

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Frontó András

Multiplatform alkalmazás fejlesztés React Native alapokon

Konzulens:

Benedek Zoltán

BUDAPEST, 2019

Tartalomjegyzék

Tartalom

[Bevezetés 3](#_Toc23105266)

[Technológiák feltérképezése 5](#_Toc23105267)

[1.1 React 5](#_Toc23105268)

[1.2 React Native 7](#_Toc23105269)

[1.3 React Native Web 9](#_Toc23105270)

[ReactXP 11](#_Toc23105271)

[1.5 Expo és React Native CLI 12](#_Toc23105273)

[1.6 Native Base és egyéb GUI könyvtárak 13](#_Toc23105274)

[1.7 Entity Framework és ASP.NET Core 14](#_Toc23105275)

[1.8 Axios 15](#_Toc23105276)

[1.9 SignalR 16](#_Toc23105277)

[Hivatkozások 18](#_Toc23105278)

# Bevezetés

Manapság már szinte mindenkinek van okostelefonja. A webalkalmazások fejlesztésénél épp ezért figyelembe kell venni ezt az egyre szélesebb körben használt platformot is. A mobil eszközöknél viszont számos más tényezőt is figyelembe kell venni amellett, hogy a kisebb kijelzőn megfelelő elrendezéseket és kényelmes felhasználói interfészeket alakítsunk ki. Az okostelefon a legtöbb ember életének szerves része, és az emberek szeretnek dedikált alkalmazásokat használni adott feladatokra (natív alkalmazásokat). Az ilyen alkalmazásoknak lényeges előnyei vannak a böngészőben betölthető weboldalakkal szemben. Ezen előnyök közé tartozik az is, hogy jobban integrálódnak a mobil felhasználási élménybe, és offline is használhatók, még ha csökkentett funkciókkal is. A natív alkalmazások fejlesztésénél abba a problémába ütközünk, hogy a piacot uraló két különböző operációs rendszer esetében (iOS és Android) a fejlesztés teljesen máshogy történik.

A fent vázolt problémákra az a megoldás tehát, hogy olyan webalkalmazást fejlesztünk, ami fut Androidon és iOS-en is natív alkalmazásként, és még a böngészőben is elindul. Az feladatom keretein belül ezt valósítom meg a React Native és a React Native Web segítségével. A React Native használata során csak JavaScriptet alkalmazva tudunk natív mobil alkalmazásokat fejleszteni. Erre épül a React Native Web, továbbá lehetőséget nyújt, hogy böngészőben is fusson az alkalmazás. Ezeknek a technológiáknak köszönhetően elég egy kódbázis natív alkalmazás készítéséhez iOS és Android platformokon. Ugyan ezt a kódbázist felhasználva, valamint kibővítve pedig egy desktopon is működő weblapot kapunk.

A dolgozatban a React Native-ot és a hozzá kapcsolódó technológiákat ismertetem. A feladat során sok multiplatform alkalmazások fejlesztéséhez készült, React Native alapú megoldással, keretrendszerrel és eszközzel találkoztam ezek lehetőségeit és gyengeségeit is tárgyalom a későbbiekben. A célom az volt, hogy a fent vázolt technológiák segítségével készítsek egy alkalmazást, mely amellett, hogy jól szemlélteti a közös kódbázis használatának előnyeit, rávilágít a nehézségekre is.

Az alkalmazásom a következő problémára próbál megoldást találni: az olyan környezetekben, ahol sok egymástól független csoport van különböző tudásterületek lefedésére, nem mindig egyértelmű, hogy ki tudhatja a választ egy szakértői kérdésre. Az alkalmazás így egyértelműen áll egy backendből, ami kezel egy tudásfát. Ez alapján a feltett kérdéseket képes azokhoz irányítani, akik azt a leghatékonyabban meg tudják válaszolni. A frontend mobilra készül React Native segítségével, mivel a mobil alkalmazás a felhasználók számára praktikusabb. Az így elkészült felhasználói felület minél nagyobb részének újra felhasználásával és a React Native Web által pedig egy desktopról is használható weboldalt készítettem. A weboldal a nagyobb képernyő miatt különböző elrendezésekkel és néhol kiegészítésekkel kényelmes alternatívát ad a praktikus hordozható mobil applikáció mellett.

Az alkalmazás legfontosabb funkciója a kérdések feltétele. A React Native, és a hozzá kapcsolódó könyvtárak rengeteg lehetőséget nyújtanak a kérdéseket tartalmazó nézetek, és a kérdések közti navigációra. Emellett a kérdések megválaszolására szolgáló valós idejű beszélgetés kialakítására is alkalmas. Az alkalmazás másik lényegi funkciója a tudásterületek kezelése és az ezekből felépített tudásfa megvalósítása. A bevitelhez és listázáshoz használt felhasználói felületek a React Native-os elemek segítségével egy teljesen natív alkalmazás élményét keltik. A legtöbb felület egy az egyben átemelhető webre a React Native Web segítségével a natív alkalmazásból. Néhol viszont a kényelmesebb felhasználói felület érdekében teljesen web specifikus a nagy képernyőre kitalált felületeket használtam.

A továbbiakban részletesen ismertetem, hogy milyen technológiákat használtam, illetve, hogy hogyan épül fel az alkalmazásom.

# Technológiák feltérképezése

Ebben a fejezetben az alkalmazásomhoz használt technológiákat fogom részletezni. Illetve azokat a React Native és React Native Web-hez kapcsolódó technológiákat, melyeket megvizsgáltam a munkám kutatási fázisában, de végül elvetettem valami okból.

## React

A React Native bemutatásához elengedhetetlen a React megismerése. A React egy a Facebook által fejlesztett JavaScript könyvtár felhasználói interfészek készítéséhez. A React könyvtár 2013-ban vált nyíltforráskódúvá.

A felhasználói felületet React esetén Component nevű elemekből épül fel. A Component-ek mellet szól, hogy így egy jól elkülönülő egységbe zárhatjuk magát a komponens viselkedését leíró logikát és a komponens kinézetéért felelős dizájnt. Minden Component újra felhasználható és egymásba ágyazható. A Component-ek rendelkeznek life cycle függvényekkel. Ezekkel lekezelhetjük például, amikor a UI elemek fel kerülnek a DOM-ra (Data Object Model), vagy valamilyen elemük frissül.

Minden Component ’props’ néven vesz be paramétereket amikor létrejön és van egy render függvénye, mely visszatér azzal, hogy hogyan is fog kinézni ez a komponens. Van egy ’state’ változója is minden komponensnek ez tárolja a komponens pillanatnyi állapotát és segít a dinamikus felhasználói felület kialakításában.

A React esetében a React komponens felépítését JSX segítségével írjuk le. A JSX a JavaScript XML-t rövidíti ez egy XML alapú kiegészítés JavaScripthez, ami a leginkább a HTML-hez hasonlít, de kapcsos zárójelek közt bármilyen JavaScript kifejezést bele lehet ágyazni.

Példa kód:

*//A Welcome egy React.Component*

*//A render függvénye határozza meg hogy hogyan fog kinézni*

class Welcome extends React.Component {

  render() {

*//Itt a JSX ami html szerű, de a kapcsos zárojelek közt*

*//javascriptet írunk, itt a komponens props-ból vesszük ki a név változót*

    return <h1>Hello, {this.props.name}</h1>;

  }

}

*//Elteszem az elementbe a Welcome komponenst*

*//A komponens létrejöttekor a name propsba a 'Sara' nevet adom*

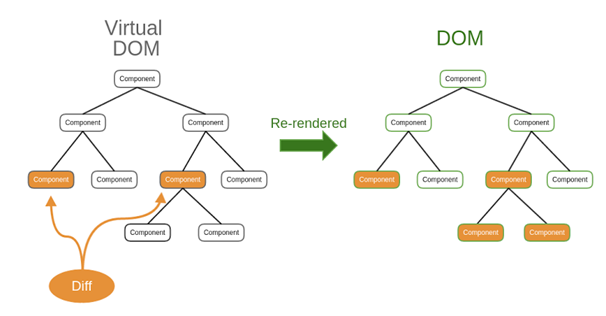
const element = <Welcome name="Sara" />;

*//Meghivoma  ReactDOM.render fuggvenyt az elementre*

*//Az oldalra kirajzolódik az element*

ReactDOM.render(element, document.getElementById("root"));

Ezen felül a Reactnak sok előnyös tulajdonsága van, amit sima HTML-lel nem lehetne megvalósítani. Például virtuális DOM segítségével kezeli a DOM-ot, aminek előnye, hogy a memóriában felépített cache segítségével csak a tényleg szükséges részeit frissíti a DOM-nak.

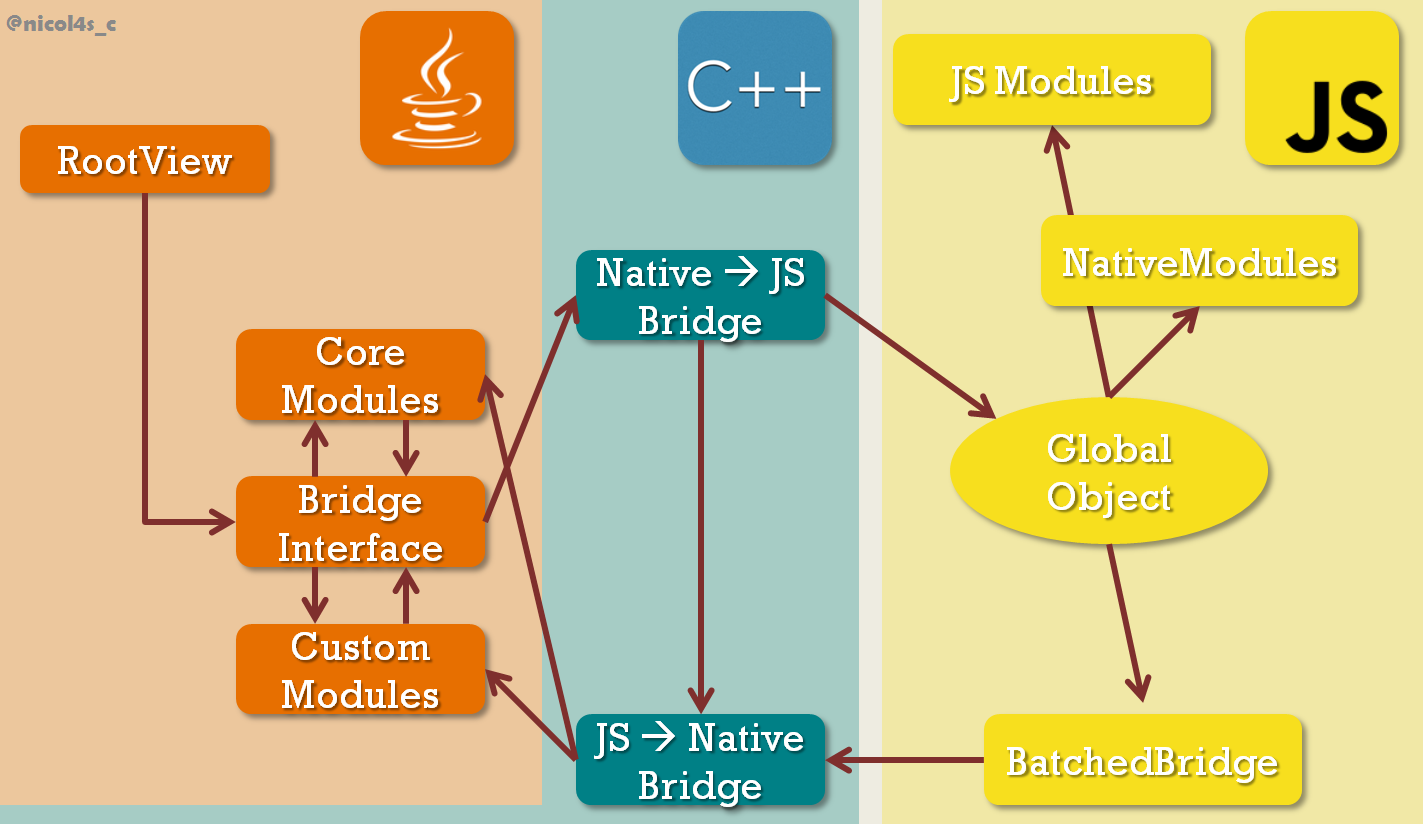


*Illusztráció a virtuális DOM különbség alapú újra renderelésére*

Összegezve: a React egy erős eszköz dinamikus weboldalak készítéséhez, csak JavaScript használatával.

## React Native

A React bemutatása után rátérhetünk magára a React Nativera ami a Reacthoz hasonlóan épül fel és hasonló koncepciókat foglal magában. A React Native szintén a Facebooktól eredő nyíltforráskódú könyvtár, amivel JavaScriptből lehet natív mobil alkalmazásokat építeni. A fő különbség a Reacthoz képest, hogy nem a DOM-ot manipulálja, hanem egy JavaScript értelmező segítségével az adott eszköz natív vezérlőivel kommunikál (natív module bridge).



*Illusztráció a React Native bridge-hez*

Ez lényeges különbség például a szintén népszerű Apache Cordovahoz képest, amivel szintén hibrid appokat lehet gyártani mobilra webalkalmazásból, de ott még közbeékelődik egy WebView is. Tehát React Nativenál például a navigáció, gombok Androidon a natív Androidos navigációt és gombokat fogják jelenteni, és ez iOS esetében is ugyanígy működik.

A natív module bridge használatának sok előnye van teljesítmény szempontjából, de lehetőséget ad arra is, hogy könnyen teljesen natív kódot adjunk az alkalmazáshoz. A teljesen natív kód természetesen feláldozza, hogy csak egyszer kell megírni a kódot iOS-re és Androidra is, de még gyorsabbá teszi a működést, és tovább növeli a teljesítményt, mert nem kell aggódni a JavaScript értelmező és a natív komponensek közti kommunikáció miatt.

A felhasználói felület itt is Component nevű elemekből épül fel. A Reactból jól ismert JSX itt is jelen van, de az alap építőelemek itt nem a HTML-ből ismerősek, hanem sajátos React Native elemek. Például Button, View, Text és még sok hasonló komponens.

Példa kód:

import React, { Component } from "react";

import { Text, View } from "react-native";

*//A HelloReactNative egy React komponens*

class HelloReactNative extends Component {

  render() {

    return (

*//Itt a JSX nem HTML elemeket használ hanem ReactNative modulokat*

      <View>

        <Text>If you like React, you'll also like React Native.</Text>

        <Text>

          Instead of 'div' and 'span', you'll use native components like

          'View' and 'Text'.

        </Text>

      </View>

    );

  }

}

export default HelloReactNative;

A komponensek felépítését JSX-ben írjuk le, a kinézetük, stílusuk definiálásához viszont a CSS-hez hasonló szerkezetű Flexbox alapú StyleSheet komponenst használjuk a React Native-ból.

Példa Kód:

import { StyleSheet } from "react-native";

const commonStyles = StyleSheet.create({

*//CSS-hez hasonló szintaxissal soroljuk fel a tulajdonságokat*

*//fontos, hogy a CSS-el szemben itt nincs öröklés és*

*//elemek megkeresése a DOM-ból. A StyleSheet elemet át kell adni*

*//a style propba annál az elemnél, akire használni szeretnénk a stílust.*

*//A stílusok listaként felsorolva össze fűződnek*

  errorTextStyle: {

*//a flexboxból ismert módszerekkel kezeljük a Layoutot*

    alignSelf: "center",

    color: "red",

    marginBottom: 5

  },

  commonText: { color: "#FFFFFF" }

});

*//Használat*

render = () => {

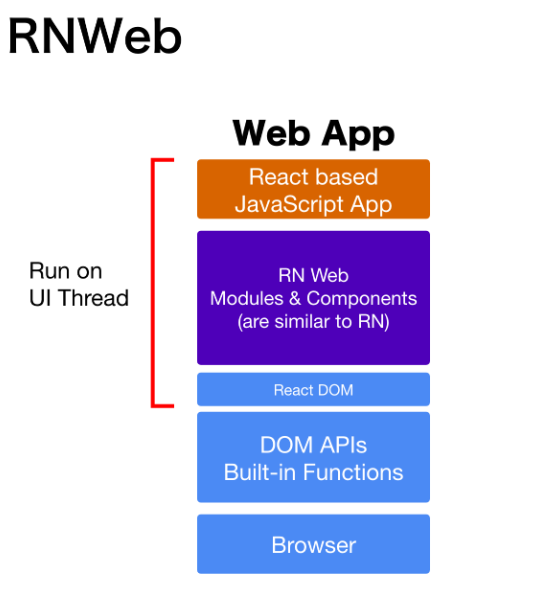
  return <Text style={commonStyles.commonText}>HelloWorld</Text>;

};

Összegezve: a React Native egy React alapokon működő natív alkalmazások fejlesztésére használt keretrendszer.

## React Native Web

A React Native esetében láttuk, hogy a legnagyobb különbség abban van, hogy hogyan épül fel a felhasználói felület, és ezzel hogyan kommunikál a JavaScript. A React Native ettől függetlenül erősen épül a webes Reactra, és a legtöbb elemük megfeleltethető egymásnak. Szeretnénk olyan web alkalmazásokat írni, melyek nem csak natív-an működnek mobilokon de böngészőben is futnak. A React Native Web teszi lehetővé, hogy teljesen React Native kódot írjunk az ott használt vezérlőkkel, komponensekkel, és egy böngészőben futó weboldalt kapjunk. A React Native Web ezt úgy éri el, hogy a React Native komponenseknek megfelelő saját komponenseket definiál, melyek a react DOM-mal kommunikálnak a bridge helyett.



*Illusztráció a React Native Web működési modelljéhez*

A web specifikus megfelelőit a React Native elemeknek a webpack.config.js-ben kell beállítani. Ez alapján a program fordítási időben, ha webre fordít a React Native Web-et használja, ha pedig mobilra akkor a React Native-ot.

Részlet a konfigurációból:

alias: {

*// Support React Native Web*

  'react-native': 'react-native-web',

  'native-base': 'native-base-web',

  'react/lib/ReactNativePropRegistry':

    'react-native-web/dist/modules/ReactNativePropRegistry'

},

A kódbázis nagyrészét újra felhasználjuk a React Native Web segítségével, de van, hogy platform specifikus kódra van szükségünk. Erre két megoldás létezik, React Native és React Native Web használatakor. Futási időben egy if elágazásban megvizsgálhatjuk, hogy milyen platformon vagyunk es annak megfelelően hajthatunk végre kódot.

Példa kód:

import { Platform } from "react-native";

import env from "../../env";

*//Itt a React Native Platform osztályát használom*

*//amivel letudom kérni a futtató környezet operációsrendszerét*

if (Platform.OS === "android") {

  this.server = env.ServerUrlForAndroid;

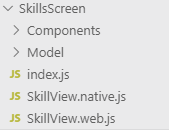
  this.props.setTitle("Skills");

} else {

  this.server = env.ServerUrlForWeb;

}

Alternatív megoldás, ha egész fájlokat akarunk csak egyik vagy másik platformon használni megadhatjuk a nevükben, hogy a natív vagy a web alkalmazáshoz tartozzanak (Android illetve iOS platform specifikus fájlokat is lehet így megjelölni). Fordítási időben dől el, hogy melyik fájlt használjuk. Ezt a módszert akkor érdemes használni, ha például ugyanannak a komponensnek teljesen más kinézetet akarunk adni weben, illetve natív környezetben.



*Kép a .native és .web jelölésű azonos nevű fájlokról*

Összegezve: a React Native Web egy hatékony eszköz az alkalmazásunk weblappá alakítására.

## ReactXP

A ReactXP egy a Microsoft által fejlesztett keretrendszer. Egy absztrakciós szint a React Native felett, azt a problémát hivatott megoldani, hogy bár a React Native képes natív alkalmazást fejleszteni minden platformra, de a View definíciók, stílusok és animációk mind platform függőek, hiszen a natív vezérlőket használja. ReactXP használatával ezek mind egységessé vállnak. Biztosítja ezt az egységes UI felületet React alapokon webre is. Ezen felül tervben van, hogy az UWP-re (Universal Windows Platform) is lehet majd fejleszteni ezen az absztrakciós szinten. A ReactXP fejlesztés TypeScript segítségével történik. A ReactXP majdnem teljesen ugyan úgy működik, mint a React Native apró eltérések vannak csak.

Példa kód:

import React from "react";

import RX from "reactxp";

interface AppProps {

  userName?: string;

}

*//Különbség, hogy nem egy React Native-os StyleSheet-et használ*

*//hanem több külön álló ReactXP stílust, így oldja meg,*

*//hogy egységesek legyenek a stílusok minden platformra*

const \_styles = {

  main\_container: RX.Styles.createViewStyle({

    justifyContent: "center",

    alignItems: "center"

  }),

  text: RX.Styles.createTextStyle({

    color: "red",

    fontWeight: "bold"

  })

};

*//A render és minden más ugyan úgy működik, mint React Native esetében,*

*//de a React Native komponensek helyett*

*//a rájuk épülő ReactXP elemeket használjuk*

export class App extends RX.Component<AppProps, void> {

  render() {

    return (

      <RX.View style={\_styles.main\_container}>

        <RX.Text style={\_styles.text}>Hello World</RX.Text>

      </RX.View>

    );

  }

}

A ReactXP, bár nagyon sok mindent leegyszerűsít és nagy segítség a felhasználói felületek egységesítésében, de megvannak a maga hátrányai. Azért, hogy az egységes kinézetet elérje csak a legfontosabb és minden platformon könnyen egységesíthető vezérlőket valósítja csak meg. Amennyiben ezek az általánosan használt komponensek elegek lennének az alkalmazás elkészítéséhez ez egy nagyon hatékony megközelítés.

Az alkalmazás fejlesztésének korai fázisaiban kiderült, hogy ezek miatt a korlátozott funkcionalitások miatt sok más külsős könyvtárral nehezen működik együtt. A kompatibilitási problémák természetesen feloldhatók megfelelő csomagolók elkészítésével, tehát ezzel a technológiával is megoldható lenne a feladat. Azért nem ezt választottam végül, mert a dokumentáció nem teszi egyértelművé mindig mi az, amit meglehet oldani ReactXP-vel és mi az, aminél vissza kell térni a React Native gyökerekhez. A csomagolók írása miatt pedig inkább bonyolította a feladatot, mint egyszerűsítette. Ezért maradtam az alap React Native mellet, mely jóval flexibilisebb a sok hozzá írt külső komponens miatt.

## Expo és React Native CLI

A React Native alkalmazások build-jének a legegyszerűbb és legteljesebb módja az Expo használata. A hivatalos React Native dokumentációban is az Expo használatát javasolják. Az Expo 2 workflow-t támogat. Az egyik esetében a cél eszközöket teljesen az ExpoSDK-n keresztül érjük el. Ekkor nem kell egyáltalán a platform specifikus eszközökkel foglalkozni ( AndroidStudio és XCode ). A másik esetben a build sokkal inkább hasonlít az eredeti React Native CLI-al is elérhető buildre. Megkapjuk a platform specifikus projekteket és teljes kontrollunk van ezek felett, de a build ekkor is támogatva és gyorsítva van az ExpoSDK-val.

Az Expo a build gyorsítása mellett, a cél eszközök kezelésében is segít. Egyik nagy előnye például az asset-ek kezelése. Az AndroidStudio-ban is látott módon az eszköz képernyő DPI-nak megfelelő erőforrásokat tölt be például. Kezeli a nyelvi beállításokat, fontokat és minden mást, ami az erőforrásokhoz tartozik.

Bár ez a legteljesebb build tool ami rendelkezésre áll a React Native alkalmazások készítéséhez végül én az eredeti React Native CLI használata mellett döntöttem. Ennek oka, hogy az Expo bár bizonyos szintig képes együtt működni az általam használt React Native Web-bel, de nem támogatja azt és a közeljövőben nem is tervezik támogatni.

Az alkalmazás fejlesztése során a React Native Web és az Expo platform specifikus font kezelése hibákhoz vezetett, ezért az Expo helyett a jóval egyszerűbb React Native CLI használatát választottam. Ez leginkább az Expo nem manegelt workflowjára hasonlít. Teljes kontrollt enged a platform specifikus projectek felett, de nem rendelkezik az Expo nyújtotta előnyökkel. Ez a legegyszerűbb build rendszer React Native-hoz.

Ezeken felül még használhattam volna a CRNWA (Create React Native Web App) build rendszert, de ennek feltérképezésére már nem jutott idő.

## Native Base és egyéb GUI könyvtárak

A felhasználói felület építésére már a React Native is sok eszközt nyújt, de pár komplikáltabb vezérlőhöz és egy egységes téma eléréséhez külső könyvtárakat használtam. Találni egy olyan komponens könyvtárat, mely teljes mértékben működik weben és natív alkalmazásokban is viszont nem könnyű.

A feladatom során több ilyen könyvtárat is kipróbáltam. A react-native-material-ui a webes világból jól ismert Material Theme-et adja hozzá a natív alkalmazásunkhoz. A React Native Web-bel viszont nem működött együtt. A React Native Web komponensei bár nagyon hasonlítanak a React Native komponensekhez ezért többnyire egy az egyben megfeleltethetők egymásnak, de a prop-okat nem teljesen egyformán kezelik és emiatt ez a komponens könyvtár nem működött.

A második próbálkozás a React Native Elemnts volt. Ezt elvileg össze lehetne kombinálni a React Native Webbel ehhez még egy tutarial is van. Ehhez viszont a CRNWA build eszközt kéne használnom, melynek megértésére nem jutott idő és nem emellett döntöttem végül. A megfelelő build rendszer nélkül viszont sajnos nem sikerült működésre bírni.

Harmadik próbálkozásként a Nachos UI-al próbálkoztam, mely külön a React Native Webhez készült, ez működött weben és natívan is. Az volt az egyetlen gond, hogy a natív alkalmazásban nagyon specifikus stílus szabályok nélkül teljesen szét esett a felület főleg mikor felület változás történt. Ez idővel nagyon frusztrálóvá vált tehát ezt is elvetettem.

A képen monitor, képernyőkép, képernyő, mobiltelefon látható

Automatikusan generált leírásA negyedik és egyben utolsó próbálkozásom a Native Base komponens könyvtárral volt. Ennek a könyvtárnak van egy külön webes forkja a Native Base Web. Ez teljesen együtt működik a React Native Web-bel. A webpack.config.js-ben ezt is mappelni kell a React Native – React Native Web mappeléshez hasonlóan. Ez a komponens könyvtár egységes gomb stílusokat, Card elemeket, interaktív listákat és sok hasznos vezérlőt tartalmaz.

A képen képernyőkép látható

Automatikusan generált leírás

*A képen a kérdéseket tartalmazó Card komponensek listája látható mobilon és weben*

Ennek a komponens könyvtárnak az egyetlen komolyabb hátránya, hogy nem minden natívon működő komponens része a webes forknak ezért az egységes kinézet kialakítása néha nehezebb.

## Entity Framework és ASP.NET Core

Mivel egy webalkalmazásról beszélünk mindenképpen szükség volt egy backendre. Az ASP.NET Core nyílt forráskódú webes keretrendszert használtam fel, ami cross platform backendek készítésére képes. Ez egy moduláris keretrendszer, ami a backend minden funkcionalitását lefedi. Például a több felhasználós rendszerhez elengedhetetlen azonosítást az ASP.NET Identity segítségével valósítottam meg.

Az adatok kezelését a Microsoft ORM (Object-relational mapping) keretrendszerével az Entity Framework segítségével oldottam meg.

## Axios

Az alkalmazás nagy része HTTP segítségével kommunikál a backend és a frontend között. Backend oldalról ezt az ASP.NET Core-val készített kontrollerek tökéletesen kiszolgálják. Kliensoldalról viszont az Axios promise alapú HTTP kliens könyvtárat használom. Ezzel a könyvtárral könnyen kezelhető http kéréseket lehet összeállítani. A kérések egy promis-val térnek vissza, ami lekezelhető mikor megérkezik a http válasz.

Példa kód:

import axios from "axios";

deleteSkill = id => {

*//Beállítom a JWT tokent a kéréshez*

  const headers = {

    Authorization: "Bearer " + this.props.jwt

  };

  this.View.current.setState({

    error: ""

  });

*//Felépítek egy HTTP delete kérést a megfelelő headerrel*

  axios({

    method: "DELETE",

    url: "http://" + this.server + `:5000/api/skills/${id}`,

    headers: headers

  })

*//Feloldom a Promiset ha visszatér*

    .then(response => {

      console.log(response);

    })

*//Hiba esetén lekezelem és lekezelem a hibát*

    .catch(error => {

      console.log(error);

      this.View.current.setState({

        error: error.response.data.error,

        loading: false

      });

    });

};

A JavaScript-ben alapértelmezett fetch-el szemben számos előnye van az Axios-nak. Az Axios az adatokat, amiket küldünk és visszakapunk automatikusan alakítja a kérésnek megfelelő formátumra, valamint vissza JSON-be. Axios-ban könnyen állítható időkorlát a kérésekre, aminek meghaladása után az Axios eldobja a kérést. Emellett képes szimultán kérések indítására is.

Összegezve: az Axios egy jól kezelhető és kényelmes könyvtár HTTP kérések indítására és HTTP válaszok kezelésére és mindenképp előre lépés az egyszerű fetch-vel szemben.

## SignalR

A HTTP API mellett a valós idejű kommunikációt is folytatok a szerverrel. Például a feltett kérdésekre adott válaszok egy valós idejű chatben jelennek meg a felhasználóknál. Ezt a SignalR segítségével oldom meg. A SignalR egy a Microsoft által fejlesztett ASP.NET keretrendszerre épülő szerveroldali megoldás, mellyel hatékony kétoldalú aszinkron kommunikáció valósítható meg.

Alapvetően Websocket segítségével valósít meg két irányú kommunikációt a szerver és a kliens között. Amennyiben a Websocket nem érhető el a SignalR egyéb alternatív kommunikációs rétegen keresztül továbbít üzeneteket. Ezt viszont mind elfedi a függvénykönyvtár és megvalósításának köszönhetően skálázható környezetben is elhelyezhető.

C# példa kód:

using Microsoft.AspNetCore.SignalR;

*//A MessageHub egy SignalR Hub leszármazott*

public class MessagesHub: Hub

{

*//Amikor üzenetet küldök ezt a metódust hívom a kliens oldalról*

    public async Task SendMessage(string message)

    {

*//Az össze kliensen elsütöm a 'ReciveMessage' eseményt*

*//és visszaadom nekik a messaget*

        await Clients.All.SendAsync("ReceiveMessage",message);

    }

}

A kliens oldalon az @aspnet/signalr könyvtár segítségével kezelem le a két irányú kapcsolatot.

JavaScript példa kód:

import \* as signalR from "@aspnet/signalr";

*//Megadom a kapcsolat paramétereit a szerverrel*

var connection = new signalR.HubConnectionBuilder()

  .withUrl("http://" + this.server + ":5000/api/thread")

  .build();

*//Ha elsüti a szerver a ReceiveMessage eseményt kiírom az üzenetet*

connection.on("ReceiveMessage", message => {

  console.log(message);

});

*//Felépítek egy kapcsolatot a szerverrel*

connection.start();

*//Definiálok egy függvényt arra, hogy hogyan küldök üzenetet*

sendMessage = message => {

*//Elsütöm a szerver SendMessage eseményét*

  this.connection.invoke("SendMessage", message).catch(function(err) {

    return console.error(err.toString());

  });

};

*//Elküldöm a szervernek, hogy 'Hello World'*

senndMessage("Hello World");

# Hivatkozások

|  |  |
| --- | --- |
| [1] | K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062. |
| [2] | P. Koopman, „How to Write an Abstract,” október 1997. [Online]. Available: https://users.ece.cmu.edu/~koopman/essays/abstract.html. [Hozzáférés dátuma: 20 október 2015]. |
| [3] | W3C, „HTML, The Web’s Core Language,” [Online]. Available: http://www.w3.org/html/. [Hozzáférés dátuma: 20 október 2015]. |
| [4] | [Online]. Available: http://www.reactnative.com/react-native-dom/. |
| [5] | [Online]. Available: https://cordova.apache.org/docs/en/9.x/guide/overview/index.html#page-toc-source. |
| [6] | [Online]. Available: https://en.wikipedia.org/wiki/React\_Native. |
| [7] | [Online]. Available: https://en.wikipedia.org/wiki/React\_(JavaScript\_library). |
| [8] | [Online]. Available: https://www.w3schools.com/whatis/whatis\_react.asp. |
| [9] | [Online]. Available: https://reactjs.org/. |
| [10] | „ReactComponent,” [Online]. Available: https://reactjs.org/docs/components-and-props.html. |
| [11] | „ReactVirtualDom,” [Online]. Available: https://jelvix.com/blog/is-react-js-fast. |
| [12] | „ReactNativeBridge,” [Online]. Available: https://levelup.gitconnected.com/wait-what-happens-when-my-react-native-application-starts-an-in-depth-look-inside-react-native-5f306ef3250f. |
| [13] | „ReactXP,” [Online]. Available: https://microsoft.github.io/reactxp/docs/getting-started.html. |
| [14] | „ExpoSDKInWork,” [Online]. Available: https://docs.expo.io/versions/latest/workflow/how-expo-works/. |
| [15] | „Expo,” [Online]. Available: https://docs.expo.io/versions/latest/. |
| [16] | „SignalR,” [Online]. Available: https://hu.wikipedia.org/wiki/SignalR. |
| [17] | „ASP.NET\_Core,” [Online]. Available: https://en.wikipedia.org/wiki/ASP.NET\_Core. |
| [18] | „Axios,” [Online]. Available: https://github.com/axios/axios. |
| [19] | „Axios vs Fetch,” [Online]. Available: https://blog.logrocket.com/axios-or-fetch-api/. |