FELADATKIÍRÁS



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Frontó András

Multiplatform alkalmazás fejlesztés React Native alapokon

Konzulens:

Benedek Zoltán

BUDAPEST, 2019

Tartalomjegyzék

[Összefoglaló 6](#_Toc26720831)

[Abstract 7](#_Toc26720832)

[1 Bevezetés 8](#_Toc26720833)

[2 Technológiák feltérképezése 10](#_Toc26720834)

[2.1 React 10](#_Toc26720835)

[2.2 React Native 12](#_Toc26720836)

[2.3 React Native Web 14](#_Toc26720837)

[2.4 ReactXP 16](#_Toc26720838)

[2.5 Expo és React Native CLI 17](#_Toc26720839)

[2.6 Native Base és egyéb GUI könyvtárak 18](#_Toc26720840)

[2.7 Entity Framework és ASP.NET Core 19](#_Toc26720841)

[2.8 Axios 20](#_Toc26720842)

[2.9 SignalR 21](#_Toc26720843)

[3 Alkalmazás követelmények 23](#_Toc26720844)

[3.1 Felhasználási esetek 23](#_Toc26720845)

[3.1.1 Belépés előtt 23](#_Toc26720846)

[3.1.2 Belépés után 24](#_Toc26720847)

[4 Architektúra 27](#_Toc26720848)

[4.1 Adatbázis 27](#_Toc26720849)

[4.1.1 Adatbázis Táblák 28](#_Toc26720850)

[4.2 Backend 29](#_Toc26720851)

[4.2.1 Modellek 30](#_Toc26720852)

[4.2.2 Adatelérési réteg 30](#_Toc26720853)

[4.2.3 Biznisz logika 31](#_Toc26720854)

[4.2.4 API réteg 33](#_Toc26720855)

[4.3 Kliensek 33](#_Toc26720856)

[4.3.1 Auth 34](#_Toc26720857)

[4.3.2 HomeScreen 34](#_Toc26720858)

[4.3.3 Egyéb bejelentkezett képernyők 34](#_Toc26720859)

[5 Alkalmazás megvalósítása 36](#_Toc26720860)

[6 Irodalomjegyzék 37](#_Toc26720861)

[Függelék 40](#_Toc26720862)

Hallgatói nyilatkozat

Összefoglaló

Abstract

# 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 egyik célszerű megközelítés egy olyan webalkalmazás kifejlesztése, amely fut Androidon és iOS-en is natív alkalmazásként, és még a böngészőben is elindul. A szakdolgozatom keretein belül egy olyan alkalmazást valósítok meg, mely a React Native és a React Native Web segítségével erre mutat példát. A React Native használata során pusztán JavaScriptet alapokon tudunk natív mobil alkalmazásokat fejleszteni. A React Native-ra épülő React Native Web pedig arra nyújt lehetőséget, hogy böngészőben is fusson az ugyanazon, illetve némiképp bővített kódbázisra épülő alkalmazásunk. Így egy nagyrészt közös kódbázisra építve natív Android, natív iOS és böngészőkben is működő webalkalmazást tudunk készíteni.

A dolgozatban a React Native-ot és a hozzá kapcsolódó technológiákat ismertetem. A feladat során számos 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 általam megvalósított alkalmazás a következő problémára próbál megoldást találni: az olyan környezetekben, ahol sok egymástól független felhasználói 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 a szakértőkhöz 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 a React Native Webre építve egy böngészőből is használható webalkalmazást is készítettem. A böngészőből elérhető webalkalmazás desktop környezetben 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 optimalizált felületeket hoztam létre.

A továbbiakban részletesen ismertetem az alkalmazásom megvalósítása során felhasznált technológiákat, valamint bemutatom az alkalmazás megvalósításának részleteit.

# 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 valamilyen 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*

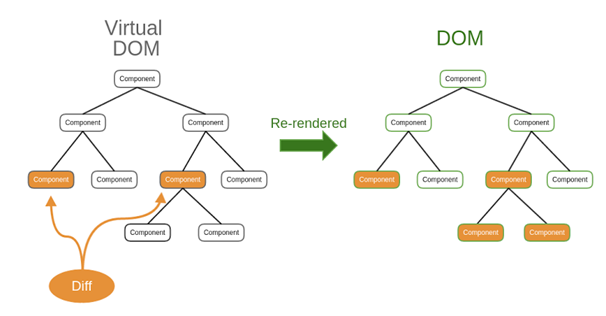
*//A komponens létrejöttekor a name propsba a 'Sara' nevet adom*

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

*//Meghivoma  ReactDOM.render fuggvenyt az elementre*

*//Az oldalra kirajzolódik az element*

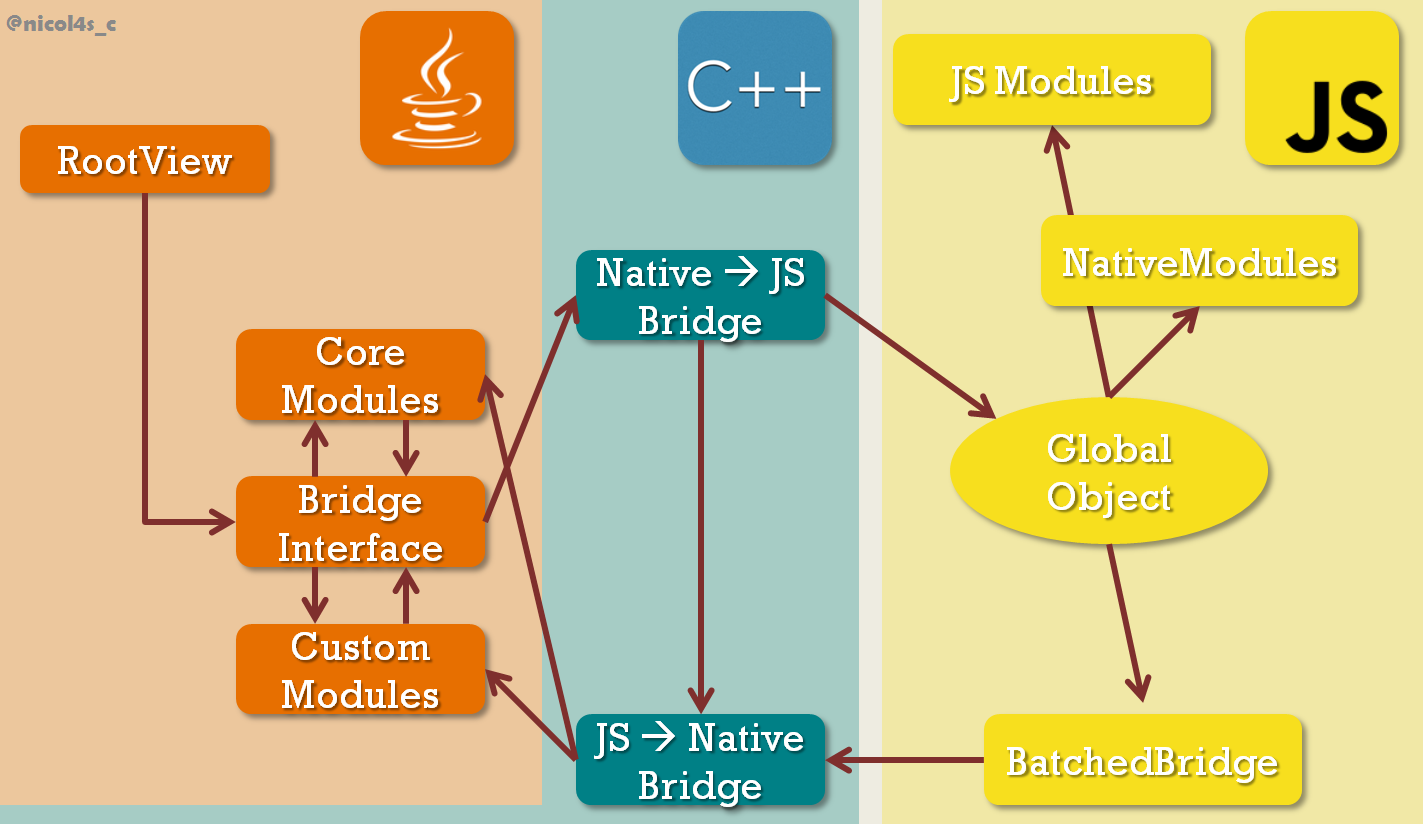
ReactDOM.render(element, document.getElementById("root"));

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.

1.ábra: 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).

2.ábra: 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*

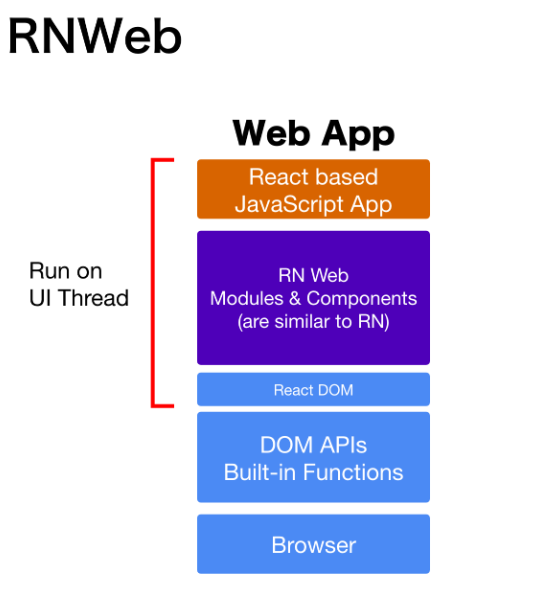
render = () => {

  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.

3.ábra: 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") {

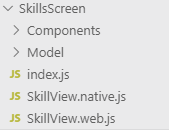
  this.server = env.ServerUrlForAndroid;

  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.

4.ábra: 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. A React Native bár 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 a Microsoft tervezi, 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 buildet ekkor is támogatja és gyorsítja az ExpoSDK.

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 Elements 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 képernyőkép látható

Automatikusan generált leírásA 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.

5.ábra: 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.

Végül emellett a könyvtár mellet döntöttem mivel jól együtt működött az alkalmazásom többi részével és az összetettebb komponensek sokat segítettek a felhasználói felület fejlesztésében.

## Entity Framework és ASP.NET Core

Mivel egy olyan rendszerről beszélünk, ami több felhasználót támogat és az ő interakciójukra épít, 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 a küldött és válaszként kapott adatokat automatikusan alakítja a kérésnek megfelelő formátumra, valamint vissza JSON-be (JavaScript Object Notation). 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 kommunikáció is folyik 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");

# Alkalmazás követelmények

Egy nagyvállalatnál vagy bármilyen környezetben, ahol sok ember és ezzel sok szakterület, ember specifikus képesség található gyakran készítenek kompetencia mátrixokat, hogy felmérjék milyen kompetenciákkal rendelkeznek. A kompetencia mátrix felhasználásával, illetve azzal a feltételezéssel, hogy a különböző területek közt kapcsolatok állnak fenn, egy tudás gráfot építhetünk. Az általam készített alkalmazás arra épít, hogy mikor kérdések merülnek fel egy ilyen környezetben nem mindig triviális, hogy kit kell elérni vagy kit érintenek a kérdések. Viszont egy tudás gráfban keresve viszonylag egyszerű elvek alapján is hatékonyan lehet a kérdéshez leginkább értőket megtalálni.

Az alkalmazás tehát egy több felhasználós rendszer, ami egy tudás gráfot felhasználva próbál szakértőket találni egy kérdéshez a többi felhasználó közül, illetve lehetőséget nyújt a kérdés megvitatására chat formában. Az alkalmazás különlegessége, hogy van egy Androidos natív kliense és egy webes kliens is (bár ez nem rendelkezik minden funkcionalitással).

## Felhasználási esetek

Az alkalmazásnak két elkülönülő állapota van. A felhasználó bejelentkezése előtt és az után.

### Belépés előtt

6.ábra: A képen az alkalmazás belépés elötti Usecase diagrammja látható

A felhasználó ebben az állapotban két dolgot tehet vagy bejelentkezik a már létre hozott fiókjába, vagy regisztrál egy újat az alkalmazásba.

**Regisztráció:** A felhasználónak meg kell adnia egy szabályos email címet, egy felhasználó nevet, amivel később az alkalmazás hivatkozni fog rá, és egy jelszót. Egy email címhez csak egy fiók tartozhat és a felhasználó neveknek is egyedinek kell lenniük.

**Belépés:** A felhasználó be tud lépni egy már létező fiókkal, az ahhoz tartozó egyedi email címet, és jelszót megadva. Bejelentkezés után a felhasználót az alkalmazás bejelentkezve tartja akkor is, ha bezárjuk egészen addig amíg a felhasználó ki nem lép.

### Belépés után

A képen szöveg látható

Automatikusan generált leírásA felhasználó miután bejelentkezett az alkalmazásba képes a különböző beszélgetési szálakat menedzselni, a különböző készségeket kezelni és a saját profilját módosítani.

7.ábra: A képen az készségek kezelésének Usecase diagrammja látható

**Készségek kezelése:** A készségek alatt valamilyen tudást kell érteni, ezekből építenek a felhasználók tudás gráfot melyet az alkalmazás később fel fog használni. A készségeket a *Mobilos felhasználók* tudják csak teljes mértékben kezelni mivel a webes kliens számára a felhasználói felület bizonyos részei készültek csak el. A *Mobilos felhasználók* egy listában látják a tudásfák gyökereit, ezeket képesek *kibontani*, így elérik azok gyerekeit.

A készségeket mobilról *fel lehet venni* a felhasználónkhoz, illetve *le lehet adni* őket. A felhasználóhoz felvett készségek alapján fogja megtalálni őket később az alkalmazás a rájuk vonatkozó kérdésekkel. Mobilos felületről ezen felül lehet *törölni* készségeket a listából, ekkor a készségek végleg törlődnek, a fából is és az őket ismerő felhasználóktól is.

A képen szöveg, térkép látható

Automatikusan generált leírásEzen felül lehet új készségeket felvenni viszont ez a *webes és mobilos felhasználók* számára is egyaránt elérhető. Új készség felvételénél megadhatók a készség szülei. Ez azt jelenti, hogy az új készség a szüleinek egy specifikusabb része esetleg épít rájuk valamilyen módon. Innentől kezdve a tudás gráfban lesz egy kapcsolat a szülők és az új készség között.

8.ábra: A képen a beszélgetési szállak kezelésének Usecase diagrammja látható

**Beszélgetési szálak kezelése:** Az alkalmazás legfontosabb funkciója az adott készségekre vonatkozó kérdések feltétele és ezek megvitatása. Az alkalmazásban szálnak nevezünk egy feltett kérdést és a rá érkező válasz üzenetek összességét. Minden felhasználó számára a legfontosabb a *személyre szabott nézet* melyben a képességeik alapján hozzájuk szóló kérdésekkel induló szálak szerepelnek. Ide azok a szálak kerülnek be melyek vagy pontosan olyan a készséggel vannak megjelölve, amivel a felhasználó is rendelkezik vagy elég magas a prioritásuk és a rajtuk lévő készségek elég közel helyezkednek el a tudás gráfban a felhasználó valamelyik készségéhez.

Azokat a szálakat, amiket a felhasználó hozott létre ki is *törölheti*, amikor úgy érzi, hogy választ kapott a kérdésére. A szálak megnyithatók egy *chatszerű nézetben*. Itt valós időben zajlik beszélgetés a kérdésről, ami elindította a szálat. A felhasználó *tehet fel* bármikor új kérdést. Az új kérdéseknél be kell állítani a készségeket, amiket a felhasználó szerint érint a kérdése. Ezen felül be kell állítani egy prioritást a kérdéshez, ami minél magasabb annál több emberhez jut el a kérdés. Az alkalmazás magasabb prioritás esetén nem csak a beállított készségek birtokosait, de a tudás gráfban közel A képen szöveg látható

Automatikusan generált leíráselhelyezkedő készségek birtokosait is értesíti.

9.ábra: A képen a profil kezelésének és a kilépésnek a Usecase diagrammja látható

**Profil kezelése:** A felhasználó képes a profilja alatt a személyes adatait kezelni. Az *email címét* képes átírni természetesen csak egy másik szabályos és eddig még másik fiók által nem foglalt címre. A *felhasználó nevet* is át tudja írni, illetve a *jelszavát* is megváltoztathatja.

**Kilépés:** A felhasználó kiléphet ezzel az alkalmazás vissza kerül a belépés előtti állapotba.

# Architektúra

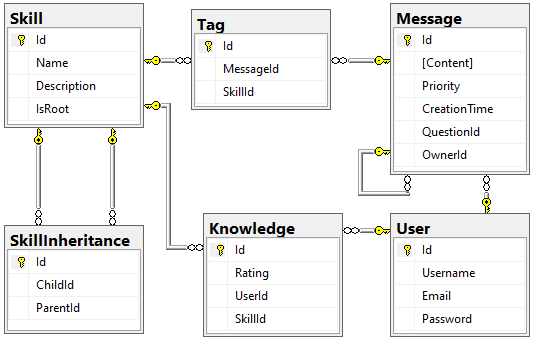
A szakdolgozatban bemutatott szoftver áll egy adatbázisból, egy backendből, és egy frontendből. Az alkalmazás egy több felhasználós rendszer a felhasználók adatait, a feltett kérdéseket és a tudás gráf alapjául szolgáló készségeket mind egy közös adatbázisban tárolja. A felhasználók azonosítását, egymással, és a rendszerrel való interakciójukat a backend végzi. A frontend pedig egy közös kódbázisra épülő Natív Android kliens alkalmazás és egy Web alapú kliens alkalmazás. Ez a két kliens működés szempontjából egymástól teljesen független.

A képen képernyőkép látható

Automatikusan generált leírás10.ábra: A képen a szoftverem absztrakt architechturális rajza látható

## Adatbázis

Az adatbázis elkészítésénél a Microsoft Entity Framework objektum-relációs leképzés részeként bemutatott Code-First megközelítést alkalmaztam. A Code-First lényege, hogy nem egy előre megtervezett adatbázishoz illesztünk C# osztályokat. A C# osztályok készülnek el először és ezekhez automatikusan legenerálja az adatbázis sémát.



11.ábra: A képen az alkalmazás adatbázisának entitás-kapcsolat diagrammja látható

A Code-First megközelítésnek hála az adatbázist könnyen le lehet cserélni az alkalmazás alatt. Én egy Microsoft SQL szervert használtam adatbázisként.

### Adatbázis Táblák

Az alkalmazásnak három fontos résztvevőből áll: a felhasználók, az üzenetek és a készségek. Ezeket és a köztük fenn álló kapcsolatokat tároljuk az adatbázisban.

**Felhasználók:** Az alkalmazás felhasználói a *User* táblában vannak eltárolva. A felhasználók rendelkeznek egy felhasználónévvel és emailcímmel, melyek egyediek minden egyes felhasználó esetén. A jelszavaikat hash-elve tároljuk.

**Üzenetek:** Az alkalmazás egyik fő funkciója az üzenetek váltása ezek az üzenetek a *Message* táblában vannak tárolva. A tartalmukat, prioritásukat és a létrehozási idejüket tartalmazza az adatbázis. Az üzenetek kérdések vagy kérdésekre adott válaszok lehetnek. A kérdések és a hozzájuk rendelt válaszok közt egy a többhöz kapcsolat van. Az üzenetek több az egyhez kapcsolattal kapcsolódnak az őket létrehozó felhasználókhoz.

**Készségek:** Az alkalmazás egy tudás gráfra épül és ennek alkotó elemei a készségek. Ezek a *Skill* táblában vannak tárolva. Mindegyik rendelkezik egy névvel, leírással és egy flag-vel, ami azt hivatott jelezni, hogy gyökér elemről beszélünk-e. A készségek lehetnek gyerek vagy szülő készségek. A gyerekek és a szülők közt több a többhöz kapcsolat van. Ezt a kapcsolatot a *SkillInheritance* tábla tárolj. A felhasználók által feltett kérdések megjelölhetők készségekkel. Ez egy több a többhöz kapcsolat az üzenetek és a készségek között, amelyet a *Tag* tábla tárol. A készségeket a felhasználók magukhoz rendelhetik. Ez egy több a többhöz kapcsolat, amit a *Knowledge* tábla tárol.

## Backend

A képen képernyőkép látható

Automatikusan generált leírásA backend kezeli a felhasználókat és a köztük és a rendszer között lezajló interakciókat. A Microsoft ASP.NET Core webes keretrendszerével készült a szerveroldali kód. A szerver elkészítésekor törekedtem a DDD (Domain-Driven design) követésére. Ennek következtében a kódot három külön álló részre bontottam. Ez a három réteg az adatelérési, bizniszlogikai, és a kliensekkel való kommunikációért felelős API réteget jelenti. A DDD elveinek megfelelően az alkalmazás többi rétege mind a bizniszlogikától függ.

12.ábra: A képen a backend Package diagrammja látható

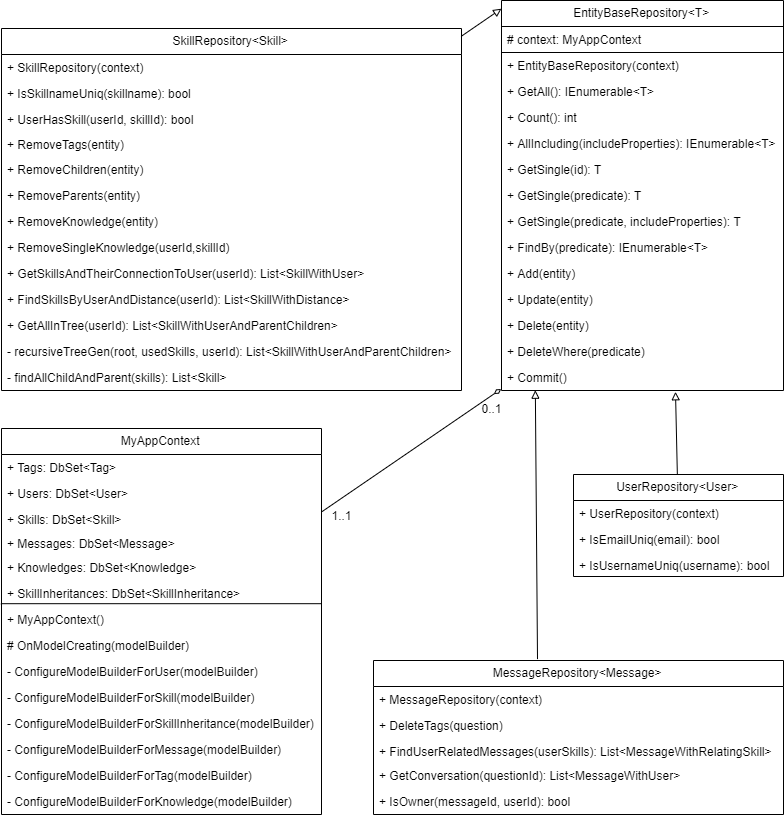
### Modellek

A képen szöveg, térkép látható

Automatikusan generált leírásA DDD filozófiájával egyedül a Model csomag megy szembe, amely a backend összes többi része által használt modelleket tartalmaz. Ezek nem csak domain modellek, az Entity Framework ezekhez a modellekhez mappeli az adatbázis megfelelő tábláit. A Code-First megközelítés során ezeket a modelleket kellet elkészíteni és ezek alapján generálódik le az adatbázis. Az összes modellben közös, hogy egy egyedi GUID segítségével azonosítható a backenden. Ezt a közös tulajdonságot egy közös interfész implementálásával érem el.

13.ábra: A képen a Model csomagban található osztályok osztály diagrammja látható

### Adatelérési réteg

Az adatelérési réteg megvalósításához a repository mintát követtem. A biznisz objektumokhoz (Message, Skill, User) készítettem saját repository-kat. Ezek egy közös általános repository-ból származnak le. A közös repository az adatbázis entitásokra általánosságban jellemző törlés, módosítás és listázás műveletek általánosított formáit tartalmazza. A biznisz objektumok egyedi adatelérési műveletei, amelyek általában két entitás közti kapcsolati táblára is építenek a biznisz objektumokra specifikus repository-kban találhatóak. Magát az adatbázis a MyAppContect osztály segítségével érem el. A kontextusra mutató változót a repository-k közösősében vettem fel és a specifikus repository-k öröklésen keresztül érik el.

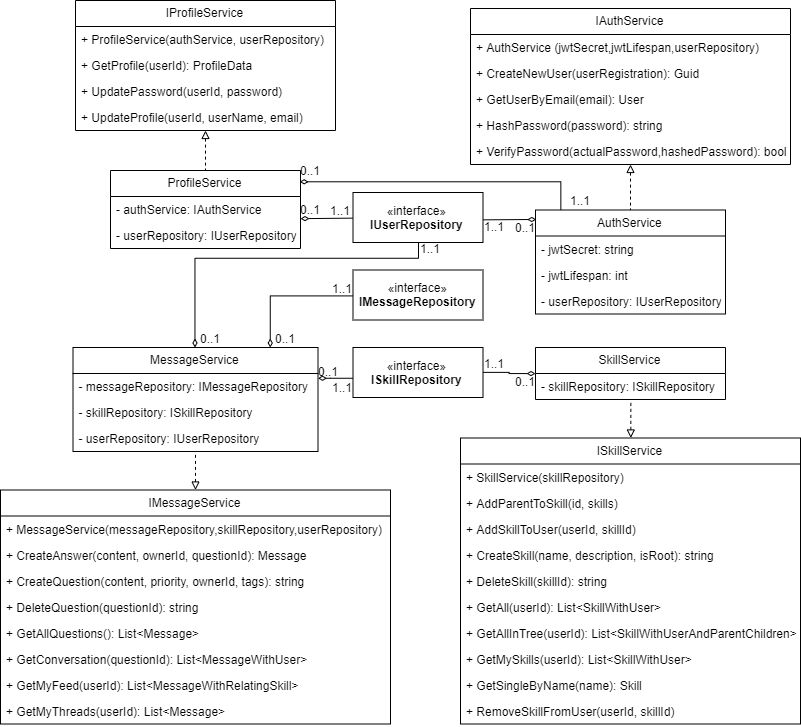
14.ábra: A képen a Data csomag osztály diagrammja látható

### Biznisz logika

Az alkalmazás lényege a biznisz logika ettől a rétegtől függ az összes többi réteg. Itt találhatók a szolgáltatások, amelyek a repository-kból kinyert adatokon végeznek műveleteket vagy több repository-ból szerzett adatokból készítenek egy biznisz objektumot. Ez a viselkedés feltételezné, hogy a biznisz logikai réteg függ az adatelérési rétegtől. A függés viszont meg kerülhető a függőség megfordításának elvét felhasználva.

Ezt a tervezési mintát követve létrehoztam a biznisz logikai rétegen belül egy RepositoryInterfaces nevű csomagot. Ettől a csomagtól függenek a szolgáltatások így a rétegnek nem lesz kifelé mutató függősége. A csomagban definiált interfészeket implementálják a repository-k ennek köszönhetően az adatelérési réteg függ a biznisz logikától és nem fordítva.

A szolgáltatások a működésükhöz szükséges repositorykat illetve egyéb szolgáltatásokat függőség befecskendezésen keresztül kapják. Ennek a tervezési mintának hála a szolgáltatás által felhasznált függőségek könnyen módosíthatók a szolgáltatás megváltoztatása nélkül.

A szolgáltatásokon és a repository interfészeken felül a biznisz logika részét képezik a különféle biznisz objektumok. Ezeket a szolgáltatások hozzák létre vagy már ilyen formában nyerik ki a repository-kból. A különálló Model csomagban megismert biznisz objektumok mellet az itt definiált formákban utazik az adat a szerveren.

15.ábra: A képen a BLL csomag részleges osztály diagrammja látható

### API réteg

Az API (Application Programable Interface) réteg a backend utolsó rétege. Ez a szerver legkülső rétege, itt találhatók a kontrollerek és a SignalR Hubok. Az API réteg elemei a szükséges szolgáltatásokat függőség befecskendezésen keresztül érik el. Ez a réteg a biznisz logikától függ és minden kérést szolgáltatások segítségével kezel le, az adat elérési rétegtől teljesen független.

A kontrollerek a Controllers csomagban találhatók. Az összes az ASP.NET Core ControllerBase ősosztályából származik le. Mindegyik rendelkezik az ApiController attribútummal ez jelzi, hogy HTTP kérések kiszolgálására hivatottak. A rendszer több felhasználós, a különböző felhasználók azonosítására és kezelésére az ASP.NET Core Identity keretrendszerét használtam. A bejelentkezésért és felhasználói regisztrációért felelős kontrolleren kívül minden kontroller rendelkezik az Authorize attribútummal. Ennek az attribútumnak hála a többi kontroller csak bejelentkezett felhasználókat szolgál ki.

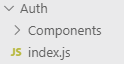
Az alkalmazásban a kommunikáció nem csak http protokoll felett zajlik. A Hubs csomagban található SignalR hub-ok az ASP.NET Core SignalR könyvtárának segítségével két irányú valós idejű kommunikációt folytatnak a klienssel.

Az API rétegben található a DTO (Data Transfer Object) csomag. Ebben a csomagban vannak definiálva azok az objektumok, amelyek a szerver felé irányuló kérésekben vagy a szerver által adott válaszokban szerepelnek. Az ilyen szállító objektumok és a biznisz objektumok közti átalakítás az AutoMapper könyvtár segítségével történik.

## Kliensek

Az alkalmazás kliens oldala két klienst jelent egy Web alkalmazást és egy Android natív alkalmazást. A két alkalmazás kódbázisa bár nagy rész közös szükség volt platform specifikus megoldások alkalmazására. A Web alkalmazás belépési pontja egy index.js fájl és rendelkezi egy index.html fájlal is. Az index.js a kiinduló index.html-re ki render-eli a kliens oldal közös belépési pontjaként szolgáló App.js már React Native-ban megírt tartalmát. Az Android alkalmazás belépési pontja az App.js. A klienskód a különböző megjelenítendő képernyők alapján van tagolva.

### Auth

Az Auth képernyő két egymást váltó ablakból áll egy bejelentkező ablakból és egy regisztrációs ablakból. A fő képernyő alképernyői egy a csomagban található Components csomag alatt helyezkednek el. Az Auth képernyő minden eleme közös a Webes és az Android kliens között.

16.ábra: A képen az Auth képernyőhöz tartozó csomag látható

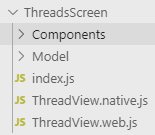
### HomeScreen

A HomeScreen routerrel ellátott képernyő. Ez a router tartalmazza az összes bejelentkezés után elérhető képernyőt és teszi lehetővé a köztük való navigációt. A router implementációja különbözik a Web és az Android kliensek között. Webnél a react-router-dom könyvtár oldja meg a navigációt. A natív esetben a React Native saját react-router-native navigációs könyvtárat használ az alkalmazás.

A felhasználó egy oldalsó menü sáv segítségével tud navigálni. Ez a HomeScreen része és a Components csomag alatt található.

17.ábra: A képen az HomeScreen képernyőhöz tartozó csomag látható

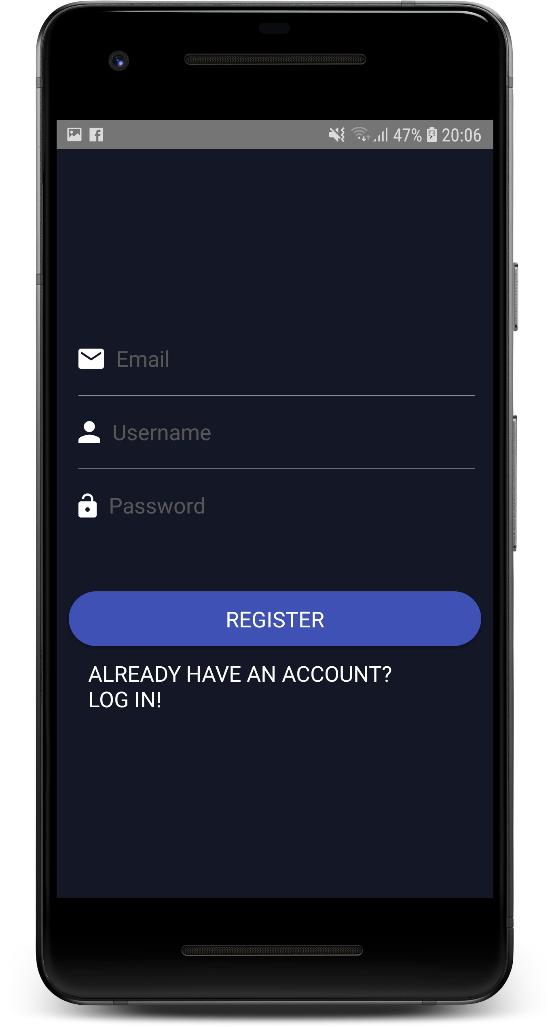
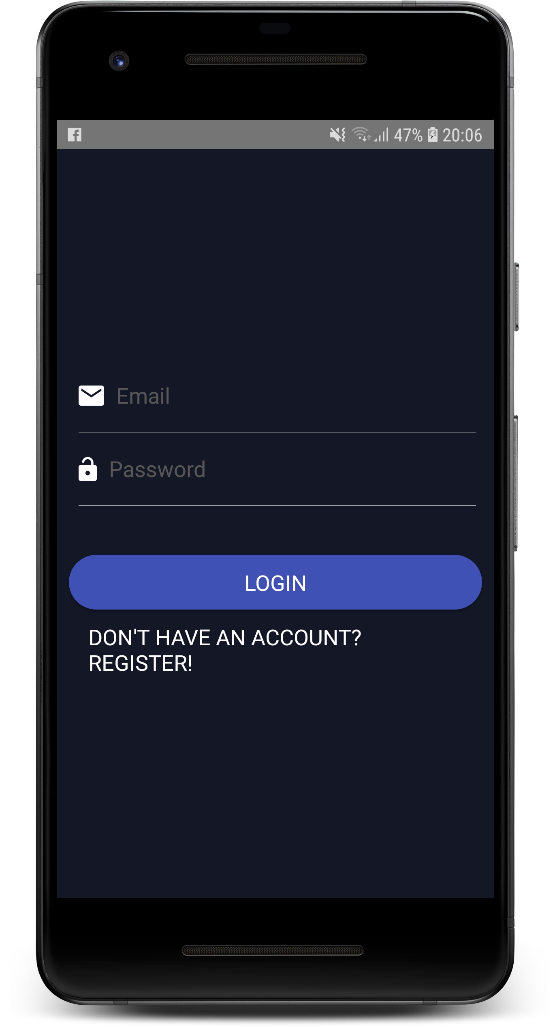
### Egyéb bejelentkezett képernyők

A kliens oldali alkalmazás többi bejelentkezett képernyője mind ugyanaz alapján a minta alapján épül fel. Minden csomagban található egy index.js ez tartalmazza a képernyő kliens oldali logikáját. Itt történik a szerverrel való kommunikáció és az adatok átalakítása a kliensnek, illetve szervernek megfelelő alakokra. Az index.js mellet van egy a képernyő kinézetéért felelős nézet (View). Ezek a nézetek lehetnek platform specifikusak, mivel a képernyők megjelenítése nem egységes a Web és az Android kliensek között. A bonyolultabb komponensek további rész komponenseket tartalmazhatnak ezek a Components csomag alatt találhatók. A komponens specifikus enumok pedig a Model csomagban kapnak helyet.

18.ábra: A képen az ThreadScreen képernyőhöz tartozó csomag látható példaképp az egyéb bejelentkezett képernyőkre

# Alkalmazás megvalósítása

## Bejelentkezés és regisztráció

Az alkalmazás Androidon és Weben is legelőször egy bejelentkező képernyővel indít. Itt a felhasználó be tud jelentkezni, ha van már fiókja. Amennyiben nincs még fiókja regisztrálhat egy újat.

18.ábra: A képen a belépő és regisztrációs felületek láthatóak

A felhasználó a két nézet között a komponensek alján található linkkel válthat. A beviteli mezőket mindkét képernyő esetén egy KeyboardAvoidingView-val vettem körbe. Androidnál ennek hála a billentyűzet nem takarja ki őket. Ez a komponens nincs hatással a Web kliensre. A kliens mind regisztrációnál, mind belépésnél egy HTTP hívást kezdeményez a szerver felé Axios segítségével.

### Regisztráció

Regisztráció esetén a felhasználó nevet, jelszót és email címet egy POST kérés formájában elküldi a kliens a szervernek. Szerver oldalon ezt az AuthController fogadja.

*//Ezt a modellt várja a szerver és az attribútumok alapján ellenőrzi is.*

public class RegisterViewModel

{

    [Required]

    [StringLength(60, MinimumLength = 2)]

    public string Username { get; set; }

    [Required]

    [EmailAddress]

    public string Email { get; set; }

    [Required]

    [StringLength(60, MinimumLength = 8)]

    public string Password { get; set; }

}

A modell validációja után a kontroller át hív az AuthService szolgáltatásba és megpróbál létrehozni egy felhasználót a megadott adatokkal. A beérkezett adatokat az AutoMapper segítségével UserRegistration biznisz objektummá alakítja.

Az AuthService CreateNewUser függvénye hozza létre az új felhasználót. A felhasználó elkészítése során ellenőrzi, hogy van e már felhasználó a megadott névvel vagy email címmel. Már létező felhasználó esetén hibát dob, amelyet a kontroller kezel le. Ellenőrzés után létrehoz egy új felhasználót és elmenti a UserRepository segítségével az adatbázisba. A függvény az elkészített felhasználó egyedi azonosítójával tér vissza.

A kontroller elkapja a felhasználó készítése során keletkező hibákat és hibát jelez vissza a kliens felé. Sikeres felhasználó generálás esetén az AuthService segítségével az új felhasználó azonosítója alapján készít egy JWT (JSON Web Token) tókent és visszaküldi a kliensnek. Innentől kezdve az alkalmazás ezt fogja használni a felhasználó azonosításához.

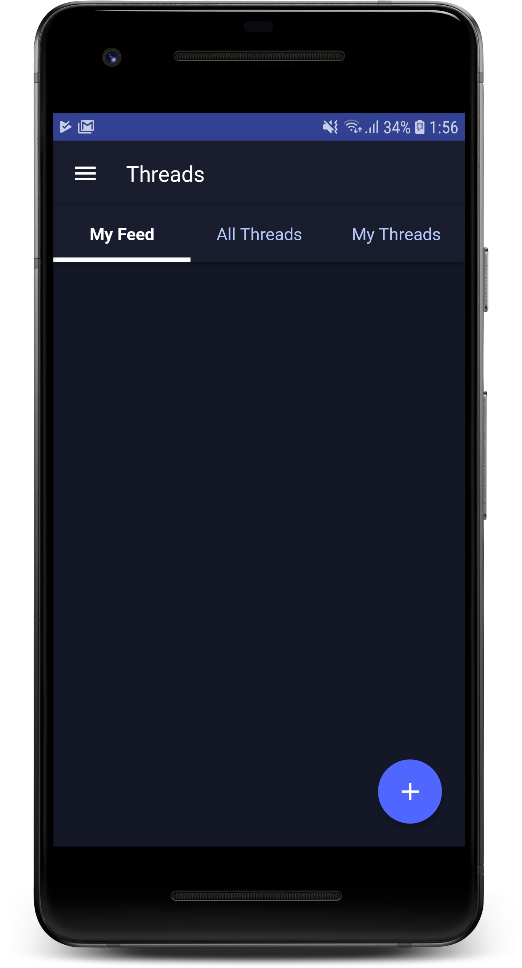
Kliens oldalon az Axios segítségével elkapja és lekezeli az alkalmazás a szervertől érkező hibákat. A sikeres felhasználó létrehozás után a deviceStorage szolgáltatást használva kimenti a kliens a JWT tókent. A deviceStorage az AsyncStorage segítségével, kulcs érték párként menti le az adatokat lokálisan. Mikor az alkalmazást újra elindítjuk, ha nem jelentkeztünk ki a felhasználó, akkor az elmentett JWT tóken alapján bejelentkezve nyílik meg az alkalmazás.

### Bejelentkezés

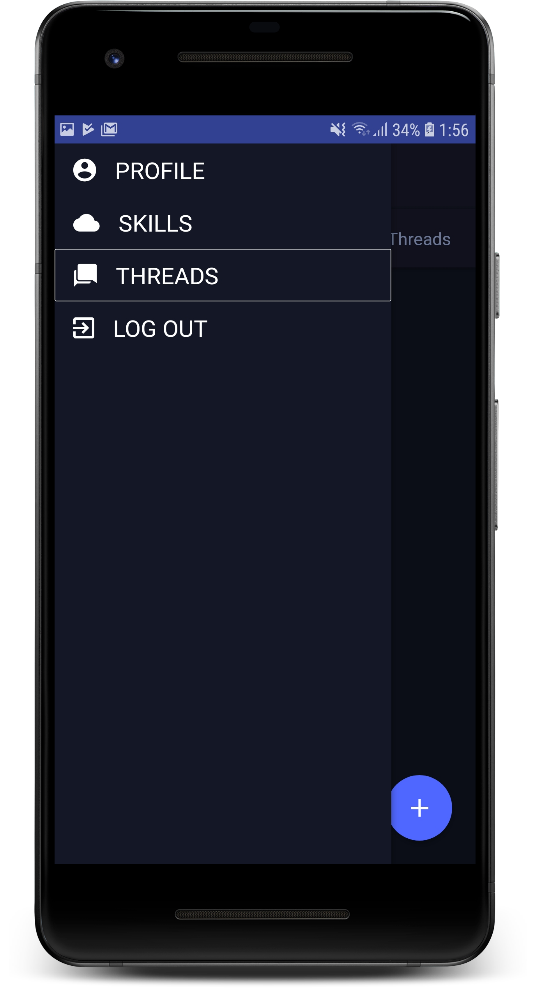
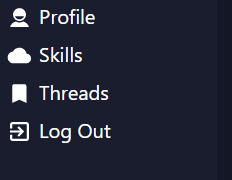
A bejelentkezés esetén egy POST kérés formájában a felhasználónevet és a jelszót küldjük fel a szerverre. A szerver ezeket a regisztrációhoz hasonlóan ellenőrzi. Az AuthService segítségével lekéri a felhasználót az email címe alapján. Ha nem létezik felhasználó ezzel az email címmel akkor hibával tér vissza a szerver. Létező felhasználó esetén ellenőrzi a jelszóhelyességét. A jelszavakat szerver oldalon hash-elve tárolom az adatbázisban. Az AuthService VerifyPassword függvénye a Crypto VerifyHashedPassword függvényével ellenőrzi a jelszót. Helytelen jelszó esetén hibát jelez a kliens számára. Helyes jelszónál a bejelentkező felhasználó egyedi azonosítója alapján generál egy JWT tókent.

A kliens helyes bejelentkezés esetén a regisztrációhoz hasonlóan elmenti a JWT tókent és amíg nem jelentkezik ki a felhasználó addig azt használva bejelentkezve tartja.

## Bejelentkezett felhasználó

A kliens indításakor Android és Web esetén is megpróbálja betölteni a tárolt JWT tókent. A lokális tárolást az Android a SharedPreferences segítségével valósítja meg. Ez egy olyan API Androidon, amely lehetővé teszi, hogy egy fájlban kulcs érték párok formájában tároljunk adatokat. A SharedPreferences fájlokat az Android keretrendszer kezeli és garantálja, hogy csak a mi alkalmazásunk írhassa és olvashassa. Web esetében a felhasználó által használt böngésző Local Storage-ának segítségével tároljuk el a JWT tókent.

19.ábra: A képen a bejelentkezett alkalmazás látható

Sikeres tóken betöltés után az alkalmazás a kezdő képernyőn indít. A kliensben ezek után egy navigációs sáv segítségével tud a felhasználó navigálni. A navigációt egy Router komponens teszi lehetővé. A navigáláshoz használt könyvtárak bár különböznek a két kliens között a Router komponens felépítése nagyon hasonló mindkét kliensben.

20.ábra: A képen a navigációs oldalsáv látható Androidon (bal) és Weben (jobb)

A két kliens fő komponensében az alapvető különbség az oldalsó navigációs sáv viselkedése és az Androidon meglévő felső fejléc. Androidhoz

# Irodalomjegyzék

|  |  |
| --- | --- |
| [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] | „SignalR,” [Online]. Available: https://hu.wikipedia.org/wiki/SignalR. |
| [5] | „ReactXP,” [Online]. Available: https://microsoft.github.io/reactxp/docs/getting-started.html. |
| [6] | „ReactVirtualDom,” [Online]. Available: https://jelvix.com/blog/is-react-js-fast. |
| [7] | „ReactNativeBridge,” [Online]. Available: https://levelup.gitconnected.com/wait-what-happens-when-my-react-native-application-starts-an-in-depth-look-inside-react-native-5f306ef3250f. |
| [8] | „ReactComponent,” [Online]. Available: https://reactjs.org/docs/components-and-props.html. |
| [9] | „ExpoSDKInWork,” [Online]. Available: https://docs.expo.io/versions/latest/workflow/how-expo-works/. |
| [10] | „Expo,” [Online]. Available: https://docs.expo.io/versions/latest/. |
| [11] | „Axios vs Fetch,” [Online]. Available: https://blog.logrocket.com/axios-or-fetch-api/. |
| [12] | „Axios,” [Online]. Available: https://github.com/axios/axios. |
| [13] | „ASP.NET\_Core,” [Online]. Available: https://en.wikipedia.org/wiki/ASP.NET\_Core. |
| [14] | [Online]. Available: https://reactjs.org/. |
| [15] | [Online]. Available: https://www.w3schools.com/whatis/whatis\_react.asp. |
| [16] | [Online]. Available: https://en.wikipedia.org/wiki/React\_(JavaScript\_library). |
| [17] | [Online]. Available: https://en.wikipedia.org/wiki/React\_Native. |
| [18] | [Online]. Available: https://cordova.apache.org/docs/en/9.x/guide/overview/index.html#page-toc-source. |
| [19] | [Online]. Available: http://www.reactnative.com/react-native-dom/. |
| [20] | „Code-First,” [Online]. Available: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx. |
| [21] | „Domain-Driven Designe,” [Online]. Available: https://airbrake.io/blog/software-design/domain-driven-design. |
| [22] | „PackageDiagram,” [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/. |
| [23] | „Class Diagram,” [Online]. Available: https://en.wikipedia.org/wiki/Class\_diagram. |
| [24] | „Agregation and Composition,” [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/. |
| [25] | „Függőség megfordításának elve,” [Online]. Available: https://hu.wikipedia.org/wiki/F%C3%BCgg%C5%91s%C3%A9g\_befecskendez%C3%A9s%C3%A9nek\_elve. |

Függelék