ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Kustra Bence Attila**
2022	Hallgató törzskönyvi száma:	**T/005714/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Biomatika és Alkalmazott Mesterséges Intelligencia Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kustra Bence Attila**
Törzskönyvi száma: T/005714/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Rendszerkomponensek fejlesztése az autizmus korai észrevételéhez való eszközhöz
Development of components for a system that detects early onset autism

Intézményi konzulens: Dr. Eigner György
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

A feladat

Az autizmus spektrumzavar világszerte sok családot érint, és az egyik leggyakoribb neurológiai rendellenesség, mely gyakran születésétől fogva hátrányosan befolyásolja egy gyermek fejlődését. Mivel a súlyosság és a tünetek széles skálája jellemzi, gyakran nehéz diagnosztizálni, vagy diagnosztizálására késői életkorban kerül sor. Sok esetben az autizmus tünetei már gyermekkorban megjelennek - az iskola előtti időszak például kritikus, mivel a szülők addig gyakran nem is tudnak a gyermek esetleges állapotáról, különösen, ha az állapot enyhe, vagy a gyermek magasan funkcionáló autista, aki magas fokú intelligencián alapuló másolással pótolja a szociális képességek hiányát. Az autizmusra általánosságban jellemző, hogy minél korábban ismerik fel, és minél specifikusabban és személyre szabottabban kezelik, annál magasabb életminőségű lehet a gyermek egész további élete. A korai felismerés és fejlesztés nagyon pozitív hatással van a gyermek neurológiai fejlődésére és szociális működésére a későbbi felnőttkorban.
A jelölt részt vesz a Élettani Szabályozások Kutatóközpont részvételével kifejlesztett óvodai autizmus-felismerő rendszer egyes összetevőinek fejlesztésében, melynek célja a magasan funkcionáló óvodáskorú autista gyermekek rekogníciója.

A dolgozatnak tartalmaznia kell:

- irodalmi áttekintés,
- a fejlesztési keretrendszer kiválasztása,
- a front-end architektúra megtervezése,
- a front-end interfészek és az adatmanipulációs mechanizmusok megtervezése,
- az eredmények értékelése.

Ph.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

.....
Dr. Eigner György
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. 05. 14.

Kustra Bence

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: KUSTRA DENCE ATTILA

Neptun kód: [redacted]

Tagozat: MÉRNÖKINFORMATIKA

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Rendszerkomponensek fejlesztése az autizmus korai észleléséhez való eszközhöz

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of components for a system that detects early onset autism

Intézményi konzulens: Dr. Eigner György

Külső konzulens: -

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.10.	TÉMA MEGBESZÉLÉSE	
2.	2022.02.25.	SZAKIRODALMAK EGYEZTETÉSE	
3.	2022.03.07	SZAKIRODALOM ÁTNÉZÉSE	
4.	2022.03.22	SZAKIRODALMI RÉSZ ÁTNÉZÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.03.22.

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: KISTÓTA BENCE ATTILA

Neptun kód: [redacted]

Tagozat: MÉRNÖKINFORMATIKA

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Rendszerkomponensek fejlesztése az autizmus korai összetételéhez való eszközhöz

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of components for a system that detects early onset autism

Intézményi konzulens: Dr. Fogarasi György

Külső konzulens: -

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.04.12	KERETRENDSZEREK ÁTTEKINTÉSE	
2.	2022.04.21	KÓD ELLENŐRZÉSE	
3.	2022.05.03	FUNKCIÓK ELLENŐRZÉSE	
4.	2022.05.11	SZAKDOLGOZAT ÁTTEKINTÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 20 22. 05. 13.

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1. Bevezetés

Az emberiség már a történelem elejétől fogva próbálta megérteni a körülötte lévő világot, amiben nagy előrelépéseket tett az évezredek során. Miközben felfedeztünk, alkottunk, építettünk, körvonalazódott bennünk a kérdés, hogy mégis miért tesszük mindezt? Mivel nincs két egyforma ember, ezért két egyforma gondolkodásmód sincs. Mindenki egy kicsit máshogy értelmezi az őt körbevevő világot, embereket. Még egyeseknek ez könnyen megy, másoknak a számunkra teljesen egyszerűnek tűnő szituációk is nagy kihívást jelentenek. Társadalmunkban rengeteg fogyatékos ember él, akiknek számottevő része küzd minden nap az autizmus spektrumzavarral. Az autizmusnál fontos a minél korábbi diagnózis, hogy már gyermekkorban elkezdődhessen a személyre szabott oktatás. Ezt megnehezíti az, hogy kisgyermekkorban nem mindig merül fel a szülőkben, hogy gyermekük viselkedésével valami probléma van. Amennyiben viszont a szülőben, az óvoda pedagógusban vagy a pedagógusban felmerül a gyanú és szeretnék a kicsit kivizsgáltatni, egy szakértői bizottság tudja megállapítani, hogy rendelkezik-e a gyermek valamilyen fejlődési zavarral. Ahhoz, hogy ezt a diagnosztizálást megkönnyítsék, Dr. Eigner György és csapata elkezdték az AutiPlay nevű webes szoftver fejlesztését, melynél az én feladatom lesz a megjelenített adminisztrációs felület fejlesztése. A program egy helyen biztosítja a gyermek tesztelését és a tesztek követését a szakértő számra. Az AutiPlay különböző játékokat és arcfelismerő technológiát használ, hogy minél pontosabb adatokat tudjon nyújtani a vizsgálatot végző szakértőknek. Szerver oldalon a szoftver a PHP Laravel nevű keretrendszerét használja felhasználói oldalon pedig a Vue.js-t. A szakdolgozat során, beszélni fogok az autizmusról és annak vizsgálatáról, a Vue.js keretrendszerről és arról, hogy miben különbözik a piacon lévő többi keretrendszertől, majd részletes beszámolót adok az általam végzett frontend fejlesztésekről, amit a szakdolgozat végén majd értékelek.

2. Az autizmusról

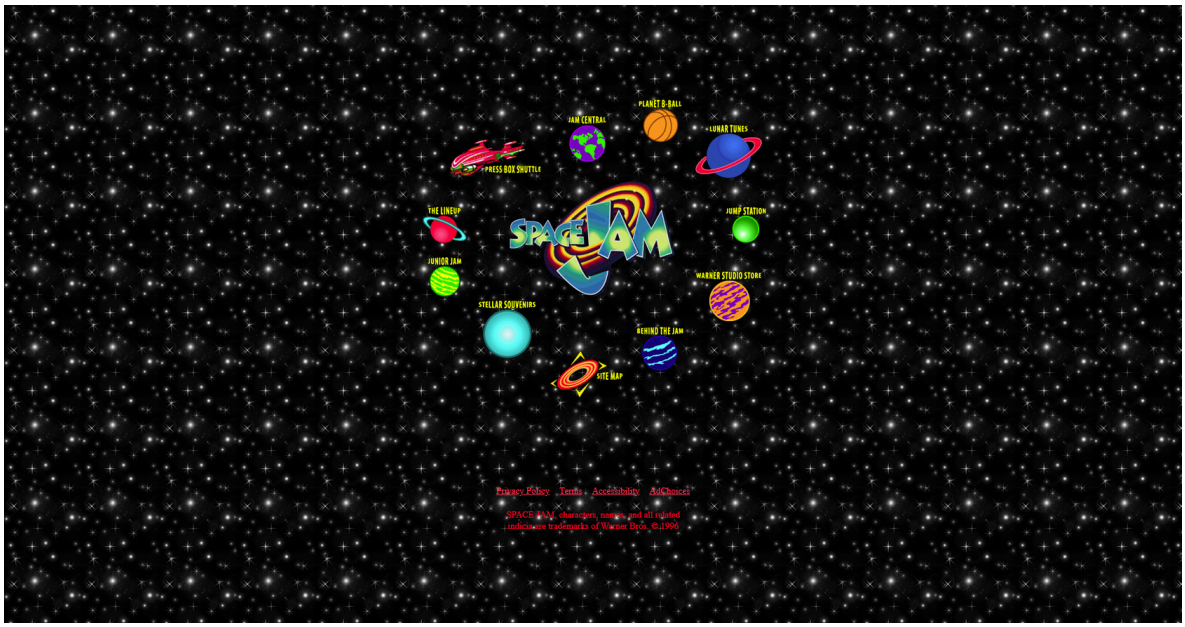
Az autizmus első dokumentálása az 1940-es évekre tehető, amikor az osztrák Hans Asperger és az amerikai Leo Kanner egymástól függetlenül találtak rá az autizmus két eltérő formájára [1]. Ezek után az 1960-as években indult el igazán az autizmus kutatása, ami azóta is rendületlenül tart. Napjainkban, a hivatalos megnevezés az autizmus spektrumzavar. Nevében a spektrum szó arra utal, hogy minden autistában máshogy jelenik meg ez a rendellenesség. Amikor az autizmusról beszélünk akkor három fő tünetet említhetünk, melyek minden autistában más-más mennyiségben fordulnak elő. Kommunikációs nehézségek: Itt általában nem a beszédet értjük, bár vannak esetek, ahol a gyermek nem tanul meg beszélni. Inkább arról van szó, hogy az autista nem tudja értelmezni a beszéd mögött megrejlő jelentést. Számunkra egyértelmű, hogy embertársunk beszéd közbeni hanglejtése valamilyen érzelmet fejez ki, nekik ezt nehéz, vagy olykor lehetetlen felismerni. Társas kapcsolatok: Íratlan szociális szabályok felismerése, betartása és megértése. Például egy autista gyermek nem valószínű, hogy látni fogja társa arcmimikájából vagy testmozgásából azt, hogy őt untatja vagy feltartja. Szociális képzelet: Sok autista számára nehézséget okoz az empátia, hogy mások helyzetébe bele tudják képzelni magukat. Ezért is fordulhat elő, hogy számunkra úgy tűnik, egy autistán beszéd közben nincs "szűrő", miközben számukra ennek létezése is kérdéses [2]. Kutatások régóta próbálják kideríteni, hogy valaki mitől lesz autista. Jelenlegi tudásunk és a kutatások arra mutatnak, hogy az autizmus erősen genetikailag meghatározott. Fontos, hogy az autizmus nem betegség, nem egy állapot, így nem is gyógyítható. Sok esetben, megfelelő tanítással és odafigyeléssel, egy autizmussal élő is ugyanolyan teljes életet tud élni, mint egy átlagos ember. Vannak autisták, akik kifejezetten kiemelkedőek érdeklődési köreikben az autizmusnak hála, mivel elméjük sokkal jobban rá van hangolva az apró részletek felismerésére és eme részletek közötti különbségek meglátására. Fontos látnunk, hogy az autistáknál a szociális képzelet hiánya nem egyenlő a hagyományos képzelet hiányával. Rengetegen vannak, akik művészi vagy mérnöki pályát választanak életük során és képesek nagy dolgokat alkotni. Azonban ahhoz, hogy ilyen eredményeket elérjenek, megfelelő oktatást kell kapniuk, mivel nem minden autizmussal élő képes hagyományos iskolákban tanulni. Fontos, hogy minél korábban megtaláljuk, hogy a gyermek, milyen módon képes optimálisan tanulni.

Vannak olyanok, akiknek szükséges minden háttér információ, ezért meg kell adnunk nekik a lehetőséget, hogy kérdéseket tehessenek fel, plusz tananyag legyen elérhető a számukra. Ezzel szemben olyan gyermekek is vannak, akik sokkal fogékonyabbak a vizuális tananyagra. Őket segíthetjük, ha videókat mutatunk nekik, vagy játékokat játszunk velük, amik a tananyaghoz kapcsolódnak. Ez viszont mind olyan információ, ami csak akkor derül ki, ha a gyermekről tudjuk, hogy teszteltetnünk kell és, hogy pontosan milyen teszteket kell elvégeztetni vele.

3. Fejlesztési keretrendszerek

3.1. Bevezetés

Az internet korai éveiben bárki akár egy délután alatt meg tudta tanulni hogyan készítsen el egy egyszerű weboldalt. Csupán értelmezte az alap HTML tag-eket, esetleg megnézte, hogy az egyes request-ek, hogy működnek, és már kész is volt a statikus weboldala. Ha tudjuk hol keressünk, akkor a mai napig találhatunk még ilyen oldalakat a weben. Erre egy példa a weboldal, amit az 1996-ban megjelent Space Jam-hez készített a Warner Bros.



3.1. ábra. 1996-os Space Jam weboldal

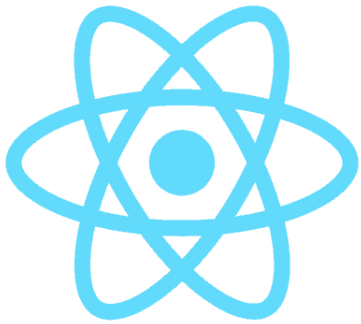
Mai szemmel ránézve elég mókásnak tűnhet, de 25 évvel ezelőtt ez igen sokat érő promóciós anyagként szolgált. Azonban ahogy teltek az évek, egyre jobban fejlődtek a technológiák, és a webfejlesztés lett az egyik legnépszerűbb ágazat a programozók között. Webfejlesztők között a legnépszerűbb nyelv a JavaScript, ami először 1995-ben jelent meg, azóta pedig az egyik legnagyobb programozási nyelvvé nőtte ki magát [3]. Manapság már minden böngésző rendelkezik saját beépített JavaScript motorjával, ami lefordítja az adott weboldalán lévő kódot. Ezek a motorok először csak böngészőkben voltak használva, de napjainkban már különböző applikációkban és szerver oldalon is gyakoriak. Legnépszerűbb megoldás közöttük a Node.js. A JavaScript, mint neve is sugallja, egy szkript nyelv, ami azt jelenti, hogy főként feladatok automatikus végrehajtására használják. A kód fordítása futásidőben történik [4].

3.2. Keretrendszerek

A szoftverfejlesztésben keretrendszer alatt egy olyan szoftverkörnyezetet értünk, ami szolgáltat számunkra egy alapfunkciót, és azt igényeink szerint tudjuk alakítani. Ezeket a keretrendszereket általában hozzá tudjuk rendelni valamilyen programozási nyelvhez, mint például az Angular-t a JavaScript-hez vagy a Laravel-t a PHP-hoz. Rengeteg előnye van annak, ha egy ilyen keretrendszert használunk. Mivel ezeket a környezeteket általában tapasztalt fejlesztők készítik, ezért biztosak lehetünk benne, hogy csak a sokat tesztelt és bizonyítottan jól működő funkciók kerülnek be. Ezen felül sok alapfunkció már be van építve, így nekünk azokat nem szükséges megírni, csak használni úgy, ahogy a keretrendszer elvárja tőlünk. Felmerülhet bennünk a kérdés, hogy miben különbözik akkor egy ilyen környezet egy sima könyvtártól. Még egy könyvtárban meghívjuk az egyes funkciókat és használjuk a visszakapott eredményt, addig egy keretrendszerben a mi kódunk van felhasználva és beillesztve az előre kreált folyamatokba. Szoftverfejlesztésben ezt hívják IoC (Inversion of Control) elvnek. Haszna abban rejlik, hogy így sokkal egyszerűbben tudunk moduláris szoftvert fejleszteni, ami napjainkban egy egyre fontosabb szempont a folyton növekvő kódbázisok mellett. Más érvek még, amik a keretrendszerek használata mellett szólnak, hogy segítenek a programtervezési minták (Design Pattern) egyszerűbb implementálásában, segítik a biztonságosabb kód írását (webfejlesztésnél ez azért fontos mert mindent meg kell tennünk a rosszindulatú felhasználók megakadályozásáért), egyszerűbbé teszik a tesztelést és jelentősen tudják csökkenteni a fejlesztéshez szükséges időt. A következő fejezetekben megvizsgálom, a népszerű webfejlesztéshez használt keretrendszereket és kiválasztom a számomra megfelelőt [5].

3.3. React

A React egy nyílt forráskódú, ingyenesen használható frontend JavaScript alapú keretrendszer, amit a Meta (korábban Facebook), egyéb cégek és független programozók fejlesztenek és tartanak karban. Főként single-page applikációk (SPA) és mobil applikációk fejlesztésére használják.



3.2. ábra. A React logója

A single-page applikáció egy olyan weboldal, ami ahelyett, hogy új tartalmak megjelenítéséhez egy teljesen új oldalt kérne le a szervertől, lekéri a megjelenítendő adatokat és újraírja azok alapján a weboldal tartalmát. Ennek az a célja, hogy gyorsabb töltési időket érjenek el és hogy egy natív applikációhoz közelebbi élményt kapjunk. A React egy deklaratív keretrendszer, amit többféle módon is magyarázhatunk. Mondhatjuk, hogy egy program deklaratív, amennyiben nem rendelkezik semmilyen "mellékhatással". Mellékhatásról, akkor beszélünk ha a program megváltoztatja olyan változók állapotát amelyek a lokális környezetén kívülre esnek. Hívhatjuk akkor is deklaratívnak, ha a program csak azt mondja meg, hogy mit hajtson végre az adott számítás, nem pedig azt, hogy hogyan. Tehát a szöges ellentettje az imperatív programozásnak [6]. A keretrendszer komponens alapú, minden elemet az ős komponensből származtatunk és utána csak azt változtatjuk, amire szükségünk van. Ettől programunk teljesen moduláris lesz, ami nagyon leegyszerűsíti a továbbá fejlesztéseket, vagy tervezés közben fellépő változtatásokat. Ezen felül képes állapotkezelésre, ami igazán megadja a dinamikus natív applikáció érzést. Minden komponens külön el tudja tárolni saját jelenlegi állapotát és frissíteni magát az alapján, amit megadunk [7].

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }
  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }
  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
  componentWillUnmount() {
    clearInterval(this.interval);
  }
  render() {
    return React.createElement(
      'div',
      null,
      'Seconds: ',
      this.state.seconds
    );
  }
}
ReactDOM.render(React.createElement(Timer, null),
document.getElementById('timer-example'));
```

A fenti kódrészletben mutatom be ezt a viselkedést. Először is létrehozzuk a **Timer** osztályt, ami a **React.Component** ősosztályból származik. Az osztály konstruktorában meghívjuk a **super** funkciót a props paraméterrel, hogy tudjuk a *this* kulcsszót használni, amivel az osztály sajat adattagjaira utalunk. Ezután létrehozzuk a **tick** metódust, amivel megváltoztatjuk a *seconds* adattagot azzal, hogy egyet hozzáadunk. Utána a **componentDidMount** metódusban másodpercenként egyszer meghívjuk a **tick** metódust. Azért itt tesszük ezt meg, mivel ez a metódus biztosít arról, hogy a benne lévő kód csak akkor kerül végrehajtásra ha a komponens már bekerült a DOM(Document Object Model)-ba. A **componentWillUnmount**-ban lévő kód hasonló módon akkor fut le, ha a komponens törlésre került a DOM-ból. Itt töröljük az időzítő értékét. Majd pedig a **render** metódusban megadjuk azt, hogy pontosan mit is szeretnénk megjeleníteni.

Az utolsó sorban, példányosítjuk a Timer osztályt, amivel elkészült a számlálónk. Még egy nagy előnye a React-nek a Virtuális DOM, ami a számítógép memóriájában eltárolja a weboldal jelenlegi struktúráját és változás esetén biztosítja, hogy csak a frissítendő komponensek tartalma változzon [8]. Ez hatalmas teljesítménynövekedést okoz és sokkal dinamikusabb érzést ad weboldalunknak.
Ez a keretrendszer csupán a megjelenítéssel foglalkozik, tehát a hálózati hozzáféréshez és egyéb feladatokhoz más külső könyvtárak használata szükséges.

3.4. Angular

Az Angular egy Google és más cégek, független fejlesztők által készített nyílt forráskódú, ingyenes keretrendszer, amit webes applikációkhoz használnak. Amikor először keresünk információt az Angular-ről összezavaró információkat ta-

3.3. ábra. Az Angular logója

lálhatunk, ameddig nem tisztázzuk magunkban, hogy szükséges megkülönböztetnünk az Angular-t és az AngularJS-t. Az AngularJS, mint nevében is látható, JavaScript alapú, míg az Angular választott programozási nyelve a TypeScript. A TypeScript a JavaScript egy kiegészítése. Legnagyobb különbség a kettő között, amiből a TypeScript név is adódik, hogy a nyelv erősen típusos. Ez azt jelenti, hogy még a JavaScript-ben típus az értékhez van rendelve, addig itt a változóhoz. Ez viszont magával vonzza azt, hogy a kódot fordítani kell. Nagyobb kódbázisokhoz a TypeScript jobb teljesítményt tud nyújtani, és a kód is könnyebben olvasható, ezért építhetett erre a nyelvre a Google. Mivel az AngularJS már nem rendelkezik hivatalos támogatással, ezért a továbbiakban az Angular-re fogok fókuszálni [9].
A keretrendszer segítségével fejleszteni tudunk SPA-kat és PWA (Progressive Web Application)-kat. A PWA egy olyan módszer, amelynek segítségével a weboldalunk

úgy tud viselkedni, mint egy natív mobil applikáció [10]. Amikor elkezdünk dolgozni, ugyanúgy megírjuk a kódunk HTML részét, mintha keretrendszer nélkül fejlesztenénk. Viszont itt már meg is jelenik az első nagy különbség, a direktívák. Ezekre tekinthetünk, úgy, mint HTML attribútumokra. A kódban ugyanott helyezkednek el, viszont sokkal több lehetőséggel rendelkeznek. A következőben, talán a leghasznosabb beépített direktívát szeretném bemutatni az **ngModel**-t. Ennek a segítségével létre tudunk hozni egy kétirányú összekötést egy HTML Form elem és a kód között. Ennek segítségével egyszerűen tudjuk dinamikusan változtatni weboldalunk tartalmát a felhasználó által bevitt adatok alapján. Ahhoz, hogy ezt használni tudjuk, csupán meg kell adnunk az importálandó modult (ebben az esetben FormsModule) és már tudjuk is használni [11].

```
import { Component, OnInit } from '@angular/core';
import { Item } from './item';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  item!: Item;
  currentItem!: Item;
}
```

Az első sorban láthatjuk, hogy importáljuk az *Item* osztályt, ami rendelkezik pár adattaggal, amik közül számunkra a *name* számít. A *templateUrl*-el hozzárendeljük a HTML fájlunkhoz, és alatta az **OnInit** metóduson belül létrehozzuk az *Item*-et és a *currentItem*-et. Ennek a segítségével tudjuk használni az **ngModel**-t a HTML-ben és jön létre a kétirányú adatkötés.

```
  <p>Current item name: {{currentItem.name}}</p>
  <p>
    <label for="example-ngModel">[(ngModel)]:</label>
    <input [(ngModel)]="currentItem.name" id="example-ngModel">
  </p>
```

Itt pedig láthatjuk a HTML-t, ahol a *label* elemben meg fog jelenni az, amit az alatta lévő *input*-ba írunk. Az Angular egy egyszerűsített MVC(Model-View-Controller) szoftverarchitektúrát A fő különbség az, hogy itt nem szükséges felbontanunk a

kódot, különböző komponensekre az MVC elvek alapján, és utána megírnunk, az összekötéseket, mivel az összekötés részét a keretrendszer megcsinálja helyettünk. React-hez hasonlóan, a kód az Angular-ben is nagyon moduláris, ezzel megkönnyítve egyes komponensek változtatását, cseréjét, ami miatt alkalmas nagy kódbázisok támogatására. A szintaktika nagyon szigorú, cserébe viszont a kód nagyon konzisztens, fejlesztőtől függetlenül. Ez olvashatóbbá teszi, egyszerűen karbantarthatóvá és könnyen tesztelhetővé sok kód mellett is.

3.5. Vue.js

A Vue.js egy ingyenes, nyílt forráskódú TypeScript frontend keretrendszer, amit Evan You készített és jelenleg is a saját csapatával dolgozik rajta. Felhasználói felületek és SPA-k fejlesztésére használható. A projekt először 2014-ben jelent meg,

3.4. ábra. A Vue.js logója

miután Evan You kilépett a Google-től, ahol sok alkalmazás fejlesztéséhez használták az AngularJS-t. Egy interjúban arról, nyilatkozott, hogy tervezés közben az volt a célja, hogy az Angular-ből kiemeljen mindent, amit szeretett, de egy mégis kis teljesítményt igénylő keretrendszert alkosson [12]. Az elmúlt években rengeteg fejlesztésen ment keresztül a rendszer, és 2022.február 7-től, a Vue 3 az alapvető verzió. Rengetegen használják, közel 10 millió letöltést ér el havonta az npm csomagkezelőn keresztül és legalább másfél millió felhasználóval rendelkezik. Legfontosabb tulajdonsága, hogy mindent komponensekre bont, aminek hála nagyon moduláris és jól skálázható. Ezeknek az elemeknek külön meg tudjuk adni a viselkedését és a kinézetét, mivel a keretrendszer egy kiterjesztett HTML szintaktikával dolgozik. Erre a megoldásra "Single-File Component"-ként hivatkoznak. Nevéből is adódóan, ezek a komponensek egy külön fájlban helyezkednek el, aminek .vue a kiterjesztése [13]. Egyszerűség kedvéért vegyük a következő kód részletet.

```
<script>
export default {
  data() {
    return {
      count: 0
```

```
    }
  },

  methods: {
    increment() {
      this.count++
    }
  },

  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">count is: {{ count }}</button>
</template>
```

A *<script>* elemben láthatjuk, hogy meghívjuk a **data()** metódust, amiben eltárolunk egy számot, ami jelenleg nullában áll. Majd a **methods** nevet felhasználva megadjuk a szükséges metódusainkat. Nekünk jelenleg csak az **increment** szükséges, ami egyel növelni fogja a számlálónkat. A **mounted**, egy beépített metódus, ami akkor kerül meghívásra, ha az elem elhelyezésre került a DOM-ban. Ez jelenleg csak annyit csinál, hogy a konzolra kiírja a számláló kezdetleges állását. Majd miután bezártuk a *<script>* elemet, megnyitunk egy *<template>* elemet, amiben elhelyezzük a HTML-t. Ide teszünk egy *<button>*-t, aminek a **@click** eseményére (ami értelemszerűen akkor következik be, ha rákattintunk az elemre) meghívjuk az **increment** metódust, ami növeli egyel a belső számlálót. Ezen kívül a gombban leírjuk a számláló jelenlegi értékét. Ezzel megkapjuk ezt a gombot:

3.5. ábra. Vue.js gomb példa

Ebben a példában látni a Vue.js egyik nagy előnyét, hogy nincs szükség külön a HTML manipulálására. Mi csak a komponens belső értéket változtattuk meg, de ez egyből látható volt a HTML-ben is. A keretrendszerben erre "Declarative Rendering" néven hivatkoznak.

A kódolás nagyon egyszerű és könnyű megtanulni, a kód első ránézésre is megérthető, nem kell törni rajta a fejünket, mint a React-nél vagy Angular-nél. Az eddig bemutatott rendszerek közül talán ez rendelkezik a legjobb dokumentációval. Felépítése logikus, könnyen követhető ezért könnyű elkezdeni, ha az ember még nem dolgozott vele, de a nehezebb funkciókat is egyszerű implementálni, mivel jó eséllyel mindig meg fogjuk találni azt, amit keresünk.

3.6. Ember.js

Most, hogy beszéltem a frontend keretrendszerek nagy hármasáról, szeretnék bemutatni egy olyan megoldást, ami talán nem annyira ismert, mégis hatalmas cégek használják. Az Ember.js egy nyílt forráskódú JavaScript alapú keretrendszer, amit olyan óriások használnak, mint a Netflix, LinkedIn, Microsoft vagy Apple [14]. Azért

3.6. ábra. Az Ember.js logója

választották ezt a rendszert, mivel nagyon jól skálázható ahogy növekszik a kódbázis mérete. Viszont ennek az az ára, hogy az Ember.js kód struktúrája nagyon merev, és sokszor olyan dolgokat is szükséges megírni, amit feleslegesnek tarthatunk éppen. Ez mellé még kapunk egy elég meredek tanulási görbét is, így nem meglepő, hogy sok fejlesztő inkább más keretrendszert használ.
Mint sok más keretrendszer az Ember.js is komponens alapú, bár nem annyira drasztikus módon, mint a Vue.js. Itt a komponensek inkább speciális HTML elemeknek tűnnek első ránézésre [15]. Mivel nagy applikációkra gondoltak elsősorban a keretrendszer tervezésekor, ezért nagy hangsúlyt fektettek az egyszerű tesztelésre. Az Ember CLI-n(Command Line Interface) keresztül, amit úgy kell elképzelni, mint a saját terminálunkat egy Ember.js rendszeren belül, tudunk generáltatni előre megírt tesztfájlokat, ahol a kód általánosan szükséges részével már nekünk nem kell foglalkoznunk. Ezen kívül a CLI-n keresztül tudunk komponenseket is generáltatni, ezzel tovább gyorsítva a munkafolyamatot [16]. Ez a keretrendszer is rendelkezik kétoldalú adatkötéssel, így, ha egy komponensen belül megváltoztatunk valamilyen adatot, akkor nem kell azzal foglalkozzunk, hogy az ezt reprezentáló HTML-t is megváltoztassuk. Ezt úgy oldják meg, hogy különböző “decorator”-eket adnak a megfigyelendő változónak, ami jelzést ad az Ember-nek, hogy ha az adott adattagnál változást észlel, akkor vizsgálja meg, hogy szükséges-e a DOM tartalmát változtatni. Ezt bemutatom a következő példával:

```
import Component from '@glimmer/component';
import { tracked } from '@glimmer/tracking';
import { action } from '@ember/object';

export default class RentalImageComponent
extends Component {
  @tracked isLarge = false;

  @action toggleSize() {
    this.isLarge = !this.isLarge;
  }
```

Itt láthatjuk, hogy a **RentalImageComponent** komponens rendelkezik egy *isLarge* változóval, ami azt jelzi, hogy a megjelenített kép kicsi vagy nagy méretű legyen. Rá van kötve a *@tracked* decorator, ami jelzést ad az Ember-nek, hogy ha változik ennek a változónak az értéke, akkor lehet, hogy változtatni kell a DOM-ban. Az *@action* decorator pedig jelzi, hogy a **toggleSize()** metódus meghívható a DOM-ból.

```
{{#if this.isLarge}}
  <button type="button" class="image large"
  {{on "click" this.toggleSize}}>
    <img ...attributes>
    <small>View Smaller</small>
  </button>
{{else}}
  <button type="button" class="image"
  {{on "click" this.toggleSize}}>
    <img ...attributes>
    <small>View Larger</small>
  </button>
{{/if}}
```

Ha az *isLarge* adattag igazat ad vissza, akkor hozzáadjuk a *large* osztályt az osztálylistához, amitől egy nagy kép jelenik meg, ha pedig ismét a képre kattintunk akkor összemegy. Láthatjuk, hogy a *button* elemhez hozzáadtuk a **click** eseményt, ami meghívja a **toggleSize()** metódust, ezzel állítva az *isLarge* változót. Ez nyilván egy nagyon egyszerű példa, de ad számunkra egy kis betekintést az Ember.js-el való munkába.

3.7. Backbone.js

A Backbone.js egy érdekes projekt, amit Jeremy Ashkenas azzal a céllal alkotott, hogy a piacon legyen olyan opció, ami nem köti meg annyira a kezünket, mint a többi. A készítő saját elmondása szerint ez nem egy keretrendszer, hanem inkább egy könyvtár, de használat közben eléggé azt érezhetjük, hogy a kettő között mozgunk. Működése az MVC (Model-View-Controller) programozói modellen alapul,

3.7. ábra. Backbone.js logó

ahol a modellel reprezentáljuk a megjelenítendő adatokat, a view pedig végzi magát a megjelenítést. Ez a felépítés segít minket abban, hogy egy jól rétegzett alkalmazást kapjunk, tehát a logikánk el legyen szeparálva a felhasználói felületünktől [17]. Ez megkönnyíti a későbbi változtatásokat, ha a szoftvert más irányba szeretnénk vinni. A korábbiaktól eltérően, itt nem feltétlen beszélhetünk kétoldali adatkötésről, mivel a model, az esetek nagy részében nem ismeri a view tartalmát, sőt még a létezéséről sem tud. A view dolgozik a model-től kapott adatokkal, és ha valamilyen változás történik, akkor a model elsüt egy "change" eseményt, amit utána a view elkap, és annak megfelelően változtatja a tartalmát. A Backbone egy előre konfigurált JSON (JavaScript Object Notation) RESTful API-val érkezik. A JSON, egy egyszerű adatcserére használt adatformátum, ami az emberek számára könnyen olvasható és írható, a gépek számára pedig könnyen generálható és feldolgozható [18]. A RESTful API (vagy röviden REST API) pedig egy olyan végpontot jelöl a szerverünkön, ami lehetővé teszi azt, hogy az adatokat manipulálhassuk különböző HTTP vagy HTTPS kérések segítségével [19].

3.8. Konklúzió

Ez a program egy több fejlesztő közötti kollaboráció, amit fontos figyelembe venni a döntésnél. Az első kérdés az lehet, hogy miért szükséges egyáltalán egy keretrendszer? Az internet nagy részét még mindig olyan weboldalak dominálják, melyek egyszerűen kombinálják a HTML-t a JavaScript-tel. Míg ez kisebb projekteknél teljesen jól működik, ha nem figyelünk oda, akkor nagyon gyorsan túlnőhet rajtunk az adott projekt. Amennyiben nem dokumentáljuk megfelelően a kódunkat, és nem követünk egy logikus kód struktúrát, akkor másoknak nagyon nehezen olvasható lesz a kód. Ez a probléma orvosolható egy keretrendszer használatával,

mivel ott minden fejlesztő egy struktúrába van beleszorítva. Emiatt az ok miatt, értelemszerű itt egy keretrendszert választani, a kérdés már csak az, hogy melyiket. Fontos, hogy a rendszer gyorsan átlátható legyen, és jó dokumentációval rendelkezzen, hogy minél gyorsabb és minél pontosabb munkát tudjunk végezni. Ez kizárja a React-et, mivel sok helyen kritizálják, hogy egyes részek nehezen megérthetőek a hivatalos dokumentációban és ezért külső forrásokban kell keresni az információt. Ez, és a keretrendszer sokszor szükségtelen komplexitása miatt elvetettük a Meta által használt megoldást. A másik, gigantikus vállalat által fejlesztett rendszer, amivel foglalkoztunk az Angular volt. Rengeteg dolgot megtalálunk beépítve, amit pedig nem, ahhoz van külön harmadik féltől való támogatás a hatalmas népszerűségnek hála. Ez a rengeteg funkció viszont azzal jár, hogy a keretrendszer nagy méretű, ami teljesítménycsökkenéssel járhat amennyiben nem optimalizáljuk megfelelően a kódunkat. Emiatt, és az elég meredek tanulási görbe miatt úgy döntöttünk, hogy a React-et is elvetjük. Így versenyben maradt az Ember.js és a Vue.js. Mindkettőnek nagy előnye, hogy rendelkeznek saját beépített CLI-vel, így sok monoton feladat megoldható egy egyszerű paranccsal. Mindkét keretrendszer rendelkezik kétoldalú adatkötéssel, hogy ne kelljen külön foglalkoznunk a DOM frissítésével. A Vue.js erős modularitása, könnyít a fejlesztésen, és a munka közben jelentkező változtatásokhoz való gyors adaptáción. Remek dokumentációval rendelkezik, és népszerűségének hála segítséget is tudunk kérni különböző fórumokon. Kis méretű és jól optimalizált, így gyengébb rendszereken is jól teljesít weboldalunk. Az egyetlen nagy előnye az Ember.js-nek a Vue.js-el szemben, az az előre beépített router. Ez a szoftver felel azért, hogy a megfelelő URL-re a megfelelő tartalmat adja vissza a szerver a felhasználónak [20]. Ez mégsem elég ahhoz, hogy hátrányait ellensúlyozza. Nehéz vele elkezdeni a munkát, és inkább nagy projektek készítéséhez alkalmas. Ezek mellett nagy méretű, ami rontja a teljesítményt. Így a végére a Vue.js lett a győztes keretrendszer, a továbbiakban ezzel fogunk dolgozni.

4. CSS Keretrendszerek

4.1. Bevezetés

Ahhoz, hogy egy weboldal használata élvezetes legyen és a felhasználó gyorsan megtaláljon mindent amire szüksége van, remek eszköt a CSS (Cascading Style Sheets).

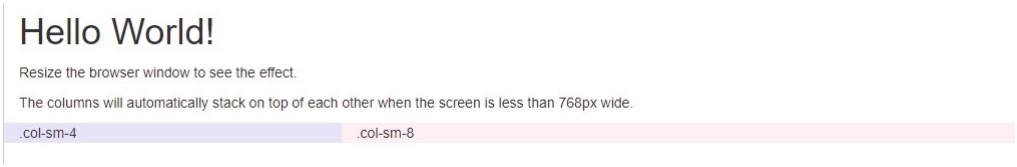
Lényegében, a CSS felel azért, hogy weboldalunk esztétikus, és szépen elrendezett legyen. Egyszerű feladatoktól kezdve, mint például egy gomb sarkának a lekerekítése, akár olyan nehéz feladatok is megoldhatóak, mint egy teljes animáció elkészítése. Viszont ahhoz, hogy ilyen komplikált dolgokat alkossunk, rengeteg tapasztalatra és a CSS olykor bonyolult interakcióinak megértésére van szükségünk. Ha a weboldalunkon minden CSS-t mi szeretnénk írni, arra rengeteg időt kell szentelnünk, és még úgy sem biztos, hogy a kívánt eredményt el fogjuk érni. Amennyiben nem rendelkezünk elég tudással és szeretnénk, hogy a projektünkön belül az összes fejlesztő konziztens stílussal oldja meg a dizájn problémákat, érdemes egy CSS keretrendszert használnunk. A következő pár fejezetben meg fogom vizsgálni a legnépszerűbb opciókat és a végén eldönteni, hogy melyiket használjuk. A fő szempontok, amiket vizsgálok, az a használat egyszerűsége, módosíthatóság és a teljesítmény.

4.2. Bootstrap

A Bootstrap-et a Twitter készítette és tartja karban a mai napig. 2011-ben osztották meg a világgal és azóta ingyenesen használható mindenki számára. A projekt nyílt forráskódú, tehát ha valamin változtatni szeretnénk arra is van lehetőségünk. Talán az egyik legnépszerűbb CSS keretrendszer a piacon. A BuiltWith szerint több mint 20 millió weboldal használ Bootsrap-et [21]. A rendszer nem teljesen CSS alapú, mivel egyes dolgokhoz JavaScript-et is használtak, emiatt szükséges minden oldal indításánál egy szkriptet is betöltenünk, ami hozzáad a töltési időhöz. Fejlesztésénél az egyik fő szempont volt a "Mobile-First" szemlélet és a mai napig is rengeteg energiát áldoznak arra, hogy a lehető legtöbb képernyőmérettel kompatibilis legyen [22]. Az alapelgondolás az, hogy a weboldalra úgy tekintünk, mint egy rácshálóval felosztott területre. Ebben a hálóban mindent 12 egyenlő szélességű oszlopra bontunk, a sorok nagysága tetszőleges. A tartalmat így tudjunk nekünk megfelelően elhelyezni. Hogy még jobban segítsék a kompatibilitást, különböző osztályok megadásával megadhatjuk, hogy bizonyos tartalmakat csak a megfelelő méretű képernyőn mutassunk. Itt látható egy egyszerű példa:

```
<div class="container-fluid">
  <h1>Hello World!</h1>
  <div class="row">
    <div class="col-sm-4"
     style="background-color:lavender;">.col-sm-4</div>
    <div class="col-sm-8"
     style="background-color:lavenderblush;">.col-sm-8</div>
  </div>
</div>
```

Először is létrehozunk egy *div* elemet a "container-fluid" osztállyal, hogy megjelöljük a Bootstrap-nek, hogy hol kell a tartalmat kezelnie. Majd ebben a konténerben létrehozunk egy újabb *div*-et, "row" osztállyal. Ez adja meg, hogy a benne lévő tartalmat egy sorba szeretnénk elhelyezni. Ebben a sorban pedig elhelyezünk kettő további *div*-et "col-sm-" osztályokkal, ahol az "sm" a kis méretet jelzi, a számok pedig azt, hogy szélességben mennyit foglaljanak el, a 12 egységből, amivel a Bootstrap dolgozik. Végeredményként pedig ezt az oldalt kapjuk.



4.1. ábra. Bootstrap példa (a fenti HTML-ből pár sort kihagytam, az egyszerűség kedvéért)

Természetesen a Bootstrap ettől jóval többet tud. A legtöbb alapvető HTML elem rendelkezik saját stílussal, hogy weboldalunk egy egységes kinézettel rendelkezzen. Ezen kívül találhatunk rengeteg tábla stílust, navigációs sávokat, oldalon belül elhelyezkedő felugró ablakokat és még rengeteg más dolgot. Mivel a Twitter által karbantartott és fejlesztett keretrendszerről van szó, bízhatunk abban, hogy folyamatos frissítések fognak érkezni és a dokumentáció is naprakész lesz. Ezek a szempontok nagyban megkönnyítik a termékkel először dolgozók dolgát.

4.3. Tailwind CSS

A Tailwind CSS azért készült, mert egy frontend fejlesztő megelégelte a "seperate your concerns" webdizájn alapelvet. A lényege ennek az elvnek, hogy amikor létrehozzuk a weboldalunkat, akkor az tartalom orientált legyen, ne stílus orientált. Ez a gyakorlatban úgy képzelhető el, hogy a HTML-ünk "nem tud" a CSS-ről, amit felhasználunk hozzá. Így elméletben a megfelelő osztálynevekkel bármilyen CSS fájlt felhasználhatunk a weboldalunkon, amíg az osztály nevek ugyanazok, és radikálisan különböző kinézeteket érhetünk el [23]. Erre a legjobb példa a CSS Zen Garden, ahol ugyanazt a tartalmat olvashatjuk, miközben válthatunk rengeteg, a közösség által készített kinézet között.

Ez egy "utility-first" keretrendszer, ami azt jelenti, hogy egyszerű, hasznos elemekből építünk fel komplex dizájnokat. Ezt úgy érik el, hogy szinte minden CSS-hez, amit fel szeretnénk használni létrehoztak különböző osztályokat.

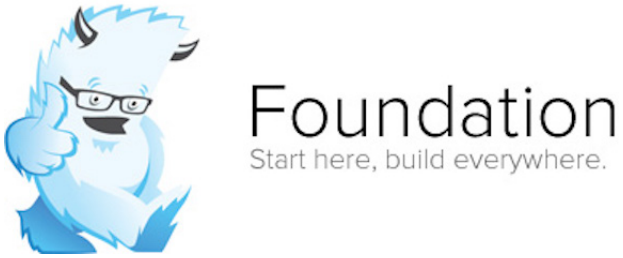
```
<img class="h-12 w-12"
src="/img/logo.svg" alt="ChitChat Logo">
```

4.2. ábra. CSS Zen Garden megvalósítások

Ez az egyszerű példa megmutatja, hogy a Tailwind segítségével hogyan helyezünk el és méretezünk egy képet. A “h-12” és a “w-12” osztályok “3rem” (a CSS-ben a “rem” egy olyan mértékegység, amely a méretét a webdokumentum gyökér elemének betűméretéből kapja) nagyságot és szélességet állít be a képnek. A keretrendszert eleinte kicsit nehézkes használni, ezt viszont ellensúlyozza a remek dokumentáció. Ami nagyon vonzóvá teszi ezt a rendszert, hogy a program élesítése közben automatikusan eltávolít minden elemet a CSS fájlunkból, amit nem használunk. Így ennek az optimalizálása előre el van végezve, nekünk csak a fejlesztéssel kell foglalkozni. Természetesen az, hogy nem kell weboldalunknak feleslegesen nagy fájlokat betöltenie minden megnyitáskor, a teljesítményt is nagyban növeli.

4.4. Foundation

A Foundation-t a ZURB nevű vállalat készítette, 2019 óta viszont önkéntesek fejlesztik és tartják karban. Felépítésében inkább hasonlít a Bootstrap-re, mint a



4.3. ábra. A Foundation logója

Tailwind-re, mivel főként nagyobb egységeket találunk meg benne, nem mi vagyunk a felelősek minden apró elemért. Természetesen, ha ezeket felül szeretnénk írni, arra mindig meg van a lehetőségünk a CSS fájlokon belül. Mint írtam, hasonlít a Bootstrap-re, ezért sok olyan jellemzőt fogunk látni itt, amit ott is. "Mobile First" filozófiát követnek, és a weboldal felosztását itt is a rácsháló elképzelésben valósul meg. A legnagyobb különbség viszont a Bootstrap-pel szemben az a lehetőségek mennyisége, szinte mindenre találunk megoldást, ami csak eszünkbe juthat, kártyák, hibaüzenetek, rengeteg különböző menürendszer, navigációs sáv és a többi. Ez a rengeteg lehetőség viszont azzal a nagy problémával jár, hogy a betöltendő adatok mennyisége lényegesen nagyobb, mint a Tailwind esetében, habár vehetjük úgy, hogy a Tailwind "csal", de ezt a konklúzióban szeretném bemutatni. A Foundation, mint a Bootstrap is a "SASS" kiegészítő nyelvre van építve. Ennek a célja, hogy a segítse a programozók dolgát CSS írás közben olyan dolgokkal, mint változó nevek, vagy egymásba ágyazás, ami nagyon hasznos, ha azt szeretnénk, hogy kódunk könnyen olvasható legyen [24].
A Foundation véleményem szerint nehezebben értelmezhető egy kezdő számára és a dokumentációt sem találtam olyan kiforrottnak, mint az előző két rendszernél.

4.5. Konklúzió

Mint azt az előző fejezetben leírtam, itt is fontos volt, hogy a keretrendszer gyors legyen és könnyen megérthető. Az érthetőséget segíti a jó és naprakész dokumentáció, amiben könnyű navigálni. Hiányoltam a Foundation-nél egy írott kezdő példát, aminek a segítségével az ember el tudna indulni a Bootstrap-pel és Tailwind-del ellentétben, ahol az ember pár kattintás után máris lát egy kis bemutatót, hogy lássa milyen lehet a munka az adott keretrendszerrel. Ez ki is zárta a Foundation-t, így maradt a másik két jelölt. Lényegesen eltérő filozófiával rendelkeznek. Míg a Bootstrap nagy egységeket ad a kezünkbe, hogy azokból építkezzünk, addig a Tailwind a lehető legkisebb elemekre épít. Ezért az utóbbi használata sokkal közelebb van a hagyományos CSS írásához, és főként azt próbálja csak egyszerűbbé tenni. Ezért ez sokkal könnyebben tanulható, ha már van valamilyen korábbi tapasztalatunk CSS írásában. A másik fontos szempont a teljesítmény volt, amit itt a betöltendő fájlok mérete nagyban befolyásol.

Ezekből az adatokból azt szűrhetnénk le, hogy a Bootstrap az egyértelmű döntés mivel a Tailwind 17-szer nagyobb tőle[25], viszont itt jelentkezik az a "csalás", amiről írtam az előző fejezetben, hiszen a Tailwind, csak azokat a CSS osztályokat csomagolja nekünk, amiket használunk, így a fejlesztők elmondása szerint a fájlméret még nagy projektek esetén sem fogja meghaladni a 10kB méretet[26], ezzel teljesen átdöntve a mérleget erre az oldalra. A teljesítménybeli különbség és az,

Site	Size	CSS source lines	CSS lines
Bootstrap	163.82 kB	7336	6
TailwindCSS	2.93 MB	97518	1

4.4. ábra. Bootstrap és Tailwind méret összehasonlítás

hogy a Tailwind nem köti meg a kezünket miközben mégis gyorsítja a munkánkat, számunkra az egyértelmű győztessé tette.

5. Fejlesztési környezet

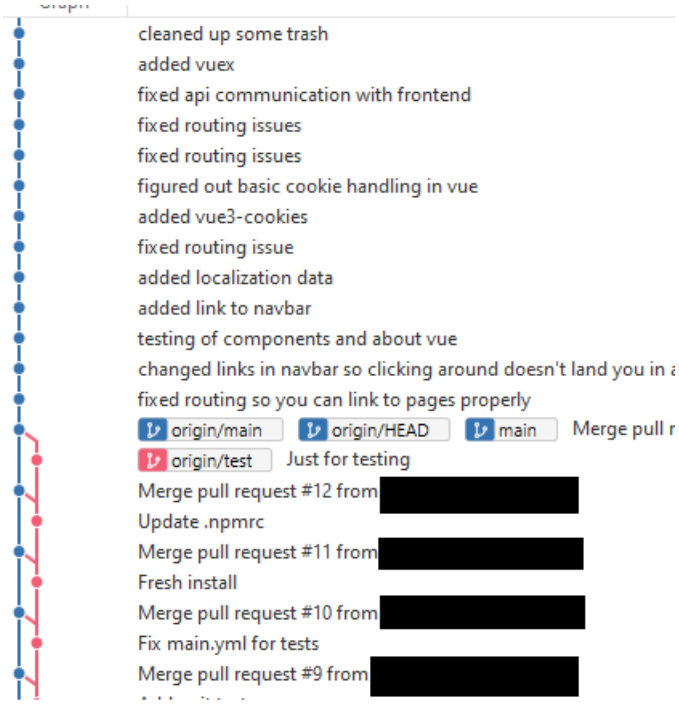
Ahhoz, hogy egy ilyen projekt fejlesztése zökkenőmentesen menjen, szükséges a megfelelő fejlesztési környezet kialakítása. Ez sok különböző elemből épül fel, mint az IDE (integrated development environment), a fejlesztési szerverek és a megfelelő verziókövetés.

5.1. IDE

Az IDE legtöbbször preferencia, vagy a lehetőség kérdése. Rengeteg opció van jelenleg a piacon, mind különböző erősségekkel és gyengeségekkel. Szerencsére egyre több ingyenes választási lehetőség is van manapság, ezek közül egy, a Microsoft által fejlesztett Visual Studio Code, ami mellett én is döntöttem. Nagyon hasznos a beépített terminál, aminek segítségével egyből a kódszerkesztőből indíthajtuk a Vue.js fejlesztési szerverét és valós időben láthatjuk a változtatásokat, mivel a Vue CLI minden mentésnél újra építi a projektet. Ezen felül nyílt forráskódú, ami lehetővé teszi a VS Code felhasználói közösségét, hogy különböző kiegészítőket készítsenek, mint az automata kód formázók, vagy a különböző színezések, aminek köszönhetően jobban átlátható lesz a projektünk. Rendelkezik még verziókövető integrációval is, ami nagyon hasznos, mivel láthatjuk, hogy melyik fájlunk változott meg a repository-ban tároltakhaz képest, és mi az, amit töröltünk, hozzáadtunk.

5.2. Verziókövetés

Napjainkban elképzelhetetlen a csapatban való fejlesztés anélkül, hogy valamilyen verziókövető rendszert alkalmazzunk. Lehetővé teszi az egyszerű a tesztelést, annak a követését, hogy ki, milyen kódért felelős és az adott változatokhoz a különböző modulok fejlesztését. A projekt a GitHub-ra van feltöltve, ahonnan a megfelelő engedéllyel ez klónozható a saját számítógépünkre. Az, hogy milyen megoldást használunk ezek után a Git műveletekre, ismételten személyes preferencia kérdése. Haladó felhasználók gyakran esküsznek a terminálra, mivel ismerik az összes parancsot, és úgy érzik, hogy számukra egy grafikus felhasználói felület inkább csak felesleges navigációs problémákat okoz, a segítség helyett. Kezdők esetében viszont sokat segíthet valamilyen külön szoftver, ami útmutatásd ad. Én a Sourcetree mellett döntöttem, mivel számomra segíti az átláthatóságot, hogy a feltöltésekhez van interaktív vizualizáció.



5.1. ábra. Sourcetree példa

A fenti képen látható az említett vizualizáció. Amennyiben többen is dolgozunk egyszerre, több különböző funkción, nagyban segíthet a megfelelő verzió megtalálásában, vagy az esetleges kompatibilitási hibák megkeresésében.

5.3. Adatbázis szerver

A projekt fejlesztési struktúrája felépül egy frontend szerverből, amit a Vue CLI indít nekünk az npm segítségével, ami kommunikál a backend szerverrel. Ennek a backend szervernek viszont szükséges valamilyen adatbázisban tárolni az adatokat, ehhez pedig az XAMMP nevű programot használjuk. Ez egy nagyon hasznos szoftvercsomag, amivel lokális szervereket tudunk futtatni egy gombnyomással. Alapértelmezett lehetőségei között tartalmaz egy Apache és egy MySQL szervert is, aminek segítségével könnyen tudjuk konfigurálni adatbázisunkat a böngészőnkből, a phpMyAdmin felületen keresztül. Azonban ezek csak a fejlesztést segítő eszközök, a rendszer élesítése közben, a "host"-olási platformtól függően szükséges lesz a további konfiguráció.

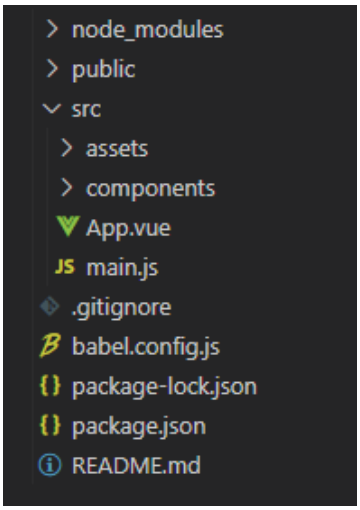
5.4. Backend szerver

A backend szerverünk a Symfony keretrendszert használja, ami a php programozási nyelvre épül. Ahhoz, hogy a Symfony megfelelő módon működjön, különböző kiegészítő .dll-eket szükséges engedélyeznünk a php installációnkban. Fontos megemlíteni, hogy a Symfony, a php nyelvvel együtt fejlődik és mivel a mi projektünk még Symfony 4-ben készült, jelenleg pedig a 6.0-ás verziószámnál járunk, ezért, ha túlságosan új php verziót telepítünk, az kompatibilitási hibákkal járhat. Ezután telepíthetjük a Symfony-t és ha egy meglévő projekttel dolgozunk, akkor a composer csomagkezelő, minden szükséges csomagot letölt nekünk, amit a projekt használ. Fejlesztés folyamán ezt a szervert a Windows PowerShell alkalmazásból futtattam a php segítségével. Mivel a Symfony-t REST API-ként használjuk, ezért a frontend nincs beleintegrálva. Ez előnyökkel és hátrányokkal is jár. Előnynek számít, hogy a legtöbb számítási kapacitást rá tudjuk terhelni a felhasználó készülékére ezzel csökkentve a szerver által igényelt teljesítményt, hátrány viszont, hogy össze kell hangolnunk a két szerver működését az élesítési folyamat közben. Nagy előny volt ennél a projektnél a Doctrine nevű könyvtár használata, aminek segítségével nem kellett az adatbázis táblákat kézzel rekreálnunk, ezt a szoftver megtette helyettünk

a megfelelő parancsok után.

6. Vue projekt felépítése

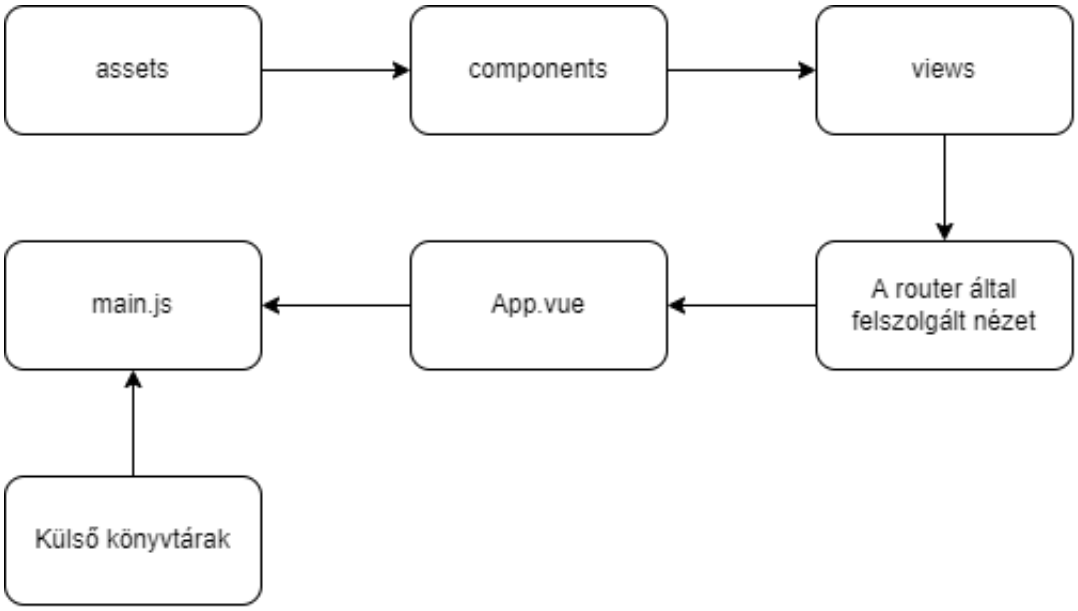
A Vue.js egy komponens alapú keretrendszer és törekszik arra, hogy ezek a komponensek külön fájlokban legyenek elhelyezve. Ebből az következik, hogy ahogyan projektünk növekszik, a fájlok száma is rohamosan növekedni fog. Ezért fontos egy konzisztens könyvtárstruktúrát felépíteni, hogy könnyen megtaláljuk azokat a fájlokat, amikre éppen szükségünk van. Szerencsére a Vue CLI nagyban megkönnyíti a munkánkat, mivel a megadott beállítások alapján létre tud hozni nekünk üres projekteket, egy alap struktúrával.



6.1. ábra. Frissen létrehozott Vue projekt

Ezek közül az *src* mappa lesz fontos, a *node_ modules* mappa az npm számára van ott, a különböző könyvtárak tárolására, a *public* mappa peddig azt a célt szolgálja, ha valamilyen oknál fogva a Vue az *assets* mappában lévő CSS-t, képeket, esetleg videókat nem tudja feldolgozni [27]. A *components* mappa, nevéből adódóan a komponenseinket tárolására hivatott. Az *App.vue* maga az a fájl, ami felcsatolásra kerül a HTML struktúrába, tehát ez mindig be van töltve. Az egyszerűség kedvéért, ide előnyös elhelyezni olyan elemeket, amiket mindig be szeretnénk tölteni, mint például a navigációs sávot, vagy a láblécet. A *main.js* pedig maga az a JavaScript fájl, ami megvalósítja a projekt működését, itt töltjük be a könyvtárakat, amiket használni szeretnénk és itt állítja össze a Vue a weboldalt.

Természetesen ahogy dolgozunk a projekten, ez a mappa is átmegy változásokon, ezek közül az első és talán a legfontosabb a *views* mappa bevezetése volt, hogy különséget tudjunk tenni azok között a komponensek között, amiket csak felhasználunk másokban és azok között, ami a felhasználó számára megjeleníti a tartalmat. Ezen kívül megjelentek a külső könyvtárak konfigurációit tartalmazó mappák is. Ez azért fontos, mert sok esetben az adott könyvtárak egyforma nevű alapfájlokat használnak, így el tudjuk kerülni a komplikációkat.



6.2. ábra. Vue fájlkezelési struktúra

7. Projekthez felhasznált könyvtárak

A Vue rengeteg beépített funkcióval rendelkezik, hogy megkönnyítse a munkánkat, viszont bizonyos helyeken kompromisszumokat kellett a készítőknek kötnie, hogy a keretrendszer méretét kordában tudják tartani. Szerencsénkre, mivel a Vue népszerűsége egyre csak növekszik, ezért szinte már minden nagyobb feladatkörre találunk könyvtárakat, amik rengeteg plusz lehetőséget kínálnak számunkra.
Ahhoz, hogy könyvtárat tudjunk használni, azt le kell töltenünk az általunk preferált csomagkezelő által, majd importálni és implementálni a weboldal építési szekvenciájába. Amennyiben szeretnénk, itt van lehetőségünk még további paramétereket

megadni a könyvtáraknak.

```
6   import axios from 'axios'
7   import VueAxios from 'vue-axios'
8   import useCookies  from 'vue3-cookies';
9   import Toast from "vue-toastification";
10
11  import './styles/tailwind.css';
12  import "vue-toastification/dist/index.css";
13
14  import { globalCookiesConfig } from "vue3-cookies";
15
16  axios.defaults.withCredentials = true
17  axios.defaults.baseURL = 'http://localhost:8000/';
18
19
20  //if our token expires we redirect to the login page
21  axios.interceptors.response.use(undefined, function (error) {···
31  })
32
33  createApp(App)
34      .use(i18n)
35      .use(router)
36      .use(store)
37      .use(VueAxios, axios)
38      .use(useCookies)
39      .use(Toast,
40      {transition: "Vue-Toastification__fade",
41      maxToasts: 20,
42      newestOnTop: true})
43      .mount("#app");
```

7.1. ábra. Vue könyvtárak

A fenti képen látható ahogy importáljuk az egyes könyvtárakat, majd a 16. sortól kezdve elkezdjük konfigurálni az axios-t, a *createApp* metódusban pedig megadjuk, hogy melyik könyvtárakat szeretnénk inicializálni.

7.1. Vue Router

A felhasználói élménynek egy nagyon fontos része, hogy hogyan jutunk el a weboldalunkon A pontból B-be. Az SPA megoldások, eleminálják ezeket a problémákat, mivel ahelyett, hogy minden új nézetnél újratöltsék az egész oldalt, csak a megfelelő tartalmakat cserélik ki, ezzel egy sokkal jobb felhasználói élményt nyújtva. A Vue Router megnevezést kapó könyvtár, amit a Vue csapata is hivatalosan támogat, pont ezt segít megvalósítani.
A Vue Router kihasználja a Vue által adott komponens készítési lehetőségeket és a hagyományos <**a**> elemeket használja fel, hogy a modern böngészők megértsék a HTML kódot.
Használatához, létre kell hoznunk egy *router* nevű mappát, abban pedig egy *index.js*

nevű fájlt. Itt egy **const** objektumban megadhatjuk a különböző "route"-okat. Ezek különböző felparaméterezhető objektumok, ahol a paraméterek segítségével adhatjuk meg, hogy milyen kérésre mit szeretnénk megjeleníteni [28]. Számunkra jelenleg az alábbiak a legfontosabb paraméterek.

- **path**: maga az URL, amit látunk a böngésző címsorában
- **name**: név, amivel hivatkozhatunk a "route"-ra. Nagy előnye, hogy így elkerülhetjük az URL-ek beleégetését kódunkba, ezzel jóval rugalmasabbá téve azt.
- **component**: megadja, hogy az adott címen milyen komponenst (vagy komponenseket, egyszerre több kezelésére is van lehetőség) szeretnénk az adott címen megjeleníteni
- **meta**: lehetőséget ad, hogy további meta adatokat adjunk tovább a weboldalnak. Mi jelenleg autentikációra használjuk.

```
const routes = [
  {
    path: "/",
    name: "App",
    component: Home,
    meta: {requiresAuth: true},
  },
  {
    path: "/login",
    name: "Login",
    component: Login,
    meta: {guest: true},
  },
```

7.2. ábra. Vue Router routes tömbje

A fenti képen látható, hogy az előbb leírták, hogyan néznek ki a kódban. Ezt a tömböt utána átadjuk a *createRouter* metódusnak, ami után legenerálja nekünk a szükséges objektumot, amit átadhatunk az alkalmazásnak.
Másik nagy előnye a Vue Router-nek, hogy megadhatók különböző navigációs figyelők, amik megakadályozzák, hogy a felhasználók olyan helyre jussanak, ahova nem szabadna [29]. Természetsen ez nem csak tiltásra használható. A projektben mi is elhelyeztünk egy olyan funkciót, ami a már bejelentkezett felhasználókat a bejelentkező és regisztrációs oldalakról átirányítja a kezdőoldalra.

7.2. Vuex

Mivel a Vue erősen építkezik a komponensekre, ezért felmerülhet bennünk egy kérdés, hogy mit csináljunk abban az esetben, ha bizonyos adatokra több nézetben is szükségünk van. Erre lehet egy megoldás, hogy minden nézetváltásnál betöltjük a szükséges adatokat. Nyilván egyből látható, hogy ez egyszerre lassítja a weboldalunkat, ezzel rontva a felhasználói élményt, és rengeteg felesleges kódmásolással is járna.
A Vuex ezt a feladatot oldja meg, mivel lehetőséget ad arra, hogy egy központi helyen tároljunk adatokat és manipuláljuk ezeket. Használatához létrehozunk egy *store* nevű mappát, amiben egy *index.js* fájlon keresztül elérhetővé tesszük a moduljainkat. Ezeket a modulokat négy további részre bonthatjuk.

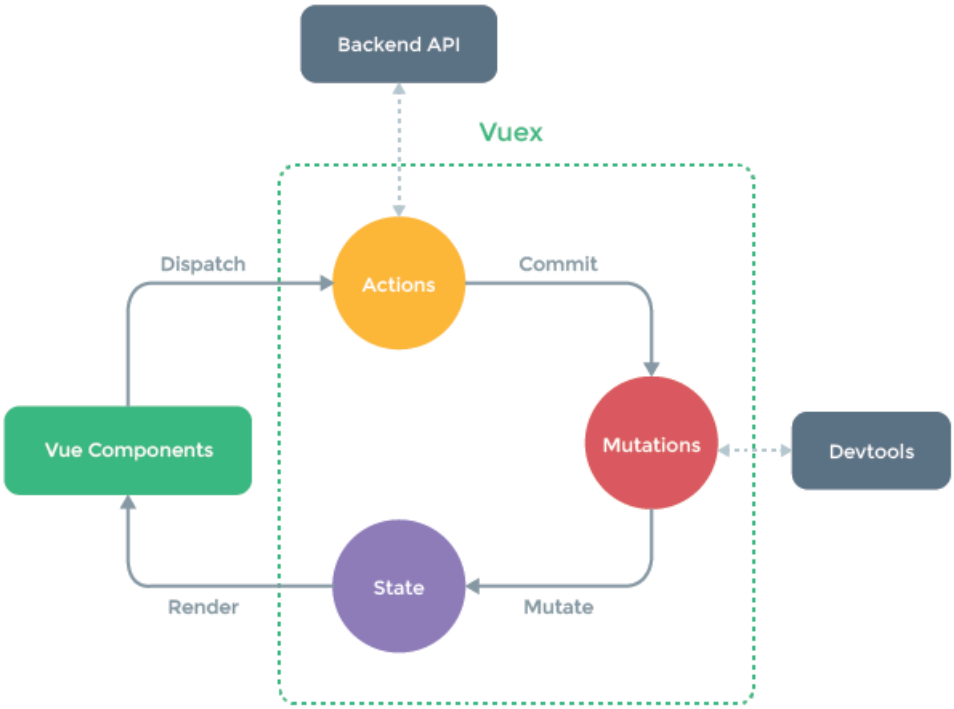
- **state**: egy tömb, amiben a szükséges adatainkat tároljuk
- **getters**: itt határozzuk meg, hogy milyen információk érhetőek el globálisan a programban
- **actions**: ebben a tömbben adjuk meg, hogy milyen műveleteket szeretnénk végezni a változóinkkal.
- **mutations**: ezek a műveletek változtatják a *state*-ek állapotát

Az adatokkal kétféleképpen tudunk dolgozni ezek után. Amikor először meglátogatjuk a weboldalt, a *state*-ben nincsenek adatok, ezeket a bejelentkezés után töltjük fel. Ezt úgy tudjuk megtenni, hogy az adott komponensbe importáljuk a *mapActions* segítmetódust, aminek segítségével a komponensünkön belül összekötünk egy metódust egy "action"-nel. Ez utána megvalósítja a "store"-ban lévő metódust. Ezek után pedig a változóinkat elérhetjük a *this.$store.getters* ponton keresztül a kódon belül bárhol. Az alábbi ábra, amit a Vuex dokumentációjában találhatunk remekül bemutatja, hogyan néz ki ez az egész folyamat.

A felhasználó elindítja a folyamatot (megnyom egy gombot, elküld egy formot, stb.). Erre a komponensben elsül egy metódus, ami elküldi a parancsot (esetenként paraméterekkel) a *store*-nak, ami kommunikációt végez az API-al, aminek eredménye alapján a mutációkat meg tudjuk hívni és módosításokat végezhetünk a *state*-ben lévő adattagokkal.

7.3. Axios

Ahhoz, hogy alkalmazásunk működjön, valamilyen módon kommunikációt kell létesítenünk a felhasználó cselekedetei és a szerver között. A JavaScript-ben létezik



7.3. ábra. Vuex state management folyamat

erre egy beépített modul, az *XMLHttpRequest* osztály. Ennek segítségével létrehozhatunk és felparaméterezhetünk kéréseket a szerver számára [30]. Az Axios egy népszerű könyvtár, ami erre az osztályra épít. Próbálták az *XMLHttpRequest*-ből keletkező kódot leegyszerűsíteni, de meghagyni a lehetőséget a részletes felparaméterezésre, miközben további funkciókkal látták el a könyvtárat. Ezek közül számunkra kettő fontos fejlesztés van, az első, hogy minden szervertől érkező választ automatikusan konvertál JSON-be, a másik pedig, hogy megvalósítja a Promise API-t [31]. Ez az API segít abban, hogy a JavaScript-ben való programozás közben az aszinkron kód visszatérései értékeit tudjuk úgy kezelni, mintha azok szinkron kódból érkéznének. Az aszinkron programozás lehetővé teszi számunkra, hogy kódunk egyszerre több szálon fusson, ezzel elkerülve azt, hogy az alkalmazás használhatatlan legyen amíg várunk a szerver válaszára [32].
A Promise API, nevéből is adódóan, egy ígéretet ad vissza a kódnak, hogy vagy a szervertől való választ, vagy egy hibaüzenetet mindenféleképpen szolgáltatni fog. Így az Axios-szal egyszerűen tudjuk kezelni a szerverrel való kommunikációt, mivel a kódban csupán annyit kell leírnunk, hogy mit tegyen a program, amint kapott választ a szervertől, és a szervertől érkező kódot kezelhetjük úgy, mint már futási

időben létező adatot.

7.4. I18n

Annak érdekében, hogy a projekt esetleges jövőbeli fejlesztéseit minél egyszerűbbé tegyük, már a kezdetektől fogva észben kellett tartania a csapatnak a lokalizációt, ha esetleg Magyarországon kívül is támogatást szeretnének nyújtani a programnak. Ezért ahelyett, hogy a kódba beleégették volna a szöveget, ezt a lokalizációs könyvtárat használják, amivel egy API segítségével mindig a kiválasztott nyelvnek megfelelő szöveget tudjuk visszaadni.

7.5. Vue Toastification

Toast-ok alatt webfejlesztésben, a kis beúszó ablakokat értjük, amit például akkor jelenítünk meg a felhasználó számára, ha vissza szeretnénk jelezni a szerverrel való kommunikáció eredményéről. Rengeteg könyvtár van, ami ehhez hasonló elemeket valósít meg, viszont legtöbbjükkel viszont az a probléma, hogy nagyon limitált a tartalom, amit elhelyezhetünk, vagy csak trükkös megoldásokkal tudjuk ezt megoldani. A Vue Toastification viszont lehetővé teszi, hogy egész Vue komponenseket adjunk át neki, ezzel hatalmas rugalmasságot biztosítva.

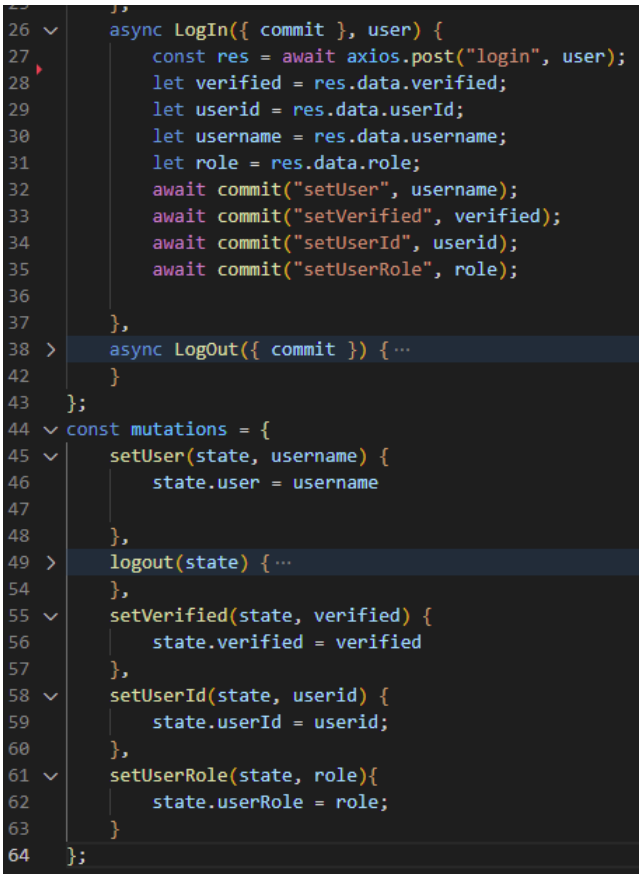
8. Autentikáció és autorizáció

- **Autentikáció**: Jelentése azonosítás. Ez a programban a bejelenetkezés közben történik meg, ahol elküldjük a szervernek a felhasználó bejelentkezési adatait, ami válaszol utána, hogy létező fiókról van-e szó és ha igen, akkor mik az adatai.
- **Autorizáció**: Jelentése hitelesítés. Miután a felhasználó bejelentkezett, a weboldal használata közben is biztosítanunk kell, hogy csak olyan műveleteket hajt végre, amihez van megfelelő jogosultsága [33].

Bejelentkezéskor a szerver, egy speciális tokent ad vissza számunkra, ezért további API hívások közben semmilyen felhasználói információt nem kell küldenünk, mivel a szerver ezt az adattagot fogja keresni a HTTP kérés fejlécében. Ehhez az Axios-ban külön be kell állítani a *withCredentials* paramétert, mivel a böngészőnk alapértelmezetten ezeket nem küldi el.
Miután megpróbálunk bejelentkezni, kapunk egy választ a szervertől, ami vagy a felhasználói adatokat tartalmazza, vagy egy visszajelzést, hogy nem sikerült a hitelesítés. Amennyiben sikeres a bejelentkezés, négy adatot tárolunk el:

- **Felhasználónév**
- **Fiók megerősítési státusza**
- **Fiók azonosító száma**
- **Szerepkör**

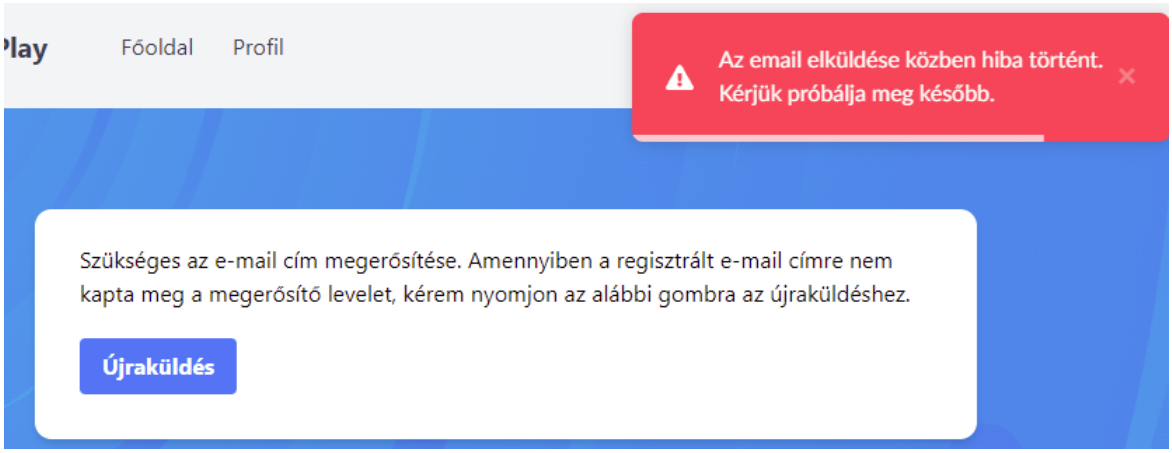
Ezek a Vuex segítségével kerülnek lokálisan elmentésre a már korábban ismertetett módszer alapján.



8.1. ábra. Bejelentkezés menete Vuex-ben

Megjegyzésként szeretném itt hozzátenni, hogy az oka annak, hogy minden adattag külön metódusban van beállítva, azért történik mert, amikor egy metódusban teszteltem, akkor a mentés nem működött konzisztensen (véletlenszerűen egyes adattagok nem kerültek mentésre).

Azelőtt, hogy a szoftvert használni tudjuk, szükséges e-mail címünk megerősítése. Legjobb esetben, a felhasználó ezt az e-mailt megkapja egyből a regisztrációt követően és el is fogadja. Amennyiben nem így tesz, egy validációs oldalra kerül, ahol lehetősége van a rendszerrel újra elküldetni ezt a levelet. A rendszer pedig ad egy visszajelzést, hogy a levél elküldése sikeres volt-e vagy sem. Ameddig ezt a felhasználó nem teszi meg, addig nem tud másik oldalra lépni, sem az oldalon lévő hivatkozások, sem a címsávban lévő URL segítségével.



8.2. ábra. Sikertelen visszaigazoló e-mail küldés belső szerver hiba miatt

Az AutiPlay-ben különböző felhasználói szerepköröket különböztetünk meg, ezek a: felhasználó, specialista és adminisztrátor. Ebből az okból kifolyólag fontos, hogy a beregisztrált tagok számára csak azt jelenítsük meg, amihez engedélyük van, ezzel elkerülve azt, hogy a szerver irányába felesleges kéréseket küldjünk, így csökkentve a terhelést. Az alábbi ábra bemutatja, hogy az egyes szerepköröknek jelenleg milyen műveletekhez van engedélye.

Felhasználó
Bejelentkezés
Kijelentkezés
Saját felhasználói adatok megváltoztatása
Jelszó megváltoztatása
E-mail megerősítése
Kérdőív kitöltése
Saját kérdőívek megtekintése

Specialista
Bejelentkezés
Kijelentkezés
Saját felhasználói adatok megváltoztatása
Jelszó megváltoztatása
E-mail megerősítése
Felhasználó kérdőíveinek megtekintése

Adminisztrátor
Bejelentkezés
Kijelentkezés
Saját felhasználói adatok megváltoztatása
Jelszó megváltoztatása
E-mail megerősítése
Felhasználó kérdőíveinek megtekintése
Más felhasználók adatainak megváltoztatása

8.3. ábra. Szerepkörök

Mint említettem, bejelentkezés során mentésre kerül a felhasználó szerepköre, ami segít a felhasználói felület kialakításában.
Ezt segítik számunkra a Vue-ba beépített direktívák, pontosan a *v-show* és a *v-if*. Mindkét direktíva akkor jelenít meg bizonyos HTML elemeket, ha a nekik átadott változó, *Boolean* kontextusban igazat ad vissza. A fő különbség a kettő között, hogy a *v-show*, a CSS *display* tulajdonságát használja, tehát attól függetlenül, hogy nem látható a felületen, a HTML kódban még megtalálható az adott elem, a *v-if*

pedig csak akkor helyezi el a kódban az elemet, ha az adott változó igaz. Legtöbb esetben az utóbbit használjuk, mivel így elkerülhető, hogy HTML manipuláció során a felhasználó olyan elemeket kezeljen, amihez nincs jogosultsága.

9. Adatvédelem

Ez a fejezet tekinthető az előző fejezet afféle magyarázatának. Míg a korábbiakban azt taglaltuk, hogy hogyan biztosítjuk, hogy az adott személy csak a számára szükséges adatokat érje el, itt ennek a miértjéről beszélek. Az internet fejlődésével rájöttek a vállalatok, hogy rengeteg ember szívesen megválik akár kényesebb személyi adataitól is, ha valamit az adott vállalat nyújt ért cserébe. Ezek közé tartozik például a születési dátum, teljes név, de akár lehet az adott ember vallása is. Később, ezzel az információval kereskedni lehet és ezek alapján célzott reklámokat mutatni az adott illetőnek, ezzel kiküszöbölve azt a problémát, hogy a hirdetés nem a célközönséget fogja elérni. Itt viszont fellép az a probléma, hogy még egyre több és több ember tölti fel adatait különböző weboldalakra, addig egy számítógépes rendszer sem tökéletes, így adatainkkal könnyen visszaélhetnek. Az "Európai Unió Alapjogi Chartája" megállapítja az EU-ban élő személyek adatvédelmi jogait[34], de ez korántsem volt elég részletes, ezért több év munkája után, 2016-ban, megalkották a világ legszigorúbb adatvédelmi szabályozás csomagját, a GDPR-t.
A GDPR (General Data Protection Regulation) célja az Európai Unióban élő polgárok adatainak védelme [35]. Amennyiben egy szervezet nem tartja be a szabályokat, akár több millió eurós bírságok kifizetésével is szembenézhetnek. A szabályzatban hét alapelvet határoznak meg, amik az alábbiak:

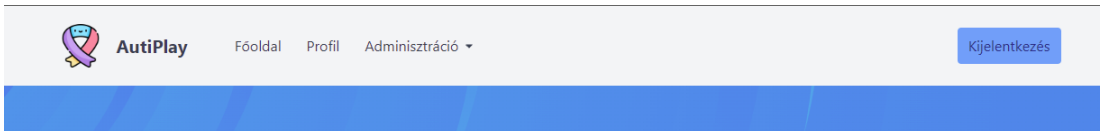
- **Törvényesség, méltányosság és átláthatóság**: A feldolgozásnak ezeknek meg kell felelnie.
- **Célkorlátozás**: Az adatokat csak olyanra lehet felhasználni, amihez az érintett korábban engedélyét adta.
- **Adat minimalizáció**: Csak szükséges adatok begyűjtése engedélyezett.
- **Pontosság**: A személyes adatoknak pontosnak és naprakésznek kell lenniük.
- **Tárolási korlátozás**: Adatot csak addig szabad tárolni amíg szükséges az adott célhoz.

- **Integritás és bizalmasság**: A feldolgozásnak olyan módon kell történnie, hogy biztosítva legyen a megfelelő biztonság, integritás és bizalmasság.
- **Számonkérhetőség**: Az adatokat felhasználó oldal felelős azért, hogy be tudja mutatni, hogy a GDPR szabályainak megfelelően jár el.

A szabályok ezen kívül azt is meghatározzák milyen esetben vagyunk jogosultak egy személy adatainak begyűjtésére. Mivel a szoftver célcsoportja az óvodás korban lévő gyermekek, ez külön komplikációkat is okozhat, azonban a törvénybe foglaltak szerint, amennyivel az érintett egyértelmű beleegyezését adja, az adatai szabadon felhasználhatóak, amennyiben tartjuk magunkat a fent foglalt hét alapelvhez.

10. Felület felépítése

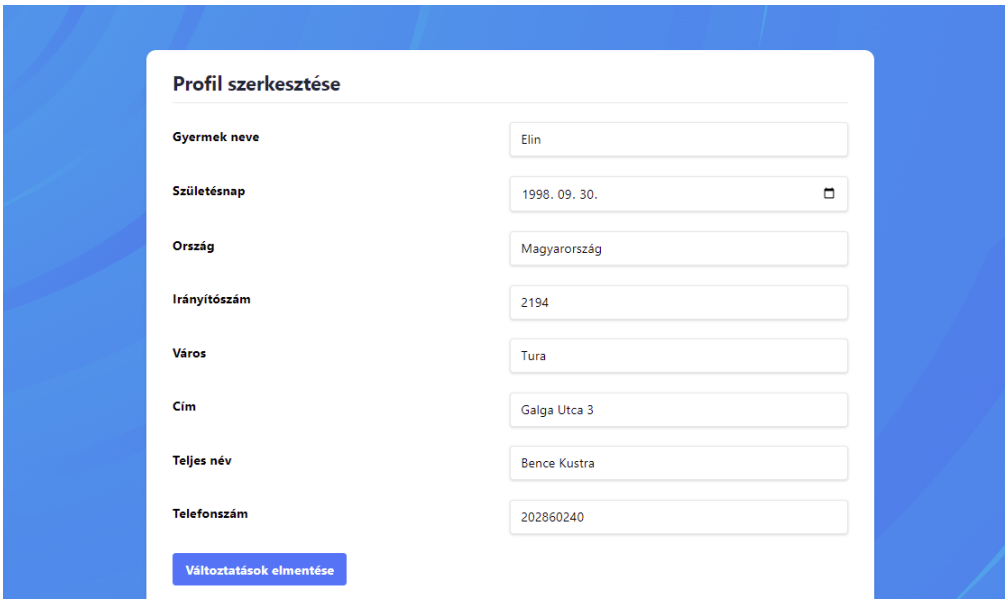
Amikor egy weboldalt tervezünk négy dolog van, amit mindenképp szeretnénk elérni. Gyors legyen, reszponzív, esztétikus és könnyen használható. Sajnos gyakran előfordul, hogy az egyik dolog a másik kárára megy. Például lehet egy nagyon szépen kinéző oldalunk, viszont, ha a felhasználónak emiatt nehezére esik megtalálni valamit, az csak rontani fog az élményen. Az AutiPlay-ben fontos szerepet kapott a könnyű navigáció, és az egyszerű megjelenés. Az elsőt az indokolja, hogy a weboldal célközönsége elég tágas, ezért nem tudjuk megbecsülni, hogy az átlag felhasználó milyen technikai tudással fog rendelkezni. A másodikat pedig az, hogy miközben a gyermek a teszt játékokkal játszik, nem szeretnénk, hogy bármi elterelje a figyelmét.



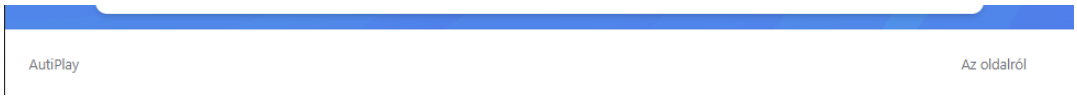
10.1. ábra. AutiPlay navigációs sáv

Az oldal tetején egy navigációs sáv található, amin elérhetőek a különböző nézetek és a bejelntkezés/kijelentkezés. Ez a sáv mindig a képernyő tetején helyezkedik el az egyszerű elérhetőség érdekében.

A legtöbb tartalom fehér kártyákon helyezkedik el, ami megfelelő kontrasztot ad a háttérhez. További előnye, hogy jól méretezhető, ami nagyban segíti a mobilos megjelenítést.



10.2. ábra. AutiPlay adatbeviteli lap



10.3. ábra. AutiPlay lábléc

Az oldal alján pedig egy lábléc lett elhelyezve, ahova később beilleszthetőek a kapcsolattartási információk és további linkek. Ez a lábléc minden esetben az oldal aljához van "ragasztva". Ennek az elérése a vártnál nehezebb feladat volt, mivel a Vue a weboldal tartalmát nem egyből a *body*-ban helyezi el, hanem egy azon belül elhelyezkedő *div*-ben. Ez a *div* valamiért nem tölti ki az egész képernyőt, csupán a megjelenítéshez szükséges minimális területet. A problémát a következő módon sikerült megoldani:

```
<template>
  <div id="app" class="flex flex-col h-screen">
    <NavBar
      ref="nav"
    />
      <router-view style="flex-grow: 1" class="w-full"/>
      <Footer/>
  </div>
</template>
```

10.4. ábra. AutiPlay elrendezés

Az egész felületet egy CSS Flexbox-ba helyeztük a *flex* osztállyal, ami gondosko-

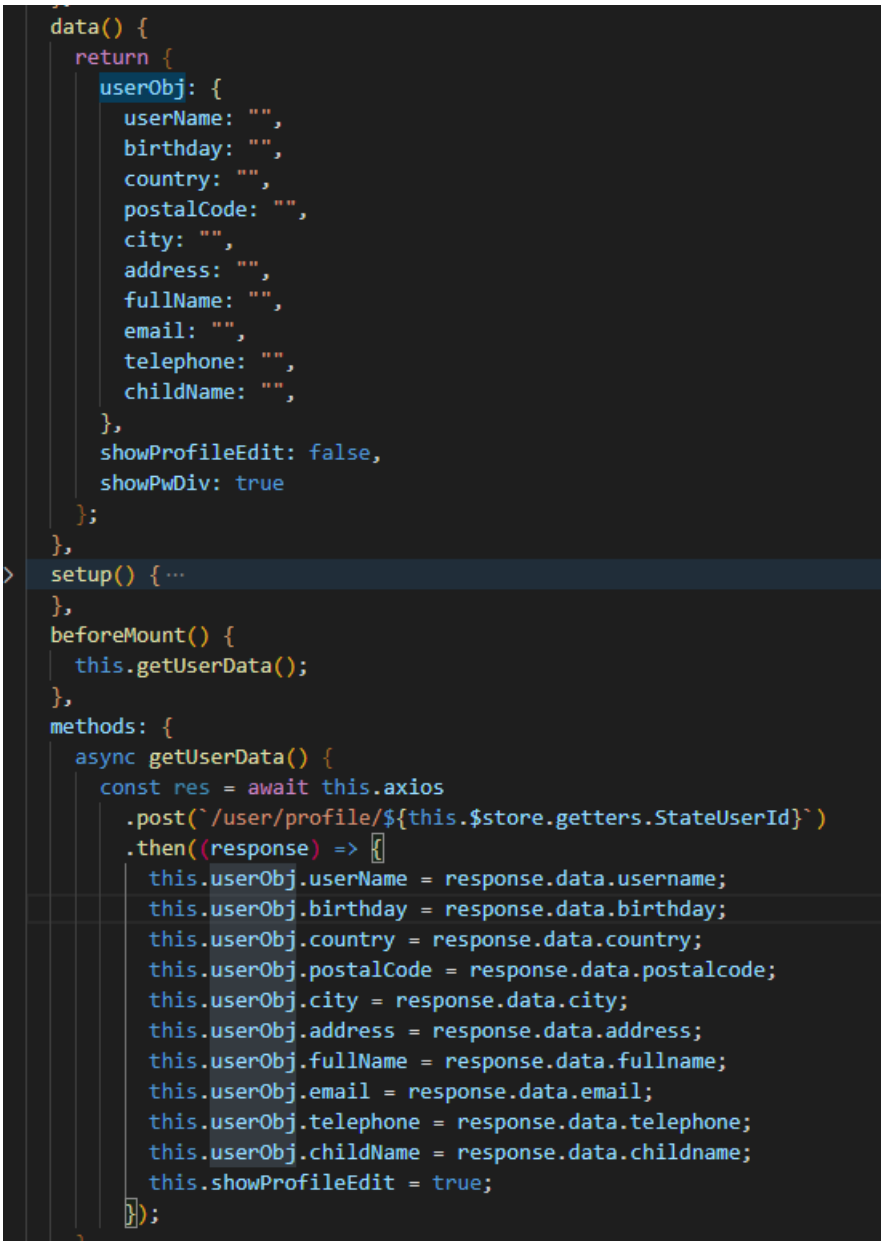
dik arról, hogy a benne elhelyezett elemek dinamikusan legyenek elhelyezve. Ahhoz, hogy elérjük a függőleges elrendezést a *flex-col* osztályt használtuk, ami megadja az ehhez szükséges paramétert. Ezek után jelentkezett egy nem várt mellékhatás, mivel a Flexbox alapértelmezetten az elemeket egyforma méretűként próbálja elhelyezni, ezzel a fő tartalmat teljesen középre tolva, amitől nagy hézagok keletkeztek. Emiatt lett megadva a *flex-grow* paraméter, aminek beállításával az oldal fő tartalma telíti ki az elérhető teret.

11. Adatmanipuláció

Egyik fő feladatomat képezte a projekten belül az adatmanipulációs folyamat kidolgozása. A rendszerben egyelőre két különböző féle adattal dolgozhatunk, a kérdőívekből kinyert és a profilokhoz kapcsolódó információval. Az alábbiakban ismertetni fogom a profilszerkesztésnek és a kérdőívek megtekintésének a folyamatát.

11.1. Profilszerkesztés

A Vue-nak nagy előnye a hagyományos JavaScript-tel szemben, hogy lehetőséget ad adattagok és HTML elemek közötti kétirányú adatkötésre a *v-model* direkítva segítségével, így nem szükséges külön az elemenkénti frissítéssel foglalkoznunk. Tervezés közben szerettem volna a Vue alapelveit követni és az egyes nézeten belüli részeket komponensekből megalkotni. Úgy döntöttem, hogy a nézet lesz felelős a logika megvalósításáért, a benne lévő komponensek pedig csak megjeneítési vagy interakciós céllal fognak rendelkezni. Ennek eléréséhez a nézet betöltés közben lekéri a szervertől a szükséges adatokat, majd amennyiben azokat megkapta, betölti a megjelenítési komponenst. Amennyiben a megjelenítési komponens valamilyen műveletet szeretne végrehajtani, ellő egy eseményt, amit a nézetben lekezelünk. Ez a következő módon valósul meg.



11.1. ábra. Profiladatok betöltése

Minden Vue komponens életciklusának különböző részei vannak, amik egyes pontokban elérhetőek, ezek között van a *beforeMount* is, ami a benne elhelyezett kódot, közvetlen a DOM-ban való elhelyezés előtt futtatja. Itt lekérjük a felhasználó adatait az azonosító száma alapján, majd ezeket a *userObj* nevű objektumban tároljuk. Közben a *showProfileEdit* nevű változót átállítjuk igazra, hogy jelezzük, hogy a betöltés befejeződött, a megjelenítésért felelős komponens betölthető.

11.2. ábra. Profiladatok megjelenítéséért felelős komponens

A következő módon átadjuk az adatokat tartalmazó objektumot, és a "@"-al kezdődő soroknál pedig megadjuk, hogy az adott eseményre melyik metódust kell elsütnünk a nézeten belül. Itt beleütköztem egy olyan komplikációba, ami sok kezdőnek okoz problémát a Vue-ban. Míg a komponensen belül a kétirányú adatkötés működik, ez már a komponensek közötti kommunikációra nem igaz. A dokumentációban leírtak szerint ezzel szeretnék megelőzni, hogy a leszármazott komponens véletlen módosítsa a szülő komponens állapotát. Így, ha valamilyen módon manipulálni szeretnénk a leszármazott komponensben a kapott adatokat és ezeket a manipulált adatokat visszaküldeni, ahhoz először ezekről egy másolatot kell csinálnunk, és azokat bekötni a *v-model* segítségével [36].

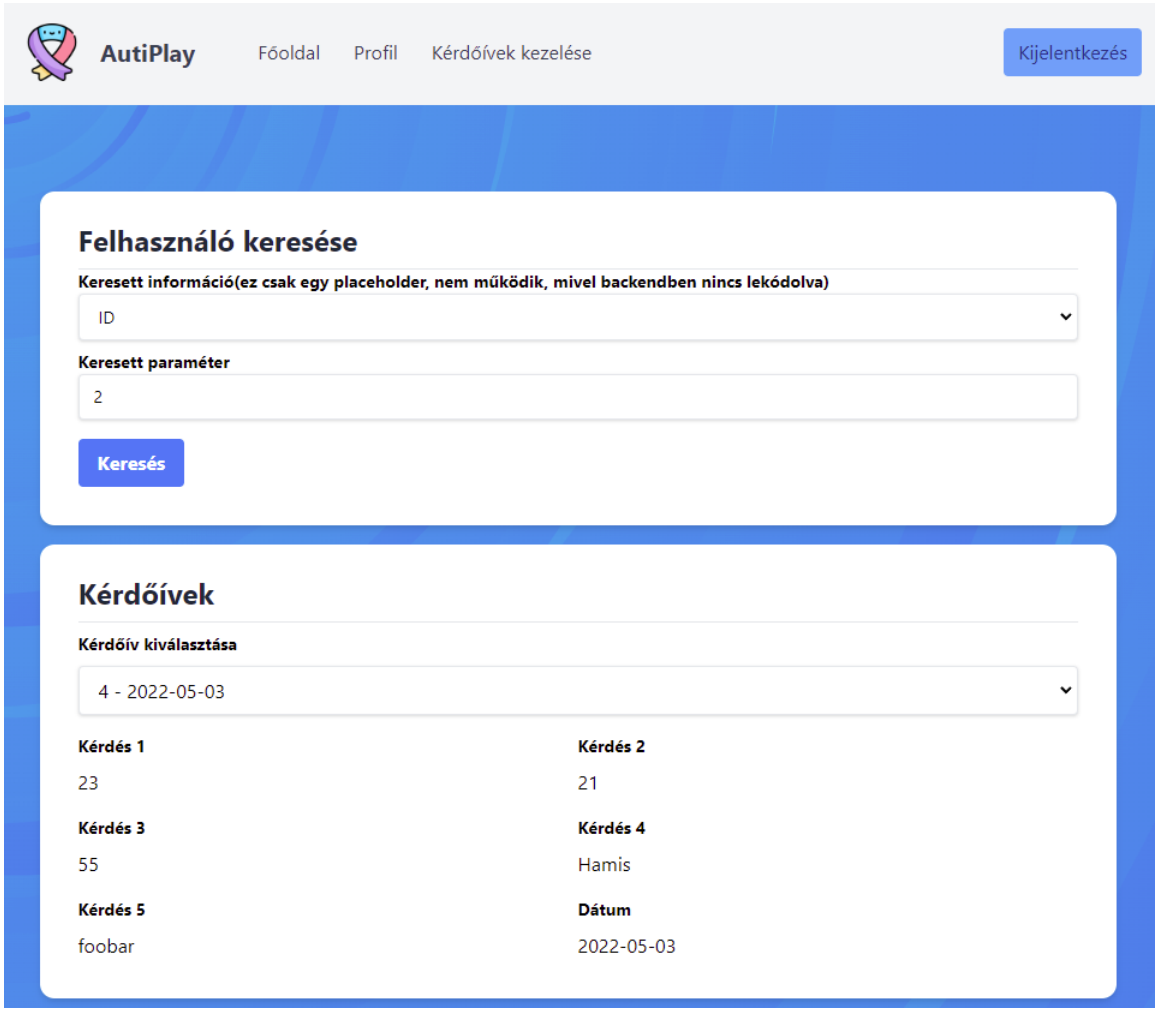
```
export default {
  props: {
    userObj: Object,
    showPwDiv: Boolean,
  },
  data() {
    return {
      userObjLocal: {
        username: this.userObj.userName,
        birthday: this.userObj.birthday,
        country: this.userObj.country,
        postalcode: this.userObj.postalCode,
        city: this.userObj.city,
        address: this.userObj.address,
        fullname: this.userObj.fullName,
        email: this.userObj.email,
        telephone: this.userObj.telephone,
        childname: this.userObj.childName,
      },
    };
  },
  methods: {
    saveChanges() {
      this.$emit("save-changes", this.userObjLocal);
    },
    saveNewPass() {
      this.$emit("save-new-pass");
    },
  },
};
```

11.3. ábra. Adatok beemelése a *props* metódus segítségével és másolása. Továbbá láthatóak az elsütött események

Amikor ezek az események elsülnek akkor a fő komponens indít egy frissítési kérelmet a szerver irányába az API segítségével majd a válasz alapján a Vue Toastification segítségével visszajelzést adunk a felhasználónak.

11.2. Kérdőívek megtekintése

A program fontos részét képzi a kérdőívek elemzése. Szerettem volna, hogy minél egyszerűbben és minél gyorsabban megtalálják a specialisták a keresett kérdőívet. Az tűnt a leglogikusabb keresési módszernek, ha először rákeresünk egy felhasználóra aztán pedig válogatni tudtunk az általa feltöltött kérdőívek közül. A felhasználói keresésnél érdemes lehet időt szánni az API kibővítésére, hogy ne csak ID alapú keresésre legyen lehetősége.



11.4. ábra. Kérdőívek megjelenítése

A későbbiekben ezen a felületen érdemes lehet erre az oldalra plusz interakciókat tenni, mint például felhasználó átirányítása más anyagokhoz, kommentek, vagy akár egy teljes kiértékelés írása.

12. Tesztelés

Szoftverünk tesztelése, ugyanolyan fontos, mint a jó kód írása. Viszont feljön a kérdés, hogy milyen módon teszteljünk, hiszen a probléma sok irányból megközelíthető. Egy ideig működhet a manuális módszer, ami közben a fejlesztett funkciót

kipróbáljuk, hogy az a tőlünk elvárt módon működik-e. Ez viszont gyorsan monotonná és feleslegesnek tűnőve válhat. Ezért miután egy kódrészlet elkészül, teszteket írunk, amiket futtathatunk minden új fejlesztés után, hogy ellenőrizzük korábban megírt kódunk épségét.

Az automatizált tesztelés legelterjedtebb formája a "Unit testing", ahol a kódot a lehető legapróbb funkcionális blokkjaira lebontjuk és azokat teszteljük [37]. A legelterjedtebb tesztelési minta az "AAA":

- **Arrange**: Ebben a részben készítjük elő a szükséges kódot. Adattagokat adunk meg, "mock"-okat készítünk (egyes funkciókat helyettesítünk, erről további információ az "AAA" minta után)
- **Act**: Kód végrehajtása
- **Arrange**: Kiértékeljük, hogy az elvégzett művelettől az elvárt eredményt kaptuk-e[38]

A "mock"-olás több okból kifolyólag is fontos lehet. Nagyobb funkciók helyetteítése esetén lecsökkenti a tesztelési időt és kevesebb erőforrást is vesz igénybe. Előfordulhat, hogy egy olyan tesztet írunk, ahol valamilyen külső erőforrással kellene kommunikálnunk, mint például egy adatbázis vagy egy API. Ez problámát jelenthet, mivel lehet, hogy az adott szerver nem elérhető, így a teszt külső okból kifolyólag nem teljesül. Ilyenkor a külső erőforrást helyetteítjük és úgy teszünk, mintha az elküldött kérésre megkaptuk volna a megfelelő választ. A legutolsó ok pedig, hogy nem mindig van lehetőség éles adatokon tesztelni, így jobb, ha azokat a kezdettől fogva helyetteítjük. A Vue beépített teszt könyvtára a *Vue test-utils* jól lefedi a teszteléssel kapcsolatos igényeinket. Lehetőséget ad komponensek egyéni vagy együttes tesztelésére és a jest npm könyvtár segítségével a "mock"-olás lehetőségét is megadja. Segítségével egyszerre tudjuk futtatni az egész tesztcsomagunkat, és amennyiben az egyes tesztek megfelelően vannak paraméterezve részletes leírást ad a hibákról, ha azok fellépnek. Mivel a Vue egyik fő célja az alacsony erőforrásigény, így sokszor külső könyvtárakat kell használnunk. A *Vue test-utils*-ban lehetőség van ezeknek a könyvtáraknak a teljes felhasználására, "mock"-olására, de akár arra is, hogy külön

a tesztkörnyezet szükségletei alapján felparaméterezzük őket.

13. Támogatott böngészők és rendszerkövetelmények

A megfelelő felhasználói élmény biztosításához még kettő tényező van, amit szükséges megvizsgálni. Az első a kompatibilitás, hogy biztosak legyünk, hogy a felhasználó azt látja, amit mi elvárunk, a másik pedig a rendszerkövetelmény, ahol azt vizsgálom, hogy potenciálisan milyen szerver kapacitásra lehet szükséges a weboldal "host"-olásához.

13.1. Támogatott böngészők

A támogatott böngészők listáját a Vue csapata pontosan meghatározta. Ebbe beletartozik minden böngésző, ami az ECMAScript 2015-ös szabványt használja [39]. Ez alól kivételt képez az Internet Explorer 11, aminél hosszas gondolkodás után úgy döntöttek, hogy nem fogják támogatni, mivel rengeteg munkát igényelt volna és a felhasználóbázis mérete kicsi és egyre csökken [40]. Az ECMAScript 2015 egy szabvány, ami meghatározza, hogy az adott böngészőmotor milyen módon valósítsa meg a JavaScript programozási nyelvet [41].

13.2. Rendszerkövetelmények

A szerver rendszerkövetelménye a fejlesztés ezen fázisában még csak becsülhető. Mivel a rendszer egyik fő komponense az arc mozgásának vizsgálata lesz így már most tudjuk, hogy lesznek esetek, ahol egyszerre fogunk rengeteg adatot kapni, amit el kell tárolni és adatbázisban rögzíteni vagy a videók miatt nagy adatcsomagokat küldeni. A játékoknál ez a kérdés nagyban függ az adott játék képi megjelenítésétől. A rendszerben lévő többi feladat alacsony erőforrás igénnyel rendelkezik, mivel csak

egyszerű HTTP kéréseket küld a felhasználó a szerver számára.

14. További fejlesztési lehetőségek

A munka során sikerült egy jó alapot kialakítani az AutiPlay-nek, viszont ahhoz, hogy egy igazán ütős programot készítsünk, még sok fejlesztést kell végeznünk. Ezekhez a backend és frontend csapat közös munkájára lesz szükség.
Felhasználók kezelésénél az egyszerűbb munka érdekében ideális lenne, ha több paraméterrel is tudnánk keresni, mivel jelenleg az API csak az azonosító szám alapján engedi lekérdezni a felhasználókat, ami egy valódi szituációban nem ideális. Ezen felül szeretnénk, ha egyszerre több, vagy az összes felhasználót is le tudnánk kérni, a könnyebb áttekinthetőség érdekében. Erre jól felhasználható lenne a JavaScript-es DataTables könyvtár, aminek a segítségével a hagyományos HTML táblázatokhoz tudunk rengeteg funkciót hozzáadni, mint például a dinamikus keresés, állítható megjelenítési méret, vagy a beépített szerveroldali kommunikáció.
A specialisták részére szeretnék készíteni egy dinamikus kérdőív tervezőt. Ez természetesen mind a megjelenítési oldalon és mind a szerver oldalon is kihívást jelent. Frontend-en dinamikusan kellene a felhasználó adatai alapján létrehozni az elemeket. Ebben viszont sokat segíteni a Vue komponens alapú természete, hiszen nekünk csak annak részleteit kellene külön beilleszteni a többi újra felhasználható lenne. Backend oldalon ez már egy kicsit komplikáltabb problémát vet fel, hiszen létre kell hozni egy olyan adatbázis megoldást, ami kezelni tudja a dinamikus kérdés és válasz mennyiséget, valamint a változó adattípusokat is. Szintén a specialisták számára készíthető lenne egy kérdőívek kiküldésére szolgáló oldal, ahol egyes felhasználók számára elküldhetnék, hogy melyik kérdőíveket szeretnék, hogy kitöltsék. Ez viszont felveti a kérdést, hogy mennyire szeretnénk óvni felhasználóink és gyermekeik identitását. Ennek a másik oldalán, szükséges lenne egy felület is, ahol kiértékelhetik

a kérdőíveket és feltölthetik megjegyzéseiket.

15. Értékelés

A dolgozat közben készült egy irodalmi áttekintés a piacon lévő fejlesztési és megjelenítési keretrendszerekről. Itt mérlegeltem az adott rendszereket és összehasonlítottam a teljesítményüket, a kód olvashatóságát, népszerűségüket és a dokumentáció minőségét. Ezek után a Vue-ra esett a választás, mivel mind a négy kategóriában jól teljesít. A fejlesztési keretrendszerek után megismételtem ezt a folyamatot a CSS keretrendszereknél is, ahol a Tailwind CSS teljesített a legjobban, az egyszerű tanulhatósága és a dinamikus CSS fájlméret miatt, mivel a könyvtár mindig csak az éppen szükséges dolgokat tölti be.
A dolgozat másik része maga a fejlesztés volt és az ott felhasznált technológiák dokumentálása. Kiválasztottam az adott szoftvereket, mint az IDE, verziókövető program, és szerverek futtatásához szükséges alkalmazások, majd írtam az adott szerverek közötti kapcsolatokról és ezen kapcsolatok létrehozásának nehézségeiről, ami legtöbb esetben a verzió különbségekből adódott, mivel a backend szerver a php egy régebbi változatára épül. Ez után bemutattam egy Vue projekt könyvtárstruktúráját. Mivel a keretrendszer nagy hangsúlyt fektett arra, hogy minden fálj, egy egyéni applikáció, így még fontosabb, hogy minden fájlt a neki megfelelő könyvtárban tároljunk, hiszen az oldalak növekedésével a fájlok száma is gyorsan fog növekedni. Ebből természetesen adódott a következő téma, a könyvtárak bemutatása. Mivel a Vue egyik alapelve, hogy a lehető legkisebb helyet foglalja, ezért sok feladatra külső könyvtárak alkalmazása szükséges, mint a Vue Router vagy a Vuex. Ezt követően részleteztem, hogy a szoftver biztonsági funkcióit hogyan oldottam meg a Vuex és a backend segítségével, majd részleteztem, hogy ez miért fontos, főleg, amikor Európában gyermekek adataival dolgozunk. Úgy érzem erre érdemes volt hangsúlyt fektetni, mivel a projekt elsődleges feladata a gyermekek segítése. A következő fejezetben részleteztem az oldal kinézetét, hogy milyen fő elemekből áll és hogy miért fontos az egyszerű megjelenés és a könnyű navigáció. Másik nagy hangsúlyt kapó feladat maga az adatmanipulációs folyamatok megtervezése volt, hogyan dolgozzuk fel a szervertől kapott adatokat, hogyan prezentáljuk azokat a felhasználónak, és hogyan továbbítjuk a tőlük kapott adatokat a szerver felé. Véleményem szerint sikerült a folyamatot jól megértenem és a megfelelő megoldásokat megtalálnom. Sajnos az API limitációi miatt nem sikerült annyi feladatot megvalósítanom,

amennyit szerettem volna, de úgy érzem, hogy sikerült egy olyan alapot felépíteni, amivel könnyen lehet dolgozni a jövőben. A dolgozat végén írtam egy keveset a tesztelés fontosságáról és annak menetéről a projektben, és ráébresztett arra, hogy csapatban dolgozás közben ez jóval nagyobb szerepet kap, mint amikor az ember egyedül dolgozik.
Összességében úgy érzem, hogy a szakdolgozat során egy jól működő, erős alapot sikerült fejleszteni az AutiPlay weboldalnak, amire a jövőben könnyű lesz építeni.

15.1. Értékelés (angol)

In this paper, I have done research regarding the development and CSS frameworks on the market. I've made comparisons based on performance, code readability, popularity and the quality of the documentation. Based on this research we chose Vue.js, since it performed well in all four categories. After the development frameworks, I've repeated this research on CSS frameworks, where Tailwind CSS was the clear winner based on it's friendly learning curve and dynamic file sizing.
The second part of the paper was about the actual development and the technologies used. I chose the appropriate software, like the IDE, version control program, and applications for running servers, then I wrote about how these servers connected to each other and the difficulties I faced during this process which mostly happened because of version incompatibility, since the backend server was built on an older version of php. After this, I presented the folder structure that Vue uses. Since the framework puts a large emphasis on each file being it's own application, it is even more important that each of them are stored in the appropriate folder, since as we create more and more pages, we'll get a lot more files as well. After this I wrote about the libraries used in the project. Since one of Vue's principles is taking the least possible space, many tasks are handled by external libraries like Vue Router or Vuex. After this I talked about the security solutions that were created with the help of the backend and Vuex, then I detailed the importance of security especially when we are working with the data of children in Europe. I felt this was important to specify since our goal is to help children. In the following chapter I talked about the looks of the website, the main elements and why simple presentation and easy navagation is important. The other big topic was the development of data manipulation processes, how to process information received from the server, how to present it to the user and how to send their data back to the server. In my opinion I have gained a good understanding of the process and I managed to use the correct solutions. Because of API limitations I didn't manage to create as many functions as I would've wished to, but I feel like I built a strong base on which it will be easy to build on in the future. At the end of the paper I've written about the importance

of testing and the process of it in this project.
In summary I feel like that during this paper, we managed to create a strong base for AutiPlay.

Irodalom

[1] B. Evans. „How autism became autism”, cím: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3757918/# (elérés dátuma 2022. 01. 05.).

[2] L. Réka. „Lehet könnyebb!”, cím: https://autispektrum.com/PDF/lehetkonnyebb.pdf (elérés dátuma 2022. 01. 05.).

[3] S. Khan. „What makes javascript so popular”, cím: https://generalassemb.ly/blog/what-makes-javascript-so-popular/ (elérés dátuma 2022. 01. 08.).

[4] R. C. Writer. „What is a scripting language and what are the most common ones”, cím: https://rockcontent.com/blog/scripting-languages/ (elérés dátuma 2022. 01. 08.).

[5] V. Singh. „What is a Framework? [Definition] Types of Frameworks”, cím: https://hackr.io/blog/what-is-frameworks (elérés dátuma 2022. 01. 11.).

[6] A. Bertram. „Declarative programming”, cím: https://searchitoperations.techtarget.com/definition/declarative-programming (elérés dátuma 2022. 01. 11.).

[7] W3schools. „React Components”, cím: https://www.w3schools.com/react/react_components.asp (elérés dátuma 2022. 01. 11.).

[8] R. Team. „Virtual DOM and Internals”, cím: https://reactjs.org/docs/faq-internals.html (elérés dátuma 2022. 01. 11.).

[9] N. Modi. „Angular vs AngularJS: Differences Between Angular and AngularJS”, cím: https://stackify.com/angular-vs-angularjs-differences-between-angular-and-angularjs/ (elérés dátuma 2022. 01. 13.).

[10] S. Richard és P. LePage. „What are Progressive Web Apps?”, cím: https://web.dev/what-are-pwas/ (elérés dátuma 2022. 01. 13.).

[11] A. Documentation. „Built-in directives”, cím: https://angular.io/guide/built-in-directives#ngModel (elérés dátuma 2022. 01. 13.).

[12] „Presentation of Vue.js”, cím: https://worldline.github.io/vuejs-training/presentation/ (elérés dátuma 2022. 01. 18.).

[13] „Single-File Components”, cím: https://vuejs.org/guide/scaling-up/sfc.html (elérés dátuma 2022. 01. 18.).

[14] „Build with the teams that never stop shipping”, cím: https://emberjs.com/ (elérés dátuma 2022. 01. 25.).

[15] „Reusable Components”, cím: https://guides.emberjs.com/release/tutorial/part-1/reusable-components/ (elérés dátuma 2022. 01. 25.).

[16] „The Ember CLI”, cím: https://cli.emberjs.com/release/ (elérés dátuma 2022. 01. 25.).

[17] „Why use Backbone, not [other framework X]”, cím: https://backbonejs.org/#FAQ-why-backbone (elérés dátuma 2022. 01. 25.).

[18] „Introducing JSON”, cím: https://www.json.org/json-en.html (elérés dátuma 2022. 01. 25.).

[19] L. Gupta. „What is REST”, cím: https://restfulapi.net/ (elérés dátuma 2022. 01. 25.).

[20] „Ember.js Router”, cím: https://www.geeksforgeeks.org/ember-js-router/ (elérés dátuma 2022. 01. 30.).

[21] „Twitter Bootstrap Usage Statistics”, cím: https://trends.builtwith.com/docinfo/Twitter-Bootstrap (elérés dátuma 2022. 02. 05.).

[22] J. Prabhu. „8 Reasons Why You Should Use Bootstrap”, cím: https://techaffinity.com/blog/why-use-bootstrap-for-frontend-design/ (elérés dátuma 2022. 02. 05.).

[23] A. Wathan. „CSS Utility Classes and "Separation of Concerns"”, cím: https://adamwathan.me/css-utility-classes-and-separation-of-concerns/ (elérés dátuma 2022. 02. 10.).

[24] „Sass: Sass Basics”, cím: https://sass-lang.com/guide (elérés dátuma 2022. 02. 14.).

[25] S. Bistrović. „Top 2021 CSS Frameworks Report, Part 1: The CSS File Sizes”, cím: https://css-auditors.com/reports/css-frameworks-part-1-2022-02/ (elérés dátuma 2022. 02. 20.).

[26] „Optimizing for Production”, cím: https://tailwindcss.com/docs/optimizing-for-production (elérés dátuma 2022. 02. 20.).

[27] „HTML and Static Assets | Vue CLI”, cím: https://cli.vuejs.org/guide/html-and-static-assets.html#the-public-folder (elérés dátuma 2022. 03. 19.).

[28] „Getting Started | Vue Router”, cím: https://router.vuejs.org/guide/ (elérés dátuma 2022. 02. 24.).

[29] „Navigaton Guards | Vue Router”, cím: https://router.vuejs.org/guide/advanced/navigation-guards.html (elérés dátuma 2022. 02. 27.).

[30] „Using XMLHttpRequest - Web APIs | MDN”, cím: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest (elérés dátuma 2022. 03. 05.).

[31] „Getting Started | Axios Docs”, cím: https://axios-http.com/docs/intro (elérés dátuma 2022. 03. 05.).

[32] „Aszinkron programozás | C# Tutorial.hu”, cím: https://csharptutorial.hu/docs/hellovilag-hellocsharp/10-tobbszalu-programozas/aszinkron-programozas// (elérés dátuma 2022. 03. 15.).

[33] „Mi a különbség az autorizáció és az autentikáció között? - ITZen tudástár”, cím: https://tudastar.itzen.hu/kb/mi-a-kulonbseg-az-autorizacio-es-az-autentikacio-kozott/ (elérés dátuma 2022. 03. 10.).

[34] „C_2012326HU.01039101.xml”, cím: https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:12012P/TXT&from=HU#d1e188-393-1 (elérés dátuma 2022. 03. 26.).

[35] B. Wolford. „What is GDPR, the EU's new data protection law? - GDPR.eu”, cím: https://gdpr.eu/what-is-gdpr/?cn-reloaded=1 (elérés dátuma 2022. 03. 26.).

[36] „Props |Vue.js”, cím: https://vuejs.org/guide/components/props.html#one-way-data-flow (elérés dátuma 2022. 04. 01.).

[37] K. Aebersold. „Software Testing Methodologies”, cím: https://smartbear.com/learn/automated-testing/software-testing-methodologies/ (elérés dátuma 2022. 04. 07.).

[38] „Unit testing fundamentals - Visual Studio (Windows) | Microsoft Docs”, cím: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022 (elérés dátuma 2022. 04. 07.).

[39] „Frequently Asked Questions | Vue.js”, cím: https://vuejs.org/about/faq.html#what-browsers-does-vue-support (elérés dátuma 2022. 05. 08.).

[40] „Proposal for dropping ie11 support in Vue 3”, cím: https://github.com/vuejs/rfcs/discussions/296 (elérés dátuma 2022. 05. 08.).

[41] „JavaScript ES6”, cím: https://www.programiz.com/javascript/ES6 (elérés dátuma 2022. 05. 08.).

Ábrák jegyzéke