>Hello, world!</h1>)
```

create-react-app / craco

A create-react-app a hivatalosan támogatott módja az SPA React alkalmazások létrehozásának. Egy modern build konfigurációt biztosít, beállítás nélkül.

React Router

A React router [13] az alapértelmezett router react alkalmazásokhoz.

Redux

A Redux [14] a Reacthez leggyakrabban használt store megoldás.

2.1.8. Angular

Az Angular [1] egy JavaScript keretrendszerer, amelyet a Google fejleszt. A célja, hogy segítségével könnyebben és hatékonyabban lehessen webes alkalmazásokat készíteni. Az Angular az MVC (Model-View-Controller) tervezési mintát követi.

ng (cli)

Az ng az Angular parancssori eszköze. Az alkalmazások fejlesztéséhez, létrehozásához és teszteléséhez szükséges eszközöket biztosít.

Angular Router

Az Angular Router [2] az alapértelemezett router angular alkalmazásokhoz.

RxJs

Az RxJS (Reactive Extensions for JavaScript) [15] egy nagyon hasznos JavaScript könyvtár, amely lehetővé teszi az aszinkron adatfolyamok kezelését. Az angular nélkül is működik, ezért nem angular projektekben is felhasználható. Fő alkotóelemei:

- Observables: figyelhető elemek gyűjteménye, amik a jövőben értékeket adnak vissza
- Observer: callback függvények gyűjteménye, amik egy Observable-t figyelnek
- Subscription: Egy Observable futását tárolja. Használható a futtatás megszakítására
- Operators: Függvények, amik segítségével funkcionális programozási stílusban kezelhetőek gyűjtemények. Pl.: map, filter, concat, reduce

NgRx

Az NgRx [10] az angularhoz ajánlott store megoldás.

2.2. Jövőbeli fejlesztések

Az alkalmazás továbbfejleszthető mobilos alkalmazássá. PWA támogatás hozzáadásával, offline használat is lehetséges, és telepíthetővé válik az app. Akár natív mobilos alkalmazássá is konvertálható, ha a PWA-t becsomagoljuk pl.: apk-ba.

Időzóna támogatás is egy fontos fejlesztés lehet a jövőben. Jelenleg az összes időpont a felhasználó lokális ideje szerint jelenik meg. A legtöbb helyen a `luxon` csomaggal vannak kezelve a dátumok, ami támogatja az időzónák kezelését a megfelelő beállításokkal.

3. fejezet

Tervezés

Az alkalmazás fő célja a keretrendszerek közötti különbség megmutatása. Azonban másodlagos célként, szerettem volna egy olyan alkalmazást, ami megmutatja a Nap pályáját, és ezzel kapcsolatos adatokat szövegesen és grafikusan is.

A tervezésnél az alábbi szempontokat vettem figyelembe:

- A főoldalon látható legyen a napfelkelte, naplemente és naplemente időpontja, valamint a nap hossza.
- A háttér dinamikusan változzon, nappal kék színű, éjszaka fekete
- A földrajzi helyzet megadható legyen koordinátákkal, vagy településkeresővel is.
- Legyen egy grafikus ábra, ami ábrázolja a Nap pályáját
- Legyen egy folyamatjelző ami százalékban jelzi, hogy mennyi telt el az adott napból (napkelte és napnyugta között)
- Reszponzív legyen az alkalmazás és mobilon is használható legyen
- Legyen egy listanézet, ahol több nap adati is látszanak, a dátumok beállíthatóak legyenek.
- A teljesítmény minnél jobb legyen. (Gyors oldalbetöltés, a grafikonok gyors kirajzolása, frissítése)
- A jövőben bővíthető legyen a Hold adatival is

3.1. Keretrendszerek népszerűsége

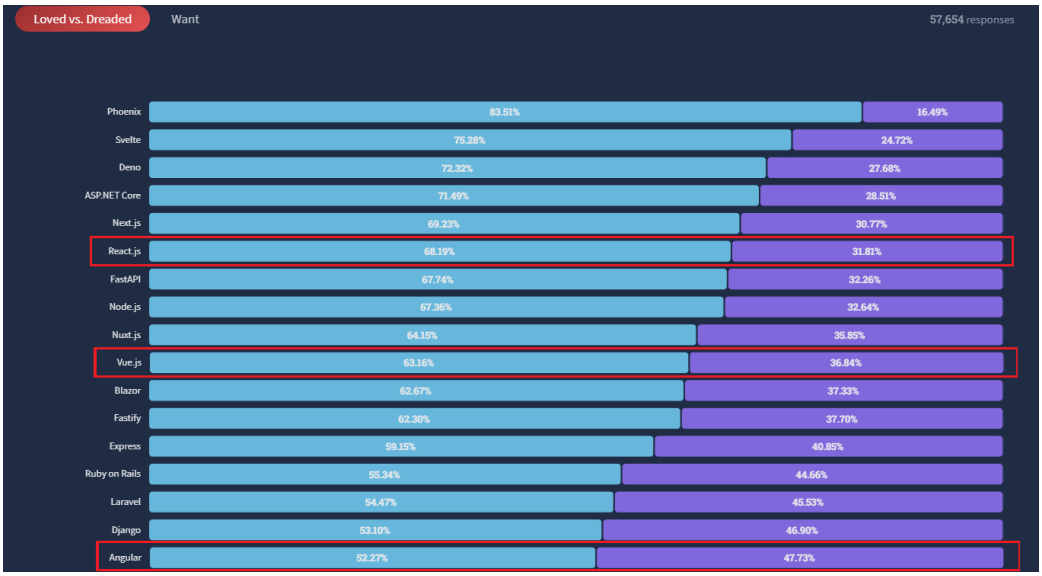
A népszerűség összehasonlításához kétféle statisztikát használtam. Az egyik a github csillagok száma, a másik a stackoverflow [16] 2022-es felmérése a kedvelt keretrendszerekről.

- A Vue-nak külön repoja van a 2-es és a 3-as verziókhoz. A 3-as verzió a legújabb, emiatt ott még kevessebb csillag gyűlt össze. Az itt [9] található csillagok száma 37 ezer, míg a régebbi repoban [8] 203 ezer van.
- A React repoban [7] 207 ezer csillag van.

- Az Angular projekten [6] 88 ezer csillag van.

A stackoverflow statisztikája azt mutatja hogy a válaszolók közül mennyien kedvelik az adott keretrendszert.

- React: 15873 válaszolóból 68.19% kedveli
- Vue.js: 6492 válaszolóból 63.16% kedveli
- Angular: 5822 válaszolóból 52.27% kedveli



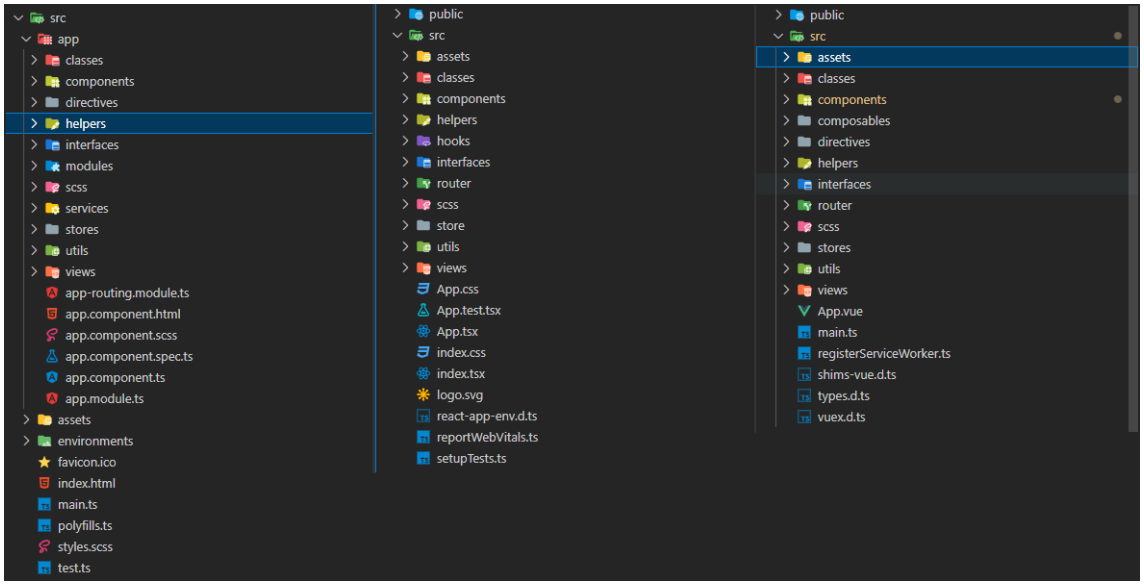
3.1. ábra. Statisztika a kedvelt / utált webes keretrendszerekről

3.2. Projektek felépítése

Minden projektnél testreszabható a felépítés, viszont én az alapbeállításokat használtam, hogy látszódjon ezek különbsége.

- `public`: Ide kerül az oldalhoz tartozó index.html és egyéb metafájlok. Pl.: favicon. Angularnál nincs `public` könyvtár, itt ezek a fájlok az `src`-n belül találhatóak.
- `src`: Az alkalmazás forrásfájlait tartalmazó könyvtár
- `app`: Ez a könyvtár csak angularnál van. A szigorúan vett forrásfájlokat tartalmazza. Pl.: az assets ezen kívül van.
- `assets`: Statikus fájlokat, pl.: ikonokat tartalmazó könyvtár
- `classes`: TypeScript osztályokat tartalmazó könyvtár
- `components`: Komponenseket tartalmazó könyvtár. Angularnál az egyes komponensek külön könyvtárba kerülnek, mert a komponensek 3 fájlból épülnek fel (html, scss, ts)

- `interfaces`: TypeScript interfészeket tartalmazó könyvtár
- `modules`: Csak Angularnál van, Angular modulokat tartalmazó könyvtár.
- `scss`: Stílusfájlokat tartalmazó könyvtár
- `composables / services / hooks`: Ide kerülnek a komponenshez nem tartozó, levélasztott logikák.
- `stores`: Store-okhoz kapcsolódó fájlokat tartalmazó könyvtár
- `router`: Routert leíró fájlokat tartalmazó könyvtár. Angularnál alapból nincs ilyen könyvtár, a router fájl az app/app-routing-module.ts.
- `views`: Nézetekhez tartalmazó könyvtár. Pl.: HomeView, ListView
- `main.ts / index.ts`: Az alkalmazás belépési pontja
- `App.vue / app.component / App.tsx`: Fő komponens, ez tartalmazza az oldal keretét és a router nézetet.

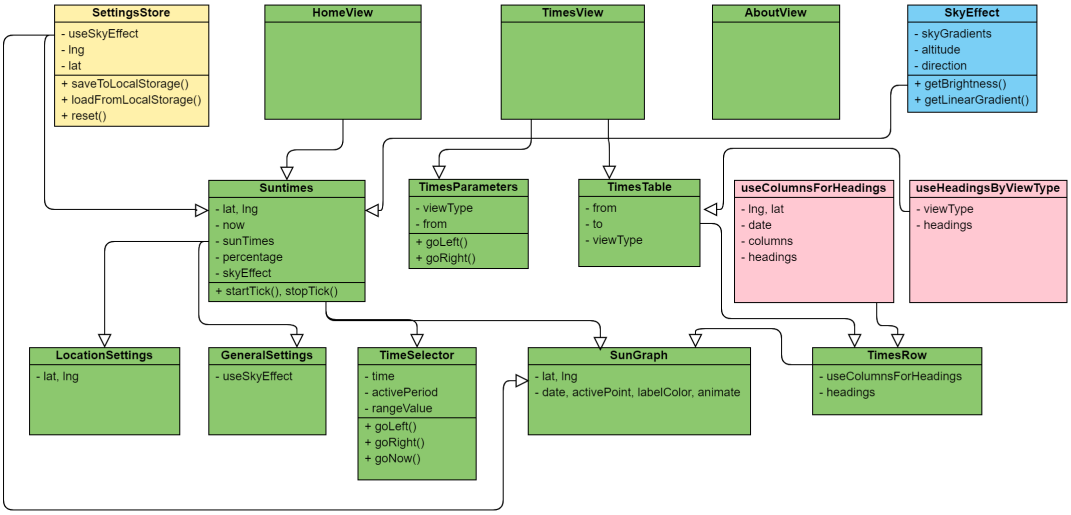


3.2. ábra. Projektek felépítése. Balról jobbra: Angular / React / Vue

3.3. Kódstruktúra

Az alábbi UML diagram ábrázolja a kód moduljainak felépítését. A jelölések a következők:

- Sárga: Store modul
- Zöld: Komponens
- Kék: Osztály
- Rózsaszín: Composable / Hook / Service



3.3. ábra. UML diagram

3.4. UI design szempontok

A webes alkalmazások reszponzív tervezése az egyik kulcsfontosságú szempont, amelyre figyelemmel kell lenni a modern webfejlesztés során. A reszponzív tervezés azt jelenti, hogy az alkalmazás megjelenése és működése alkalmazkodik a felhasználó által használt készülék képernyőméretéhez és eszközállásához. Néhány fontos szempont, amit érdemes figyelembe venni a webes alkalmazások reszponzív tervezése során:

- Képernyőméret és felbontás: Az alkalmazásnak alkalmazkodnia kell a különböző képernyőméretekhez és felbontásokhoz, beleértve az asztali számítógépek nagyobb képernyőit, a táblagépek közepes méretű képernyőit és a mobiltelefonok kisebb képernyőit is. Az alkalmazásnak úgy kell megjelenítenie az adatokat és az elemeket, hogy azok jól láthatóak és könnyen kezelhetőek legyenek az összes eszközön.

- Felhasználói élmény: Az alkalmazásnak kényelmes felhasználói élményt kell nyújtania az összes eszközön. A felhasználóknak könnyen hozzá kell férniük az alkalmazás funkcióihoz, és az alkalmazásnak intuitív és felhasználóbarát felületetet kell biztosítania az összes eszközön. A navigáció, a gombok és az interakciók is megfelelően működjenek minden képernyőmérethez és érintéses készülékhez.

- Tartalom optimalizálása: Az alkalmazás tartalmát és elemeit is optimalizálni kell a reszponzív tervezés során. A szövegeket és grafikonokat is úgy kell megjeleníteni, hogy azok jól láthatóak legyenek a különböző képernyőméretekhez és eszközbeállításokhoz. A tartalom megfelelő méretezése, pozícionálása és megjelenítése alapvető fontosságú a reszponzív alkalmazásokban.

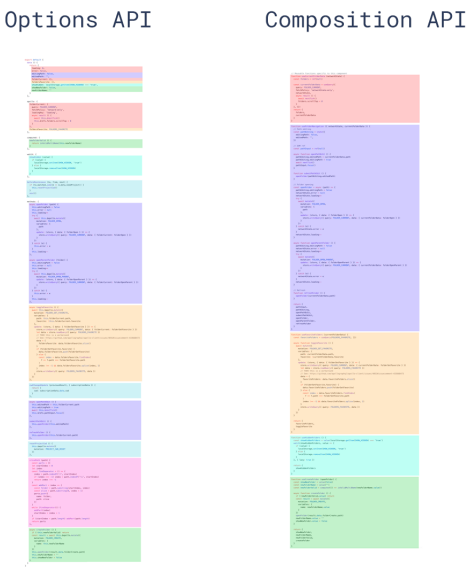
- Gyors betöltés: Az alkalmazásnak gyorsan kell betöltődnie az összes eszközön. A mobiltelefonok és a táblagépek használatakor különösen fontos a gyors betöltés, mivel ezekben az eszközökben korlátozott a sávszélesség és a kapcsolat stabilitása. Az alkalmazásnak fájlai minél kisebbek legyenek és.

3.5. Leválasztott logikák (composables, hooks, services)

A keretrendrendszerek alap építőelemei a komponensek. Ezek tartalmaznak templatet, ami a megjelenést írja le, valamint az alkalmazáslogikát leíró függvényeket és változókat. Bizonyos komponensek azonban nagyon bonyolulttá válhatnak. Vegyünk példának egy könyvtárszerkezetet megjelenítő komponenst [19], amely ezeket a funkciókat tartalmazza:

- Jelenlegi könyvtár listázása, és a változások követése
- Könyvtár navigáció kezelése (megnyitás, bezárás, frissítés)
- Új könyvtár létrehozása
- Kedvenc könyvtárak megjelölése, csak a kedvencek megjelenítése
- Rejtett könyvtárak elrejtése / megjelenítése
- Jelenlegi könyvtár módosításainak kezelése

Ezek logikailag jól elkülöníthető részek, azonban a kisebb komponsensekre bontás általában nem célszerű vagy körülményes lenne. Az összetartozó függvényeket érdemes lenne egymást követő sorokba írni, nem pedig külön csoportosítani a függvényeket és külön a változókat (vue options api, angular/react class szintakszis). Így könnyebben átlátható egy funkció, mivel nem szükséges a fájl különböző részeire görgetni.



3.4. ábra. Az összetartozó logikák azonos színnel

Valamint jó lenne ha az egyes részek a template megjelenítése nélkül használhatóak lennének más komponensekben. Például A könyvtár adatait visszaadó funkciót egy másik komponensben a könyvtárban található fájlok méretének kiszámításához szeretnénk használni.

Ezekre a problémákra nyújtanak megoldást a leválasztható logikák. Ez vue-nál a composition api-t reactnál a hook-okat, angularnál a service-eket jelenti.

3.5.1. Composition API

A Vue keretrendszerben a `composable` egy olyan függvény, ami a Composition API felhasználásával modularizált és újrafelhasználható stateful kódot tartalmaz. Ezeket konvenció szerint `use` kezdettel nevezzük el.

A Composition API három részből épül fel:

- Reactivity API: Például: `ref()`, `reactive()`, `computed()` és `watch()`, amely függvények segítségével direkt módon hozhatunk létre reaktív állapotot, computed állapotot és watchereket.
- Lifecycle Hooks: Például: `onMounted()`, `onUnmounted()`. A komponens különböző eseményeikor futtatandó függvényeket adhatunk meg.
- Dependency Injection: `provide()` `inject()`. A Vue dependency injection rendszere. Bármilyen mélységű gyerek komponensnek biztosít információátadást.

Az alábbi képeken egy példát hozok a saját programomból composable-re. Az első képen egy `useColumnsForHeadings` nevű composable definiálok. A feladata, hogy egy táblázat fejléceiből (string tömb) és egy dátumból meghatározza az adott fejléchez tartozó adatot. Például a `sunset` stringhez a naplemente időpontját megformázva kell visszaadnia. A függvény paraméterei reaktívak, tehát ha a használat helyén változik a dátum vagy a fejléceket tartalmazó tömb, akkor az eredmény is változni fog. Először a storeból lekérem a földrajzi helyzetet, mivel ezekhez is szükség van a számításokhoz. Ezután a computed függvény segítségével deklaráltam egy `ISODate` nevű változót, ami a dátumot ISO formában lévő stringként adja vissza. A második computed már az adott napra kiszámolt értékeket tartalmazza. Végül a columns computed a fejlécekhez társítja az adatokat. A composable végén objectként adom vissza az egyes változókat, így egyszerűen használható az, amire szükség van.

3.5.2. Hooks

React-ben a `Hook`-ok lehetővé teszik komponensen kívüli működés definiálását és használatát. A React tartalmaz több beépített Hook-ot, valamint sajátot is létrehozhatunk.

A Vue-nál már megismert működést leíró függvényeknek megfelelő beépített Hook-okat találunk React-nél.

A főbb beépített Hook-ok a következők:

- State(állapot) hooks: `useState()`: egy reaktív változót deklarál. `useReducer()`: szintén reaktív változót deklarál, viszont a reducer függvény tartalmazza az update logikát.
- Ref hooks: `useRef()`: egy NEM reaktív változót deklarál. Megváltozása esetén nem renderelődik újra a komponens. Olyan információ tárolására szolgál, amikre nincs szükség a rendereléshez. Pl.: egy DOM csomópont
- Context(kontextus) hooks: `createContext()`, `useContext()`: Bármilyen mélységű gyerek komponensnek biztosít információátadást.

```ts
export default function useColumnsForHeadings (headings: Ref<string[]>, date: Ref<DateTime>) {

    const settingsStore = useSettingsStore()
    const { lng, lat } = storeToRefs(settingsStore)


    const ISODate = computed(() => {
        return date.value.toISODate()
    })

    const timesResult = computed((): GetTimesResultLuxon => {
        return SuntimesUtility.transformGetTimesResultDatesToLuxon(
            SunCalc.getTimes(date.value.toJSDate(), lat.value, lng.value)
        )
    })

    const columns = computed(() => headings.value.map(x => {
        switch (x) {
            case "date":
                return ISODate.value
            case "sunrise":
                return formatTime(timesResult.value.sunrise)

            case "sunset":
                return formatTime(timesResult.value.sunset)

            case "dusk":
                return formatTime(timesResult.value.dusk)

            case "dawn":
                return formatTime(timesResult.value.dawn)


            default:
                return ""
        }

    }))



    return {
        columns,
        ISODate,
        timesResult
    }
}

```

3.5. ábra. Composable példa vue-ban

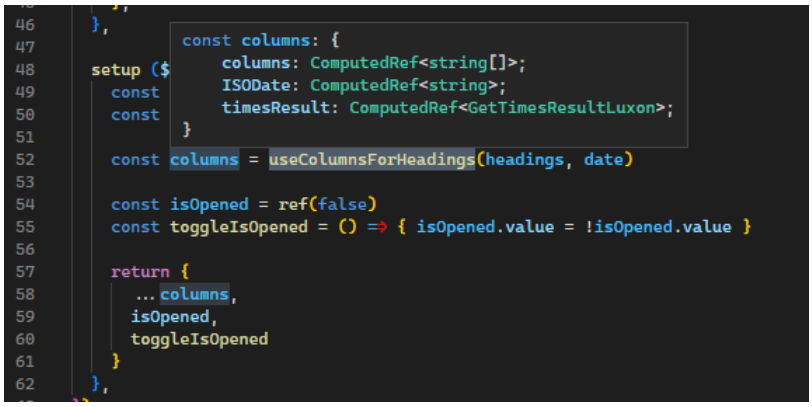
- Effect hooks: `useEffect()`: A paraméterben megadott változók megváltozása esetén lefuttat egy megadott callback függvényt.
- Performance(teljesítmény) hooks: `useMemo()`: Erőforrásigényes számítás eredményét menti, paraméterben megadható, hogy mely változók megváltozása esetén számoljon újra. `useCallback()`: függvénydefiníciót cache-el

3.5.3. Dependency Injection / Services

Az Angular a dependency injection(`DI`) módszerrel biztosítja a komponensektől független logikát. A DI az angular keretrendszer beépített része és lehetővé teszi az Angular decorator-ral rendelkező classok számára, hogy megadják mely függőségekre van szükségük. Ilyen classok lehetnek a: `Components`, `Directives`, `Pipes`, and `Injectables`.

Két fő szerep van a DI rendszerben: `dependency consumer` (függőség fogyasztó) és `dependency provider` (függőség biztosító).

Az Angular a két szereplő közötti interakciót egy absztrakciós réteggel teszi lehetővé, amit `Injector`-nak hívnak. Amikor egy függőséget kérelmeznek, az injector ellenőrzi a



3.6. ábra. Composable használata vue-ban

saját nyilvántartásában, hogy van-e már elérhető példány kért függőségből. Ha nincs, akkor létrehoz egy újat.

A legalapvetőbb példa függősegre egy osztály, ami bármilyen logikát vagy állapotot tartalmazhat. Ezeket az osztályokat szokás `Service`-nek is hívni. Osztályokon kívül függvények, object-ek, és primitív típusok is lehetnek függőségek.

Függőség biztosításához először a függőségnek szánt osztályt az `@Injectable()` dekorátorral kell megjelölnük. Ez után attól függően, hogy mely szinten akarjuk elérhetővé tenni a függőséget, jeleznünk kell ezt a DI rendszer számára.

- `Komponens szintű` függőség megadásához, egy adott komponens `providers` tömbjében kell az osztályt megadnunk. Ilyenkor a függőség elérhető lesz ezen komponens összes példányában és az összes olyan egyéb komponens és direktíva számára is, amit ezen kompones template-jében használunk. Ha komponens szinten adunk meg egy függőséget, akkor minden egyes komponens példányhoz egy új függőség példány jön létre.

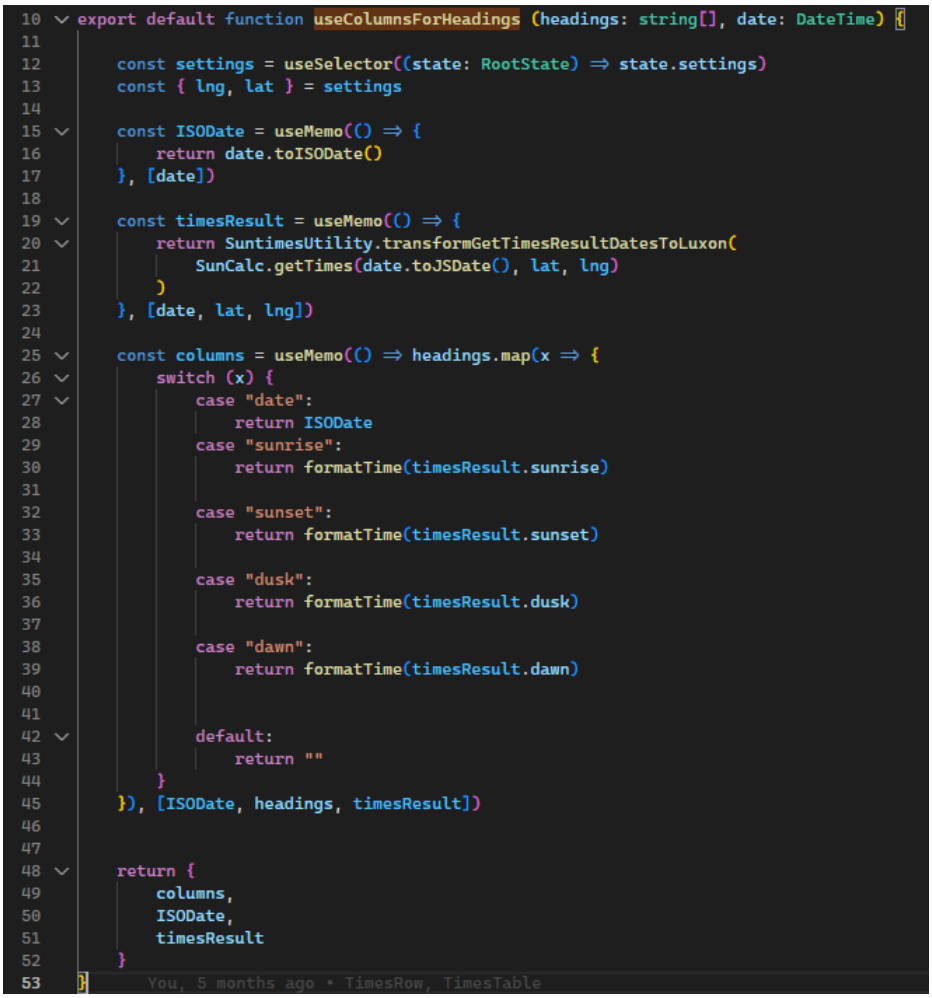
- `Modul szintű` függőség megadásához, egy adott modul `providers` tömbjében kell az osztályt megadnunk. Ebben az esetben a függőség elérhető lesz az ezen modulba tartozó összes komponens, direktíva és pipe számára. Ha modul szinten adunk meg egy függőséget, ugyanaz a példány lesz elérhető az összes komponens számára.

- `Root szintű` függőség megadásához, az `@Injectable()` dekorátorban a `providedIn: 'root'`, opció szükséges. Ilyenkor egy példány lesz a függőségből az egész alkalmazásban, és bármilyen class számára elérhető.

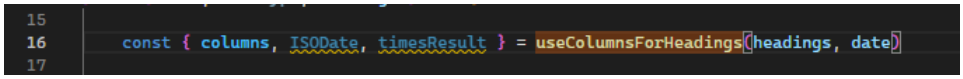
3.6. Store-ok összehasonlítása

A Store könyvtárak elsősorban az egész alkalmazás számára hozzáférhető, megosztott adat tárolására szolgálnak. Viszont ezt megvalósíthatnánk egy külső fájlban, mindenféle könyvtár nélkül is pl.: `export const state = reactive()` . Akkor mi szükség van a store-ra, mi az előnye?

A `pinia` store a vue Reactivity API-ját felhasználva biztosítja az állapotkövetést, azonban ezen felül jó Devtools támogatást nyújt (idővonalon láthatjuk a változásokat,



3.7. ábra. Hook példa react-ben



3.8. ábra. Hook használata react-ban

láthatjuk mely komponensek használnak adott store-okat, és "időutazhatunk" az állapotok között). Ezen kívül plugineket is használhatunk, amikkel megvalósítható például az undo-redo művelet, jó a TypeScript támogatottsága, valamint támogatja a SSR (server side rendering)-t. A többi store-nál is hasonló előnyöket találunk.

A Redux és az NgRx immutable módosítási logikát használva még kiszámíthatóbbá teszi az állapotváltozásokat.

Az NgRx a `SHARI` alapelvet ajánlja annak eldöntésére, hogy szükség van-e store-ra:

- **S**hared: az állapothoz sok komponensnek és service-nek hozzá kell férnie
- **H**ydrated: az állapot perzisztens és egy külső tárhelyről töltődik vissza (pl.: localstorage)
- **A**vailable: az állapotnak elérhetőnek kell lennie amikor újra belépünk egy adott route-ra
- **R**etrieved: az állapot olvasásához/megváltoztatásához side-effectek tartoznak

- **I**mpacted: az állapotra hatással vannak más forrásból számazó műveletek

A store-ok hátrányai viszont (főle NgRx/Redux pattern), hogy sok kódot igényelnek.

Általánosságban akkor van szükségünk store-ra, ha a SHARI elvből egy vagy több funkcióra szükségünk van.

Mutation

A mutation-ök a store állapotának megváltoztatására szolgálnak. Nem tartalmazhatnak aszinkron műveleteket. A pinia-ban nincsenek mutation-ök, mivel ezek nélkül is követni tudja a változásokat. Helyette action-öket használhatunk, ami viszont lehet aszinkron is.

Reducer

Az NgRx és a Redux használja a reducer-eket. Ezek olyan függvények, amik paraméterként megkapnak egy változtatástípust és visszaadják az új state-et. Erre azért van szükség, mivel a state immutable.

Async thunk

Reduxban aszinkron logikát async thunk-ban használhatunk. Azonban itt nem lehet módosítani az állapotot. Helyette egy speciális callback regisztrálásával automatikusan meghívhatunk egy mutation-t, amikor befejeződött az aszinkron futás.

Effects

Hasonlóan a redux-hoz, angularban sem tehetünk aszinkron logikát a mutation-ökbe, ezért ezeket egy effectben helyezhetjük el. Az effekt függvények egy megadott mutation futáskor lépnek működésbe (így side-effectek futtatására is jók, innen a név). Működésük befejeződését követően az eredményt átadhatják egy másik mutation-nek így megváltoztatva az állapotot.

3.7. Egyéb összehasonlítások

3.7.1. Composition API vs Hooks

A Vue Composition API hasonló kompozíciós képességekkel rendelkezik, mint a React Hooks, de van néhány fontos különbség. [5]

A React Hook-ok újra meghívódnak amikor a komponens frissül. Ez különböző veszélyeket hordoz magában, amik a tapasztaltabb React fejlesztőket is összezavarhatják. Valamint teljesítményoptimalizálási gondokat is okozhat, ami nagyban befolyásolja a fejlesztői élményt. Például:

- A Hook-ok hívási sorrend érzékenyek és nem lehetnek feltételesek.
- React komponensben megadott változókat elfoghatja egy hook, ami miatt "elavulttá" válnak (nem frissülnek, mert a komponens frissülése során egy új változó jön létre, más referenciával), ha a fejlesztő nem adja meg a helyes függőségeket

tartalmazó tömböt. Emiatt a React fejlesztők ESLint szabályokra hagyatkoznak, hogy biztosan a jó függőség tömb legyen átadva az egyes hook-oknak. Azonban a szabály gyakran nem elég okos és túlkompenzál a helyesség érdekében (az összes lehetséges függösőget felhasználja), ami felesleges újraszámolásokhoz vezet, és bosszankodáshoz, amikor bonyolultabb esetekkel találkozunk.

- Számításigényes feladatokhoz a `useMemo()` hookot kell használni, aminek szintén kézzel kell megadni a helyes függőség tömböt.

- Az olyan eseménykezelők, amik gyerek komponenseknek vannak átadva, feleslegesen a gyerek újrarenderelését okozzák, amikor a szülő frissül. Ahhoz, hogy ezt elkerüljük, a függvényt a `useCallback()` hook-al be kell csomagolni. Erre szinte mindig szükség van, és ennek a Hook-nak is meg kell adni a helyes függőség tömböt. Ha ezt elmulasztjuk, akkor túl sokszor fog újrarenderelni az alkalmazás, ami teljesítményproblémákat okoz.

- Az elavuló változó probléma, párhuzamos műveletekkel kombinálva, bonyolulttá teszi annak átlátását, hogy egy adott Hook kódja mikor fut. Emiatt a változtatható (mutable) állapottal dolgozás, aminek meg kell maradni az újrarenderelések után is, nehézkessé válik.

Ezzel szemben a Vue Composition API:

- Csak egyszer hívja a `setup()` vagy `<script setup>` kódot. Emiatt nem lehetnek "elavult" változók a kódban. Ezenkívül a Composition API hívások hívássorrendtől függetlenek és lehetnek feltételesek is.

- A Vue futásidejű reaktivitás rendszere automatikusan összegyűjti a reaktiv függőségeket, amik `computed`-hez és `watch`-hoz szükségesek, így nem kell ezeket kézileg megadni.

- Nem szükséges cache-elni az eseménykezelő függvényeket, mert ezek nem okozzák a gyerekkomponensek újrarenderelését. Általában véve a Vue finomhangolt reaktivitás rendszere biztosítja, hogy a gyerek komponensek csak akkor frissüljenek, amikor szükséges. A kézi gyerek frissülés optimalizálásokkal nagyon ritkán kell foglalkozniuk a Vue fejlesztőknek.

3.7.2. Működési mechanizmusok összehasonlítása

3.1. táblázat. Működési mechanizmusok összehasonlítása

	Vue	React	Angular
Állapot tárolása	ref(), reactive()	useState(), useReducer()	osztály adattagja
Számított érték	computed()	useMemo(), useCallback()	osztály getter, rxjs map(), Observable
Változó figyelése	watch()	useEffect()	rxjs subscribe()
Lifecycle hooks	onMounted(), onUnmounted()	useEffect()	ngOnInit(), ngOnDestroy()

```
You, last month | 1 author (You)
@Injectable({
    providedIn: "platform"
})

export default class ColumnsForHeadingsService {

    lng = new BehaviorSubject(0)
    lat = new BehaviorSubject(0)


    lng$
    lat$

    constructor(private store: Store<RootState>) {
        this.lng$ = this.store.pipe(select(selectLng))
        this.lat$ = this.store.pipe(select(selectLat))


        this.lng$.subscribe(newVal => {
            this.lng.next(newVal)
        })

        this.lat$.subscribe(newVal => {
            this.lat.next(newVal)
        })
    }

    setup (headings: BehaviorSubject<string[]>, date: BehaviorSubject<DateTime>) {
        const _ISODate = new BehaviorSubject(date.value.toISODate())      You, 2 months ago • TimesView
        const ISODate = date.subscribe((newVal) => {
            _ISODate.next(newVal.toISODate())
        })

        const computeTimesResult = ([date, lat, lng]: [DateTime, number, number]) => {
            return SuntimesUtility.transformGetTimesResultDatesToLuxon(
                SunCalc.getTimes(date.toJSDate(), lat, lng)
            )
        }
        const timesResult = combineLatest([date, this.lat, this.lng]).pipe(map(([date, lat, lng]) => {
            return computeTimesResult([date, lat, lng])
        }))

        const _timesResult = new BehaviorSubject<GetTimesResultLuxon>(computeTimesResult([date.value, this.lat.value, this.lng.value]))
        timesResult.subscribe((newVal) => {
            _timesResult.next(newVal)
        })

        const columns = combineLatest([headings, _ISODate, timesResult]).pipe(map(([headings, _ISODate, timesResult]) =>
            headings.map(x => {
                switch (x) {
                    case "date":
                        return _ISODate
                    case "sunrise":
                        return formatTime(timesResult.sunrise)

                    case "sunset":
                        return formatTime(timesResult.sunset)

                    case "dusk":
                        return formatTime(timesResult.dusk)

                    case "dawn":
                        return formatTime(timesResult.dawn)


                    default:
                        return ""
                }

            })
        ))
        const _columns = new BehaviorSubject<string[]>([])
        columns.subscribe(newVal => {
            _columns.next(newVal)
        })

        return {
            _ISODate,
            timesResult,
            _timesResult,
            columns,
            _columns
        }
    }

}
```

3.9. ábra. Service példa angular-ban

```
35    constructor(columnsForHeadingsService: ColumnsForHeadingsService) {
36      const { _columns } = columnsForHeadingsService.setup(this.headings$, this.date$)
37      this._columns = _columns
38    }
39
40    _columns: BehaviorSubject<string[]>
```

3.10. ábra. Service használata angular-ban

4. fejezet

Megvalósítás

4.1. Aloldalak

A program 3 különböző oldalból áll:

- Főoldal
- Listanézet
- Névjegy

A főoldalon láthatóak az aktuális nap adatai, valamint egy görbe, ami a nap pályáját mutatja. Itt állítható be a földrajzi hely és egyéb beállítások. A listanézet oldalon egy hónap adatai láthatóak napi bontásban, valamint minden naphoz tartozik egy görbe a nap pályájáról. A névjegy oldalon a program leírása található.

4.2. Nap adatainak meghatározása

A Nap adatainak meghatározásához alapvetően a földrajzi helyzetre és egy időpontra van szükség. A konkrét számításokat a SunCalc könyvtár végzi. A Nap adatai két fogalmat is jelenthet. Az első esetben egy naptári napra eső időpontokat értünk, ami olyan eseményeket jelöl, mint például a napfelkelte vagy a naplemente időpontja. Ezeket az adatok a `Suntimes.getTimes(date: Date, latitude: number, longitude: number): GetTimesResult` függvény adja meg.

A második esetben adott pillanatban a Nap pozícióját jelenti. Ez két értékből áll, egy altitude számból, ami a Nap horizonttal bezárt szögét jelenti fokban, valamint egy azimut számot, szintén fokban. Ezt a `Suntimes.getPosition(date: Date, lat: number, lng: number): GetSunPositionResult` függvény adja meg.

Az alábbiakban mindkét definícióra mutatok konkrét példát. A kódok a vue alkalmazásból származnak, de a többiben is nagyon hasonlóak.

Az első esetben a földrajzi helyzetet a store-ból kérem le, aztán a Suncalc.getTimes függvény eredményét egy saját transzformáló függvénnyel luxon-os DateTime-okká alakítom. Ez lesz a végleges eredmény.

```
1       const settingsStore = useSettingsStore()
2       const { lng, lat } = storeToRefs(settingsStore)
3
4       const timesResult = computed((): GetTimesResultLuxon => {
```

```
5       SuntimesUtility.transformGetTimesResultDatesToLuxon(
6       SunCalc.getTimes(date.value.toJSDate(),
7       lat.value,
8       lng.value)
9       )})
```

A második esetben a naptári nap minden egyes másodpercéhez meghatározza a `getSunPathForDay` függvény a Nap pozícióját.

```
1       export function getSunPathForDay
2       (day: DateTime, lat: number, lng: number, n = 1440) {
3       const start = day.startOf("day").toJSDate(),
4       end = day.plus({ days: 1 }).startOf("day").toJSDate()
5
6       const startMs = start.getTime()
7
8       const diff = Math.floor((end.getTime() - start.getTime()) / n)
9       const times = Array.from({ length: n }, (_v, k) => {
10
11      const _position = SunCalc.getPosition(new Date(startMs + k *
            diff),
12      lat,
13      lng)
14
15      return {
16              time: new Date(startMs + k * diff),
17              altitude: radians_to_degrees(_position.altitude),
18              azimuth: radians_to_degrees(_position.azimuth) + 180,
19          }
20      })
21
22      return times
23      }
```

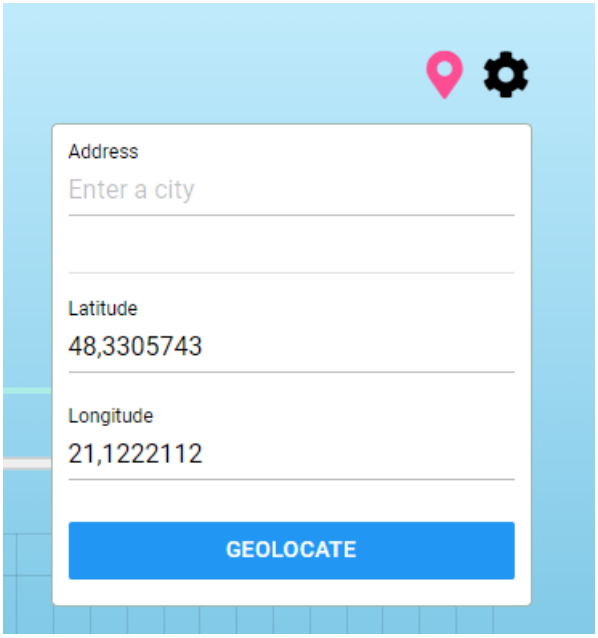
4.3. Helymeghatározás

A földrajzi helyzet megadására többféle megadási módot kínál az alkalmazás. Amellett, hogy manuálisan beírhatók a koordináták, automatikus helymeghatározásra is van mód, valamint városnév is megadható.

A helymeghatározáshoz a `navigator.geolocation` API-t használtam. Ahhoz, hogy ez működjön a felhasználónak engedélyt kell adnia a helyadatai eléréséhez, a gomb megnyomását követően.

```
1
2
3       // Geolocate
4       const geolocate = () => {
5       navigator.geolocation.getCurrentPosition((position) => {
6       lat.value = position.coords.latitude
7       lng.value = position.coords.longitude
8       })
9       }
```

A városnév alapú helymeghatározáshoz az OpenStreetMap alapú ingyenes Nominatim API-t használtam. A használatához csak a keresendő szöveget kell átadni és



4.1. ábra. Helybeállítások

ez alapján adja vissza a koordinátákat. Nemcsak városnévvel, de bármilyen térképen megtalálható objektummal működik.

```
1
2       const assembleGeocodeURLFromAddress = (address: string) => {
3
4       return encodeURI(`https://nominatim.openstreetmap.org/search
5       ?format=json&q=\${address}`)
6       }
7
8       const geocode = async (address: string):
           Promise<NominatimPlace> => {
9                try {
10                       const res = await axios.get<NominatimPlace[]>(
11                       assembleGeocodeURLFromAddress(address))
12
13                       if (!res.data.length)
14                       throw new Error("Not found")
15
16                       const place = res.data[0]
17                       return place
18
19                   } catch (err) {
20                       console.error("geocode failed :(")
21                       throw new Error("Error")
22                   }
23          }
```

4.4. Dinamikus háttérszín

A kezdőlapon a háttérszín napszaknak megfelelően változik. Nappal világoskék színek láthatóak, éjszaka pedig fekete. Reggel és este a valódi égre emlékeztető vöröses színek

láthatóak átmenetesen. 25 előre definiált linear-gradient alapú szín lehet a háttér, azt hogy mikor melyik, a napszak dönti el. Az átmenethez először meghatározom az aktuális és a következő háttérszínt. Mindkettőt kirajzoltatom, majd a pontos átmenetet a következő háttér átlátszatlansági értéke adja meg.

```
1       export class SkyEffect {
2
3               static skyGradients = [
4                       "linear-gradient(to bottom,
5                       #00000c 0%, #00000c 100%)",
6                       ...
7                   ]
8
9               altitude
10              direction
11
12
13              getBrightness () {
14                      const maxBrightnessAltitude = 20
15                      const minBrightnessAltitude = -18
16                      const maxBrightnessValue = Math.floor(
17                      SkyEffect.skyGradients.length / 2)
18                      const minBrightnessValue = 0
19
20                      if (this.altitude > maxBrightnessAltitude)
21                      return maxBrightnessValue
22
23                      if (this.altitude < minBrightnessAltitude)
24                      return minBrightnessValue
25
26                      return ((maxBrightnessValue) *
27                      (this.altitude + minBrightnessAltitude * -1))
                            /
28                      (maxBrightnessAltitude - minBrightnessAltitude
                            )
29
30                  }
31
32              getLinearGradient ()
33              // direction: true: goes up; false: goes down
34                  {
35                      const absoluteBrightness = this.getBrightness
                            ()
36                      const currentIndex = this.direction ?
37                      Math.floor(absoluteBrightness) :
38                      Math.floor((SkyEffect.skyGradients.length - 1)
                            -
39                      absoluteBrightness)
40                      const nextIndex = Math.min(currentIndex + 1,
41                      SkyEffect.skyGradients.length - 1)
42                      return {
43                              current: SkyEffect.
44                              skyGradients[currentIndex],
45                              next: SkyEffect.skyGradients[nextIndex
                                    ],
46                              nextOpacity: absoluteBrightness % 1
                                    === 0 ?
47                              0 :
```

```
48                              (this.direction ?
49                              absoluteBrightness % 1 :
50                              1 - (absoluteBrightness % 1)),
51                          }
52                  }
53
54
55              constructor({ altitude = 0, direction = true }) {
56                      this.altitude = altitude
57                      this.direction = direction
58                  }
59          }
```

A SkyEffect osztályt felhasználva, a Suntimes komponensben meghatározható a háttérszín. Az alábbi kód ezt mutatja be. A kapott háttér és előtér színértékek ezután egy style object-ba kerülnek, ami a wrapper div elemre van alkalmazva. Így ezek a színek css változókon keresztül bárhol kiolvashatóak és felhasználhatóak.

```
1       watchEffect(() => {
2       if (!useSkyEffect.value)
3       return
4
5       skyEffect.altitude = Number(sunPosition.value.altitude)
6       skyEffect.direction = Boolean(percentage.value < 50)
7       })
8
9       const backgroundColor = computed(() => {
10      if (!useSkyEffect.value)
11      return { current: "#ffffff", next: "#ffffff", nextOpacity: 0 }
12
13      return skyEffect.getLinearGradient()
14      })
15
16      const foregroundColor = computed(() => {
17      if (!useSkyEffect.value)
18      return "black"
19
20      return sunPosition.value.altitude > 10 ? "black" : "white"
21      })
22
23      const styleForWrapper = computed(() => {
24      return {
25              "--background-sun-current":
                    styles.backgroundSunCurrent,
26              "--background-sun-next": styles.backgroundSunNext,
27              "--foreground-sun": styles.foregroundSun,
28              "--background-sun-primary":
                    styles.backgroundSunPrimary,
29              "--opacity-sun-next": styles.opacitySunNext,
30          }
31      })
32
33      watch(backgroundColor,
34      (newVal) => {
35      styles.backgroundSunCurrent = newVal.current
36      styles.backgroundSunNext = newVal.next
37      styles.backgroundSunPrimary =
38      !useSkyEffect.value ?
```

```
    "white" :
    sunPosition.value.altitude > 10 ? "white" : "black"
    styles.opacitySunNext = newVal.nextOpacity
    }, { immediate: true })

    watch(foregroundColor,
    (newVal) => {
    styles.foregroundSun = newVal
    },
    { immediate: true }
    )
```

```
    <div class="suntimes__wrapper" :style="styleForWrapper">
    <div class="background background__current"></div>
    <div class="background background__next"></div>
    ...
    </div>
```

```

    .background {
    position: fixed;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    z-index: -1;

    &__current {
    opacity: 1;
    background: var(--background-sun-current);
    z-index: -2;
    }

    &__next {
    z-index: -1;
    opacity: var(--opacity-sun-next);
    background: var(--background-sun-next);
    }
    }
```

4.5. Listanézet

A listanézet célja, hogy egy adott hónap adatait megjelenítse. A nézet két fő komponenst tartalmaz: TimesParamaters és TimesTable. A TimesParameters-ben választható ki a hónap és az, hogy a Nap vagy (majd a jövőbeli fejlesztések után) a Hold adatai legyenek megjelenítve. A TimesTable tartalmazza a listát, amit a beállított paraméterek alapján jelenít meg. A lista sorait a TimesRow kopmonens alkotja. Ez a komponens megjeleníti a paraméterként kapott oszlopok értékeit. Ezenkívül egy gombot is tartalmaz, amely kattintásra lenyitja a sort és egy grafikont jelenít meg, ami a Nap pályáját ábrázolja.

```
    <template>
    <div class="view__times">
      <h1>Times - {{ monthName }}</h1>
      <div class="view__times__times-table">
```

```
5         <TimesParameters v-model:from="from" v-model:viewType="
              viewType"></TimesParameters>
6         <TimesTable :from="from" :to="to" :viewType="viewType"></
              TimesTable>
7       </div>
8     </div>
9   </template>
10
11  <script lang="ts">
12  // imports
13  export default defineComponent({
14    name: "TimesView",
15    components: {
16      TimesParameters,
17      TimesTable,
18    },
19    setup () {
20      const from = ref(DateTime.now().startOf("month"))
21      const viewType: Ref<SuntimesViewType> = ref(
            SuntimesViewType.SUN)
22      const to = computed(() => {
23        return from.value.endOf("month")
24      })
25      const monthName = computed(() => {
26        return from.value.toFormat("LLLL")
27      })
28      return {
29        from,
30        viewType,
31        to,
32        monthName
33      }
34    }
35  })
36  </script>
```

```
1     <template>
2     <tr class="times-row">
3       <td v-for="(column, index) of columns" :key="index">{{
            column }}</td>
4       <td class="buttons-col">
5         <button class="mui-btn mui-btn--small mui-btn--primary"
              @click="toggleIsOpened" title="Expand / collapse">
6           <i class="fas fa-caret-down"></i>
7         </button>
8       </td>
9     </tr>
10    <tr v-show="isOpened">
11      <td colspan="6">
12        <KeepAlive>
13          <SunGraph v-if="isOpened" :date="date" :animate="false"
                /></KeepAlive>
14        </td>
15      </tr>
16    </template>
17
18  <script lang="ts">
19
```

```
20    import { DateTime } from "luxon"
21    import useColumnsForHeadings from "@/composables/
          useColumnsForHeadings"
22    import { toRef, defineComponent, ref, PropType } from "vue"
23    import { SuntimesViewType } from "@/interfaces/Suntimes"
24    import SunGraph from "./SunGraph.vue"
25
26    export default defineComponent({
27      name: "TimesRow",
28      components: {
29        SunGraph
30      },
31      props: {
32        date: {
33          type: Object as PropType<DateTime>,
34          required: true,
35        },
36
37        viewType: {
38          type: String as PropType<SuntimesViewType>,
39        },
40
41        headings: {
42          type: Array as () => Array<string>,
43          default: (): string[] => [],
44          required: true,
45        },
46      },
47
48      setup ($props) {
49        const headings = toRef($props, "headings")
50        const date = toRef($props, "date")
51
52        const columns = useColumnsForHeadings(headings, date)
53
54        const isOpened = ref(false)
55        const toggleIsOpened = () => { isOpened.value = !
              isOpened.value }
56
57        return {
58          ...columns,
59          isOpened,
60          toggleIsOpened
61        }
62      },
63    })
64    </script>
```

4.6. Treeshaking és Code splitting

4.6.1. Treeshaking

Az egyik legjobb mód az oldal betöltési teljesítményének javítására, ha kisebb méretű a letöltendő JavaScript fájl. Ennek érdekében célszerű egy csomagoló programot használni. Az én dolgozatomban a webpack-et használtam mindhárom programhoz. A

webpack képes arra, hogy egy adott csomagból importáláskor, csak a szükséges, importált függvényeket / változókat csomagolja bele a kész JavaScript fájlba. Ezt nevezzük Treeshaking-nek. Mindhárom keretrendszer tartalmaz olyan API-kat, amik elősegítik a kisebb csomagméretet.

```
import { computed, defineComponent, ref, toRefs } from "vue"
import { Component } from '@angular/core'
import { useMemo, useState } from "react"
```

4.6.2. Code splitting

A Code splitting szintén egy teljesítménynöveli megoldás. A csomagoló program több kisebb méretű fájlt állít elő, amik szükség esetén vagy akár párhuzamosan tölthetőek be. Jól kivitelezett code-splitting-el azok a funkciók, amikre éppen szükség van, az oldal betöltésekor letöltődnek, a többi viszont csak szükség esetén. Az én programomban a különböző route-hoz tartozó kódot kerülnek külön fájlba és oldalnavigációkor töltődik le a szükséges fájl. Ahhoz, hogy a webpack külön fájlba tegyen egy importált modult, az import hívást egy függvényen belül kell elhelyezni, ami szükség esetén hívódik meg. Ez a függvény meghívható kézzel, de az én programomnál a routerek ezt intézik.

```
() => import('../views/HomeView')
```

Ráadásként készítettem betöltésjelző és hibajelző komponenst is. A betöltésjelző komponens az új fájl letöltése közben jelenik meg, míg a hibajelző akkor, ha valamilyen hiba miatt nem sikerül betölteni a fájlt. Ehhez az egyes keretrendszerekben eltérő kód szükséges, ezért ezeket külön részletezem.

Vue

Vue-nál megadható route komponensként egy olyan függvény, ami AsyncComponent-tel tér vissza. A defineAsyncComponent függvénynek adható meg paraméterként a saját loader és error komponens, valamint egyéb beállítások.

```
function lazyLoadView (AsyncView: Promise<Component>) {
        const AsyncComp = defineAsyncComponent({
                loader: () => AsyncView,
                loadingComponent: Loading,
                errorComponent: Error,
                delay: 5,
                timeout: 30000,
                suspensible: false,
                onError (error, retry, fail, attempts) {
                        if (error.message.match(/fetch/) &&
                        attempts <= 3) {
                                retry()
                            } else {
                                fail()
                            }
                    },
            })

        return AsyncComp
    }
```

```
    {
        path: '/',
        name: 'home',
        //Route komponens definialasa
        component: lazyLoadView(import('../views/HomeView.vue'))
    },
```

React

React-nél a beépített lazy függvény jelzi, hogy a komponenst szükség esetén kell betölteni. A hibakomponenst minden egyes route rekordhoz meg kell adni, ezért, hogy ezt megkönnyítsem, egy saját függvénnyel állítom elő a rekordokat, ami mellékeli a hibakomponenst.

```
    import { lazy } from "react"

    const HomeView = lazy(() => import('../views/HomeView'))

    const createRouteRecord = (record: IRouteObject): IRouteObject
        => {
    return {
            errorElement: <Error />,
            ...record
        }
    }
```

```
    //Route rekord definialasa
    createRouteRecord({
            path: "",
            name: 'home',
            element: <HomeView />,
        }),
```

A betöltés komponens, a router nézet köré helyezett Suspense komponensnek adható meg.

```
    <Suspense fallback={<Loading />}>
    <Outlet context={{ setRouteClass }} />
    </Suspense>
```

Angular

Angularnál a loadChildren kulcscsal adható át egy függvény ami betölti a route komponenst. A betöltés és hiba komponens megjelenítésének szabályozásához létrehoztam két változót a főkomponensben, majd a router eseményeinek kezelésével kapnak értéket. Végül a visszajelző komponenseket elhelyeztem a template-ben a router nézet elé, feltételes megjelenítéssel.

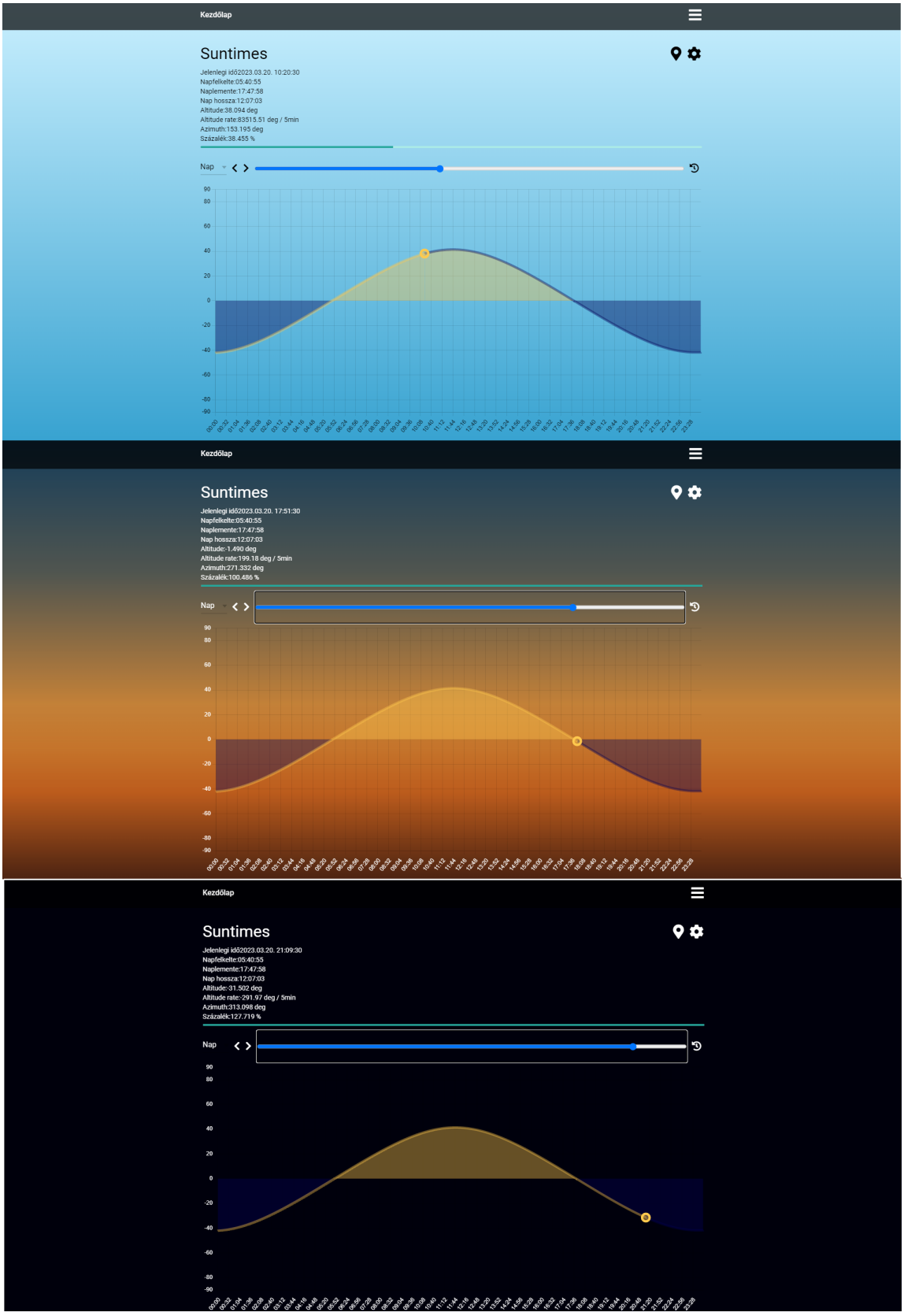
```
    //Route rekord definialasa
        { path: 'home', title: 'home',
            loadChildren:
            () => import("./views/home-view/home-view.module")
            .then(m => m.HomeViewModule) },
```

```
    this.router.events.subscribe((routerEvent: Event) => {
    if (routerEvent instanceof NavigationStart) {
            this.showLoadingComponent = true
            this.showErrorComponent = false
        }

    if (routerEvent instanceof NavigationError) {
            this.showErrorComponent = true
        }

    if (routerEvent instanceof NavigationEnd ||
    routerEvent instanceof NavigationCancel ||
    routerEvent instanceof NavigationError) {
            this.showLoadingComponent = false
        }
    })
```

```
    <div class="container">
    <loading-component *ngIf="showLoadingComponent"></
       loading-component>
    <error-component *ngIf="showErrorComponent"></error-component>
    <router-outlet></router-outlet>
    </div>
```



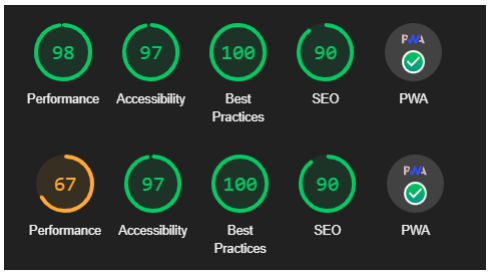
4.2. ábra. Dinamikus háttér

5. fejezet

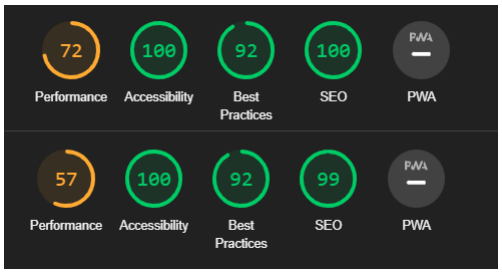
Tesztelés

5.1. Lighthouse teszt

A Lighthouse teszteknek 4 összetevője van: Teljesítmény, Accessibility, Best practices és SEO. A teljesítmény az oldal betöltésének a sebességét méri. A teszt futtatható desktop és mobil módban is, mobil módban egy gyengébb eszköz teljesítményét szimulálja. A képeken a felső sorban a desktopos futás, az alsóban a mobilos látható. A teszteket a listanézet oldalon futtattam, mivel a kezdőlapon a grafikon kirajzolásra nagyban befolyásolja az eredményt.



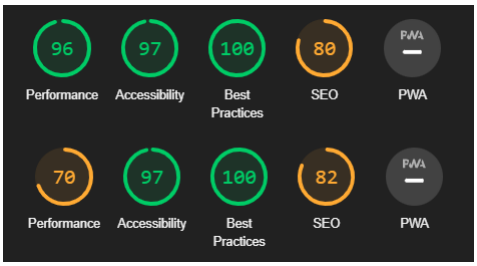
5.1. ábra. Vue Lighthouse teszt



5.2. ábra. React Lighthouse teszt

5.2. Teljesítmény tesztek

Az alábbi táblázatban a keretrendszerek teljesítmény tesztje [3] látható. Egy keretrendszer kétféleképpen végezhet módosítást egy listán: vagy megtartja a hozzárendelést a listaelem és a DOM Node között (kulcsozott), vagy szabadon átrendezi és hozzárendeli



5.3. ábra. Angular Lighthouse teszt

a DOM Node-okat (nem kulcsozott). A kulcs használatával hatékonyan frissül a lista, amikor elemeket adunk hozzá, törölünk vagy átrendezünk, ezért szinte mindig ilyenre van szükség. A tejlesítmény teszt első része kulcsozott listákon végzett adatmódosításból áll. Ezek gyakori műveletek a legtöbb alkalmazásban.

Amint látható, a legtöbb műveletnél a Vue és az Angular gyorsabb volt mint a React. A select row tesztben a React-nek kétszer annyi ideig tartott a művelet. A swap-rows tesztben a Vue jelentősen gyorsabb idő alatt végzett, 29.3 ms alatt, szemben az Angular 170.5 ms és a React 168.6 ms idejével.

A második táblázat a keretrendszerek indulási idejével foglalkozik. Itt első helyen a Vue, másodikon a React, harmadikon az Angular végzett.

A harmadik táblázatban a memóraifoglalás látható. A sorrend itt is ugyanaz mint az indulási időknél.

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	vue-v3.2.47	angular-v15.0.1	react-v18.2.0
Implementation notes			
Implementation link	code	code	code
create rows creating 1,000 rows (5 warmup runs).	45.4 ±0.7 (1.22)	45.7 ±0.4 (1.23)	52.2 ±0.9 (1.41)
replace all rows updating all 1,000 rows (5 warmup runs).	45.6 ±0.6 (1.13)	50.8 ±0.7 (1.26)	54.6 ±1.0 (1.35)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	120.3 ±3.8 (1.24)	112.3 ±2.4 (1.15)	145.6 ±3.8 (1.50)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	19.9 ±1.0 (1.74)	19.5 ±4.6 (1.70)	44.4 ±1.5 (3.89)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	29.3 ±0.3 (1.00)	170.5 ±1.0 (6.28)	168.6 ±1.4 (6.21)
remove row removing one row. (5 warmup runs). 4x CPU slowdown.	51.6 ±0.8 (1.14)	45.4 ±0.9 (1.00)	53.5 ±1.1 (1.18)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	494.5 ±4.3 (1.19)	504.1 ±3.2 (1.21)	682.9 ±3.5 (1.64)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	98.9 ±0.8 (1.14)	106.3 ±0.6 (1.23)	117.7 ±0.6 (1.36)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	37.1 ±1.4 (1.28)	71.4 ±3.5 (2.46)	40.3 ±1.0 (1.39)
geometric mean of all factors in the table	1.23	1.60	1.85
compare: Green means significantly faster, red significantly slower	compare	compare	compare

Startup metrics (lighthouse with mobile simulation)

Name	vue-v3.2.47	angular-v15.0.1	react-v18.2.0
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,027.1 ± 49.2 (1.12)	2,837.9 ± 52.5 (1.57)	2,552.8 ± 50.0 (1.42)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	197.0 ±0.0 (1.38)	282.8 ±0.0 (1.99)	280.9 ±0.0 (1.97)
geometric mean of all factors in the table	1.25	1.77	1.67

Memory allocation in MBs ± 95% confidence interval

Name	vue-v3.2.47	angular-v15.0.1	react-v18.2.0
ready memory Memory usage after page load.	0.9 (1.38)	1.6 (2.47)	1.1 (1.73)
run memory Memory usage after adding 1,000 rows.	3.8 (2.09)	4.7 (2.62)	5.0 (2.76)
update every 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	3.8 (1.94)	4.8 (2.45)	5.5 (2.80)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	1.2 (1.75)	2.3 (3.29)	1.9 (2.75)
run memory 10k Memory usage after adding 10,000 rows.	27.7 (2.38)	29.8 (2.55)	36.2 (3.10)
geometric mean of all factors in the table	1.88	2.66	2.58

5.4. ábra. Benchmark

5.3. Felhasználói útmutató

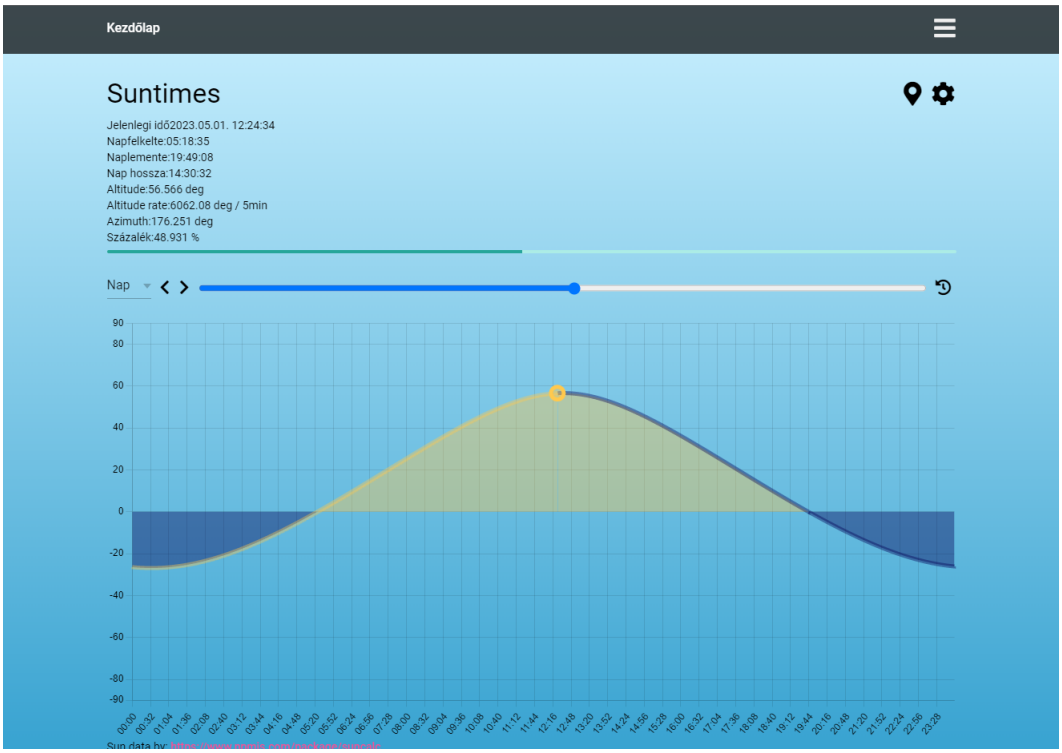
5.3.1. Kezdőlap

A Kezdőlap az oldal fő felhasználói felülete. Alapértelmezetten az aktuális időpontra vonatkozó adatok láthatóak. Ezek sorrendben a következők: jelenlegi idő, napfelkelte, naplemente, nap hossza, magasság (horizonthoz képest, szögben), azimut, százalék (napfelkelte és naplemente közötti eltelt idő százalékban). A százalék folyamatsávon is megjelenik.

A jobb felső sarokban található két gomb, amelyek beállítás modalokat nyitnak meg. A baloldali gomb nyitja a földrajzi helyzetet beállító modalt. Itt településnév megadással vagy földrajzi koordinátákkal állítható be a kívánt hely. A jobboldali gomb általános beállításokat tartalmaz, itt kapcsolható ki a dinamikus háttérszín.

A szöveges adatok alatt található az időpontválasztó felület. Itt kiválaszthatjuk, hogy a beállítócsúszka milyen terjedelmet fedjen le. Választható opciók: Év, Nap, Óra. A nyilak segítségével a kiválasztott terjedelmnyi időt ugorhatunk előre vagy vissza, a csúszka segítségével pedig ezen a terjedelmen belül választhatjuk ki a pontos időpontot. A jobb oldalon található gomb visszavált a jelenlegi időre.

Legalul található egy görbe, ami a kiválasztott naphoz rajzolja ki a Nap pályáját. Az x tengely az időt ábrázolja, az y pedig a magasságot fokban. Amikor a Nap a horizont alatt tartózkodik, a görbe feletti terület kék színű, ellenkező esetben a görbe alatti terület sárga színű lesz.

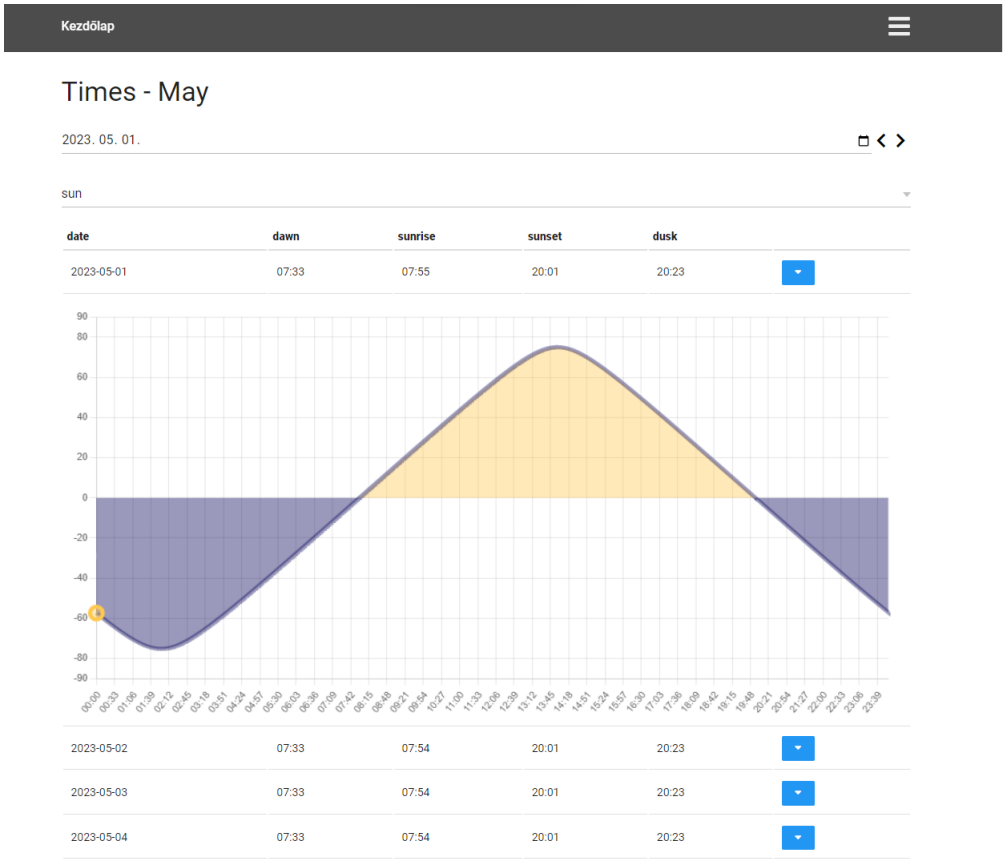


5.5. ábra. Kezdőlap

5.3.2. Listanézet

A listanézet tetején kiválaszthatunk egy dátumot. Csak a hónapok első napja választható és a kiválasztott hónap napjai fognak megjelenni.

A listában egy elem egy napot jelöl. Az adott naphoz a következő adatok jelennek meg: dátum, hajnal időpontja (civil twilight), napfelkelte időpontja, naplemente időpontja, alkonyat időpontja (civil twilight). A táblázat sorai lenyithatóak. Lenyitás után egy grafikont kapunk, ami ugyanolyan, mint a kezdőlapon található.



5.6. ábra. Listanézet

6. fejezet

Összefoglalás

A szakdolgozatom során a jelenleg három legnépszerűbb frontend keretrendszert hasonlítottam össze. Ehhez segítségül szolgált az elkészített alkalmazás, ami a Nap adatait és pályáját ábrázolja.

Odafigyeltem, hogy a felhasználói élmény intuitív és a megjelenítés esztétikus legyen. Például a háttérszín egyedi megoldás, ami megragadja a felhasználó figyelmét. A kódot legjobb tudásom szerint készítettem el, figyelembe véve, hogy a jövőben tovább szeretném fejleszteni az alkalmazást és akár publikálni is azt.

Mindhárom keretrendszernek megvannak a maga előnyei és hátrányai. Bármelyik használata jó döntés lehet, ez inkább az adott projektől és egyéni preferenciáktól függ.

A Vue kényelmes és könnyen használható, könnyen tanulható, ennek ellenére haladóbb, bonyolult feladatok is megvalósíthatóak vele. Hátránya, hogy kevesebben használják, főleg azért mivel később jelent meg, mint a másik kettő. Ebből következik, hogy kisebb a közösségi támogatottsága, mint például a harmadik féltől származó csomagok és fórumbejegyzések.

A React alapjait könnyű elsajátítani, viszont meredek a tanulási görbéje, emiatt nehéz tényleg jól ismerni. Ez abból adódik, hogy a másik két keretrendszerrel ellentétben, manuálisan kell optimalizáni a gyors futás érdekében. A React által nyújtott szabadság miatt a kód minősége nagyon változó lehet. Ez reflektálódik az internetes fórumokon megtalálható kódokon is. Ugyanakkor nagyon sok jól megírt és támogatott csomag van hozzá, amik megkönnyítik a fejlesztést. Ennek a legnagyobb a felhasználói bázisa és a fejlesztője a Facebook.

Az Angular egy előre megadott, jól átlátható projektstruktúrával segíti a fejlesztőket. Megkönnyíti a kód kisebb egységekre felosztását a modulrendszer, így önállóan működő, tesztelhető részeket kapunk. Az Angular fontos része az RxJS, ami kezdők számára nehezen érthető lehet, viszont ha megértjük, akkor egy hasznos eszközt kapunk, ami megkönnyíti az eseménykezelést és az azzal kapcsolatos feladatokat. A keretrendszer egyik legnagyobb előnye az, hogy viszonylag kötött felépítéséből adódóan a kódolás meghatározott keretek között zajlik és emiatt kevesebbet enged hibázni.

Úgy érzem, hogy a szakdolgozat elkészítése közben rengeteg hasznos dolgot tanultam és új ismeretekre tettem szert.

7. fejezet

Summary

During my thesis, I compared the three currently most popular frontend frameworks. The application I made, which depicts the data and the path of the Sun, helped with this.

I made sure that the user experience was intuitive and the visuals were aesthetically pleasing. For example, the background color is a unique solution that grabs the user's attention. I implemented the code to the best of my ability, taking into account that in the future I would like to further develop the application and even publish it.

All three frameworks have their advantages and disadvantages. Using either one can be a good decision, it depends more on the project and individual preferences.

Vue is convenient and easy to use, easy to learn, yet more advanced and complex tasks can be implemented with it. The disadvantage is that fewer people use it, mainly because it was released later than the other two. As a result, it has less community support, such as third-party packages and forum posts.

The basics of React are easy to learn, but it has a steep learning curve, which makes it difficult to really get to know it well. This is due to the fact that, unlike the other two frameworks, it must be manually optimized in order to run quickly. Because of the freedom React provides, code quality can vary greatly. This is also reflected in the codes found on Internet forums. At the same time, there are many well-written and supported packages that make development easier. It has the largest user base and it's developed by Facebook.

Angular helps developers with a predefined, transparent project structure. The module system makes it easier to divide the code into smaller units, so we get independently functioning, testable parts. An important part of Angular is RxJS, which can be difficult for beginners to understand, but once you understand it, you get a useful tool that facilitates event management and related tasks. One of the biggest advantages of the framework is that, due to its relatively fixed structure, the coding takes place within a specific framework and therefore allows for fewer mistakes.

I feel that I learned a lot of useful things and gained new knowledge while writing the thesis.

Hivatkozások

Az internetes források utolsó ellenőrzése: 2023.05.17

[1] Angular documentation. https://angular.io/docs.

[2] Angular router documentation. https://angular.io/guide/routing-overview.

[3] Benchmark. https://krausest.github.io/js-framework-benchmark/current.html.

[4] Chart.js documentation. https://www.chartjs.org/docs/latest/.

[5] Comparision with react hooks. https://vuejs.org/guide/extras/composition-api-faq.html#comparison-with-react-hooks.

[6] Github angular. https://github.com/angular/angular.

[7] Github react. https://github.com/facebook/react.

[8] Github vue 2. https://github.com/vuejs/vue.

[9] Github vue 3. https://github.com/vuejs/core.

[10] Ngrx documentation. https://v7.ngrx.io/docs.

[11] Pinia documentation. https://pinia.vuejs.org/introduction.html.

[12] React documentation. https://reactjs.org/docs/getting-started.html.

[13] React router documentation. https://reactrouter.com/en/main.

[14] Redux documentation. https://redux.js.org/.

[15] Rxjs documentation. https://rxjs.dev/guide/overview.

[16] Stackoverflow. https://survey.stackoverflow.co/2022/#web-frameworks-and-technologies.

[17] Suncalc github. https://github.com/mourner/suncalc.

[18] Vue cli documentation. https://cli.vuejs.org/.

[19] Vue composition api vue-cli gui examle. https://vuejs.org/guide/extras/composition-api-faq.html#more-flexible-code-organization.

[20] Vue documentation. https://vuejs.org/.

[21] Vue router documentation. https://router.vuejs.org/guide/.

CD Használati útmutató

A CD gyökérkönyvtárában különböző könyvtárakban találhatóak az egyes alkalmazások valamint a dolgozat.

- dolgozat: Dolgozat latex forráskódja és a pdf
- suntimes-vue: Vue-val elkészített alkalmazás
- suntimes-angular: Angular-al elkészített alkalmazás
- suntimes-react: React-el elkészített alkalmazás

Az egyes alkalmazások könyvtárában megtalálható a lefordított, böngészőben futtatható kód. Ez tetszőleges webkiszolgálóval működésre bírható. Az összes url-nek az `index.html`-re kell mutatnia, mivel a routing frontenden van megoldva. Egy egyszerű webkiszolgáló a `serve`, amihez elhelyeztem egy helyes konfigurációt tartalmazó fájlt (serve.json) a megfelelő könyvtárban.

Telepítése:

```
1 npm install --global serve
```

Ezután a lefordított fájlokat tartalmazó könyvtárba belépve, a következő paranccsal indítható:

```
1 npx serve
```

A lefordított fájlok a következő könyvtárakban találhatóak:

- Vue: dist/
- React: build/
- Angular: dist/suntimes-angular/

A kód fordítására is lehetőség van. Ehhez NodeJS-re lesz szükség, valamint az npm csomagkezelőre. A fordításhoz érdemes a CD-ről írható könyvtárba helyezni a fájlokat, mivel a függőségek letöltése közben lemezre írás történik. A NodeJS letölthető innen: https://nodejs.org/en

A fordításhoz be kell lépni a fordítandó program könyvtárába, majd először az

```
1 npm install
```

futtatásával feltelepíteni a függőségeket. Ezután fordítható a kód. A package.json fájlokban található az összes futtatási mód. Itt felsorolom, hogy fejlesztői módban hogyan fordíthatóak az alkalmazások. Fejlesztői módnál egy webszerver is elindul, valamint ha változást érzékel a forrásfájlokban, akkor automatikusan újrafordul. Így a terminálon kapott url-en és porton tekinthető meg az eredmény.

```
1 // Vue
2 npm run serve serve
3 // React
4 npm run start
5 // Angular
6 npm run start
```