**Projekt Labor**

**Alkalmazásfejlesztés web alapú mobil platformra**

Baumgartner János Tancsics László

témavezető Németh Ádám

Abafalvi Balázs

csapattagok

Tartalom

[1. Felhasználói dokumentáció 3](#_Toc56013763)

[2. Fejlesztői dokumentáció 4](#_Toc56013764)

[2.1 Feladat leírása 4](#_Toc56013765)

[2.2 Az alkalmazott fejlesztői eszközök 5](#_Toc56013766)

[2.2.1 React 6](#_Toc56013767)

[2.2.2 Ionic 6](#_Toc56013768)

[2.3 Adatmodell leírása 7](#_Toc56013769)

[2.3.1 Firebase 7](#_Toc56013770)

[2.3.2 Adatbázis felépítése 7](#_Toc56013771)

[2.4 Munkaterv 9](#_Toc56013772)

[2.4.1 Feladat és egyéb tudnivalók átbeszélése 9](#_Toc56013773)

[2.4.2 Ismerkedés a technológiákkal 9](#_Toc56013774)

[2.4.3 Funkcionális követelmények 9](#_Toc56013775)

[2.4.4 Kódolás 14](#_Toc56013776)

[2.4.4.1 Tancsics László kódja (menürendszer*+*Rólunkoldal) 14](#_Toc56013777)

[2.4.4.2 Németh Ádám kódja (Árajánlat kérés, Login, Naptár) 19](#_Toc56013778)

[2.4.4.3 Abafalvi Balázs kódja (Admin felület) 20](#_Toc56013779)

# 1. Felhasználói dokumentáció

# 2. Fejlesztői dokumentáció

## 2.1 Feladat leírása

A projekt során egy webes alkalmazás fejlesztése a cél, amely megkönnyíti, valamint hatékonyabbá teszi a kis- és középvállalkozások (KKV) számára az egyes szolgáltatások használatát

Az alkalmazás segítségével lehetőség nyílik a felhasználók számára az árajánlatok kérése időponttal együtt. Az admin felhasználó pedig ezeket a kéréseket el tudja fogadni, utasítani, esetlegesen „talonba” helyezni. Továbbá lehetőség nyílik admin oldalról a statikus oldalak tartalmának szerkesztése.

Manapság a BYOD (Bring Your Own Device) trend hatalmas népszerűségnek örvend. Mindenki a saját eszközein próbálja intézni az ügyeit. Ezen eszközök közül a legnépszerűbb kétségtelenül az okostelefon, hiszen könnyen hordozható, és szinte mindent meglehet vele csinálni, amit egy asztali számítógéppel is. Így nem is volt kérdés, hogy ennek a webalkalmazásnak mobilapplikációként is működnie kell.

## 2.2 Az alkalmazott fejlesztői eszközök

**Fejlesztői környezet**

* Atom Editor

**Programozási nyelvek**

* HTML5
* CSS
* Javascript

**Adatbázis nyelv**

* JSON (NOSQL)

**Adatbázis kezelő rendszer**

* Firebase

**Javascript könyvtár**

* React

**Keretrendszer**

* Ionic

**Szövegszerkesztő**

* Word

**Képszerkesztő**

* GIMP 2

**További web technológia**

* Progressive Web Apps (PWA)

### 2.2.1 React

A **React** egy Facebook által kifejlesztett nyílt forráskódú JavaScript könyvtár, amit felhasználói felületek programozására használunk. Az MVC (Model View Controller) programramtervezési mintából a React a *„V-t”* képezi. Tehát csak React-et használva nem lehet web alkalmazást fejleszteni!

Komponensekből tevődik össze. Ezek JavaScript függvények, tetszőleges számú inputtal és egy outputtal. JSX (JavaScript XML) szintaxissal íródnak. A komponensek fejlesztése ES6 szabvány szerint történik, amely a következőket határozza meg:

* A komponensek tetszőleges számú bemenetet fogadhatnak és ezeket az úgynevezett „*props”* nevű objektumba tárolják.
* Mindig kell lenni-e egy *„render”* metódusnak. Ez a metódus határozza meg, hogy mi legyen megjelenítve, amikor a komponens meghívódik.
* Minden komponens egy belső állapotot menedzsel. Amikor az állapot változik akkor az adott komponens *„render”* függvénye meghívódik.

React könyvtár segítségével íródott alkalmazások teljesítménye kivételesen magas. Ennek az oka nagyon egyszerű. A render-elés Virutal DOM (Document Object Model) segítségével történik. Tegyük fel, hogy sok kis *„View”* változást szeretnénk eszközölni. Ezek a változások egyfolytában frissítik a DOM-ot🡪alkalmazás teljesítménye romlik. React megoldása: Ezek a *„view”* módosítások tárolódjanak el a VDOM-ban. Ezután jön egy algoritmus, amely először összehasonlítja a a Virtual DOM jelenlegi és előző állapotát, majd végrehajtja azt a minimum számú módosítást az eredeti DOM-on, ami ténylegesen szükséges, a kívánt állapot eléréséhez. Felhasználói interakciónál is ez történik!

### 2.2.2 Ionic

Az Ionic egy nyílt forráskódú keretrendszer, amit az Ionic cég fejlesztett ki. Segítségével hibrid mobilalkalmazásokat fejleszthetünk. Angularra fejlesztették ki először. 2019-ben oldották meg, hogy React és Vue-val is tudjon együtt dolgozni. Használatához npm modul és Node.js szükséges.

Miután Ionic segítségével lefejlesztettünk egy alkalmazást, azt natív alkalmazásboltba terjeszthetjük (App Store/Google Play). Az alkalmazásboltból Cordova (Android) vagy Capacitor (iOS) segítségével telepíthetjük saját mobilunkra.

Tartalmazza a legújabb web technológiát az úgynevezett *„PWA” (Progressive Web Apps)-*t. A PWA lehetővé teszi a felhasználók számára, hogy a webalkalmazás native applikációként működjön. Tehát az alkalmazás ikonja (mint a rendes appoknál) kint lesz az asztalon. Elindításkor nem egy web-es felület nyílik meg, hanem egy applikációs felület. Továbbá a webalkalmazás hozzáférhet a telefon különböző eszközeihez, mint például a kamerához, szenzorokhoz. Teszi mind ezt úgy, hogy rajta sincs fizikailag a tárhelyünkön, hanem a böngészőben könyvjelzőként van elmentve.

Az Ionic által támogatott platformok: Windows (10), Andorid (4.4-től), iOS (10-től).

## 2.3 Adatmodell leírása

### 2.3.1 Firebase

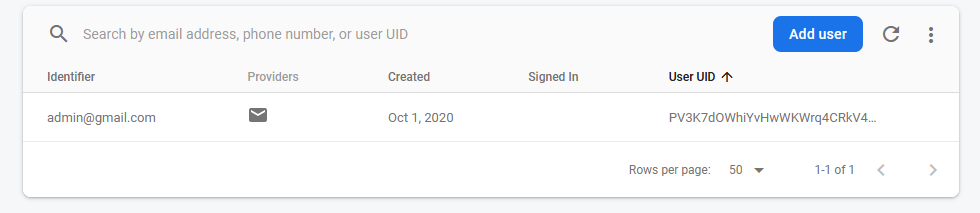
A **Firebase** egy szolgáltatáscsomag a Google-től. Segítségével különböző backend feladatok oldhatóak meg könnyedén. Ilyen például a felhasználók beléptetése akár Google vagy Facebook azonosítással, adatok felhős tárolása, vagy éppen lehetőséget nyújt a felhasználók számára, hogy kedvencek közé rakhassanak tartalmakat.

Firebase szolgáltatásai nem csak a fejlesztésre koncentrálnak, számos lehetőséget nyújt arra, hogy mobil applikációnk mérhető növekedést érjen el:

* Mérhetők/elemezhetők a felhasználók interakciói
* Az appunkat aktívabban használó közönség felé kommunikálhatunk új alkalmazás verziók kiadása nélkül.
* Fizetett kampányokkal szerezhetünk további felhasználókat

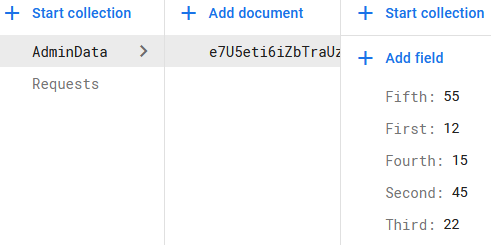
Ez egy „Realtime Database”, tehát az adatok szinkronizálása valós időben történik. NoSQL felhőalapú adatbázist használ. Itt tárolódnak az adatok. JSON alapú, így könnyen belátható, hogy egyszerű adatszerkezetek tárolására alkalmas ez a rendszer. **Használjunk SQL alapú rendszert bonyolultabb adatszerkezetek tárolása esetén!**

### 2.3.2 Adatbázis felépítése

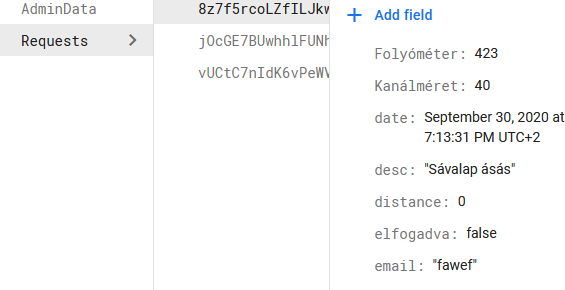
 A projekt során két alapvető authorizációs adatot tárolunk, amelyek a következők: Username (Email cím), jelszó.Mivel nem kötelezzük a felhasználókat, hogy az alkalmazás funkcióinak használatához regisztrálniuk kelljen ezért, csak egy admin felhasználó van beleégetve a rendszerbe. A beleégetést az Add user gombra rámenve lehet elvégezni. A jelszó automatikusan, titkosítva kerül elmentésre (User UID).

További két collection:

1. AdminData: Itt tároljuk az árképzéshez tartozó konstans értékeket.
   * First: Sávköltség (Sávalap ásás munkához)
   * Second: Drainköltség (Ház körüli drainezés munkához)
   * Third: Négyzetméter-költség (Térkő alap előkészítés munkához)
   * Fourth: Köbméter-költség (Törmelék elhordás munkához)
   * Fifth: Tükörköltség (Tüköralap munkához)



1. Requests: Itt tároljuk a bejövő munkák paramétereit:
   * desc: A Munka megnevezése
   * munkához tartozó mértékegységbeli adatok (pl:. Folyóméter,Kanálméret)
   * date: A Munka határideje
   * distance: A munka helyének és a telephely közötti távolság
   * elfogadva: Küldéskor ez a mező mindig hamis (admin változtatja majd át, ha elfogadja a munkát).
   * email: A Munkakérő email címe (egyedi azonosítást tesz lehetővé).



## 2.4 Munkaterv

### 2.4.1 Feladat és egyéb tudnivalók átbeszélése

Mint minden termék előállításánál, itt is az első lépés a feladat értelmezése volt. A témavezető átfogóan elmondta, hogy mit kell megvalósítanunk és hogy azt milyen technológiával kell megtennünk. Ezután tisztázásra kerültek a határidős kérdések. Végezetül a konzultációs időpontok kialakítására helyeződött a hangsúly. Mindenki egyetértett, hogy a lendületes szép munkát csak akkor tudjuk fenntartani, ha legalább heti egy alkalommal *„jelentünk”* a témavezetőnek.

### 2.4.2 Ismerkedés a technológiákkal

Mielőtt egyből fejest ugrottunk volna a projekt közepébe, elkezdtünk megismerkedni a technológiákkal. Szerencsére számtalan jobbnál-jobb anyag található fent a világhálón.

Segítségünkre voltak:

* A technológiák hivatalos oldalai
* Fejlesztők, fejlesztői csoportok által készített Youtube videók.
* Nyilvánosan publikált anyagok.

Folyamat alatt megtanultuk a technológiák:

* Alapvető ismérveit.
* Előnyeit, hátrányait.
* Kódolási konvenciáit.
* A projekt generálásának lépéseit.

A folyamat végére, mindenki elkészítette a saját kis gyakorló programját.

### 2.4.3 Funkcionális követelmények

Elérkeztünk az alkalmazás fejlesztés egyik legfontosabb lépéshez. A témavezetőnk összeszedte az összes megvalósítandó funkcionális követelményt, amelyek a következőek:

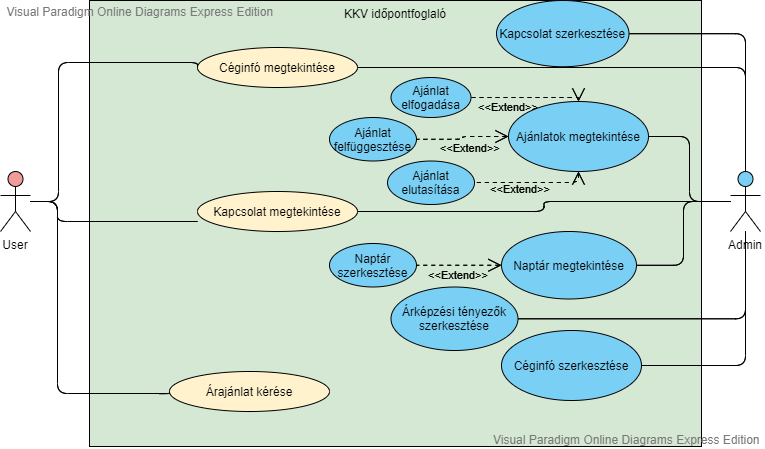
1. Felhasználói oldalon:

* **Kezdőlap:** Miről szól ez az alkalmazás? Milyen lehetőségeket kínál a felhasználóknak az alkalmazás? Ezekre a kérdésekre kell választ adnunk.
* **Rólunk:** Egy hosszabb leírás, hogy mivel foglalkozunk. Alapvető elérhetőségi információk megadása (email, telefonszám, cím). Továbbá biztosítani a felhasználók számára, ha további kérdésünk van, akkor közvetlenül az oldalon email formájában fel tudják azokat tenni.
* **Árajánlat kérés:** Cél, hogy az ügyfelek alapvető paraméterek megadásával, árajánlatot kapjanak, hogy az adott munkanemet a cégünk mekkora ellenérték fejében fogja elvégezni. Ha az ügyfél elégedett a kapott árral, akkor tudjon leadni rendelést a munkával kapcsolatban.
* **Bejelentkezés:** Admin számára bejelentkezés lehetősége

1. Admin oldalon:

* **Kezdőlap:** Ugyanaz. mint a felhasználói oldalon.
* **Rólunk:** Ugyanaz. mint a felhasználói oldalon, de az oldal tartalmát lehessen szerkeszteni.
* **Árajánlat kezelése:** Itt tekinthetőek meg az ügyfelek által leadott kérések. A kérések mellet legyen mindig megjelenítve a telephely és a munkahely közötti távolság! Az admin tudjon rendelkezni a kérésekről. Három féle rendelkezést különböztetünk meg:
  + - * *Kérés elfogadása:* Ekkor, a kérés paraméterei között megadott határidő alapján, kerüljön be egy naptárba az adott ügyfél által leadott munka kérelem. Kapjon a felhasználó e-mailes tájékoztatást a munka elfogadásáról.
      * *Kérés elutasítása:* Kérés kitörlődik az adatbázisból. Kapjon a felhasználó e-mailes tájékoztatást a munka elutasításáról.
      * *Kérésről később rendelkezés lehetősége:* Nem történik semmi, csak a következő kérésre ugrik.
* **Napirend:** Egy naptár jelenik meg ezen az oldalon. Ez a naptár tartalmazza a már elfogadott munkákat. Lehessen a **naptárat szerkeszteni**, olyan módon, hogy már az elfogadott munkákat lehessen visszamondani (törölni), vagy módosítani.
* **Árak szerkesztése:** Az admin tudja az árképzési változókat az alkalmazásból módosítani.
* **Kijelentkezés:** Admin számára kijelentkezés lehetősége.

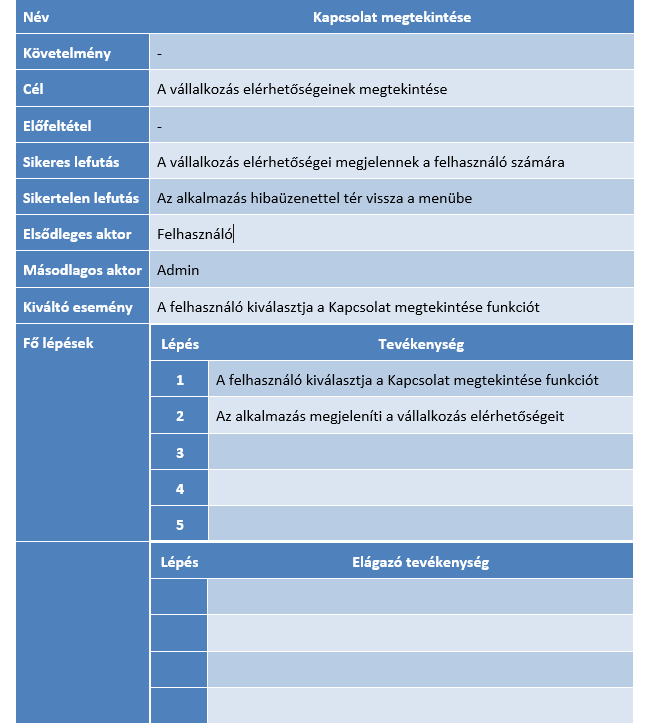
A funkcionális követelmények jobb megértése érdekében Use Case diagramot készítettünk:



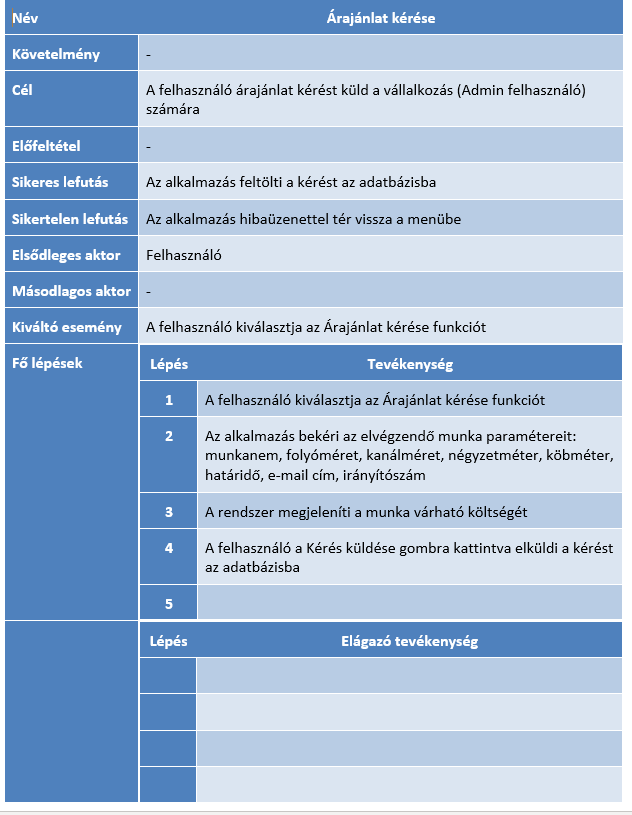
A Use Case diagramm lényege, hogy reprezentálja a felhasználói interakciókat a rendszerrel. Ebben az esetben két felhasználót (aktort) különböztetünk meg: admin, és a hagyományos felhasználó. A diagrammon ezek a pálcika emberek. Az aktorokhoz különböző Use Casek (funkciók) tartoznak. A diagrammon ezek az ovális körök. Az aktorok és Use Case-k között futó élek a kapcsolatok.

Diagrammon jól láthatóak az egyes funkciók hovatartozásai. Továbbá látható, hogy nagyon sok funkció az *„extended”* jelzőt kapta. Ez annyit jelent, hogy azok a funkciók nem mindig futnak le. Érthető hiszen minek terheljük fölöslegesen az alkalmazásunkat azzal, hogy rengeteg funkció fut mikor, könnyen szűrhetőek megfelelő feltételekkel.

Vizsgáljuk meg néhány funkciót kicsit részletesebben. Ezt az ügynevezett Use Case tábla segítségével tehetjük meg. Ezek a táblák lehetővé teszik számunkra, hogy az egyes funkciókba részletesebb belátást nyerjünk. Kezdjük a legkönnyebbel a *„Kapcsolat megtekintése”*-vel



Ahogy arra kitértünk még a legelején, ez egy hagyományon statikus oldal lesz. Egy menürendszer segítségével a felhasználó kitudja majd választani ezt a funkciót és be fog töltődni a megfelelő tartalom az oldalra. Ugyan ez a funkció természetesen admin oldalról is elérhető lesz majd egy kis plusszal megfűszerezve.

Nézzünk meg egy kicsit izgalmasabbat: *„Árajánlat kérése”*.

Szituáció: A programnak le kell tudni kezelni-e a kérést ami a felhasználótól jön. Elöször is a kérést valamilyen formában tárolni kell, különben a kérések elvesznének a semmiben. Erre egy korszerű és egyben egyszerű megoldás az adatbázis. Mi a firebase adatbázis szolgáltatásait vesszük igénybe. Ha a felhasználó nem tölti ki a kötelezően kitöltendő mezőket, akkor az árajánlat kérés meghiúsul, egy hibaüzenet szólítja fel a felhasználót a hiányosságok pótlására. Ha mindent kitöltött a felhasználó, akkor nincs más dolga mint leadni a kérést. Ekkora kérés az adatbázis-ba kerül és admin oldalon elérhető lesz.

### 2.4.4 Kódolás

A mai modern alkalmazás fejlesztésénél a programozó felek felosztják egymást között a feladatokat. Ezáltal időt, energiát takarítanak meg. Ráadásul a fejlesztési folyamat is sokkal lendületesebb lesz. **De** felvetül ilyenkor egy nagyon fontos kérdés: Hogy fejlesszünk úgy, hogy kódjaink mindig szinkronba legyenek? Ne kelljen minden nap egymásnak küldözgetni az egyre növekvő alkalmazást. A válasz egyszerű. Használjunk ilyen fajta szoftvert, ami támogatja ezt a fajta fejlesztést. Az egyik legismertebb ilyen eszköz a GitHub, amit mi is használunk. Használata egyszerű. Létre kell hozni egy csoportot, amibe meg kell hívni a projektben szereplő embereket. Töltsük le a GitHub asztali alkalmazását. Nyissuk meg. Ezután file🡪 local repository. Itt beírjuk a csoport nevét és készen is vagyunk. Az adott alkalmazást, amit ide felrakunk (commit) bárki pillanatok alatt leszedheti (pull) magának. Nem is letöltésről van itt szó, hanem szinkronizálásról. A meglévő projekted kódját összehasonlítja az utoljára commit-elt kóddal, és csak a változásokat eszközöli a saját kódodban.

### 2.4.4.1 Tancsics László kódja (menürendszer*+*Rólunkoldal)

A menürendszer kialakítása egy alkalmazásban mindig kényes ügy. Mielőtt neki kezdünk össze rakni el kell döntenünk, hogy *„single page”* vagy *„multiple page”* webalkalmazást szeretnénk fejleszteni. Mi a multi page mellet döntöttünk. Tehát több lapból álló webalkalmazás menürendszerét kellet kialakítani.

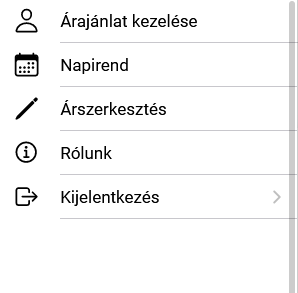
Felmerülő kérdések:

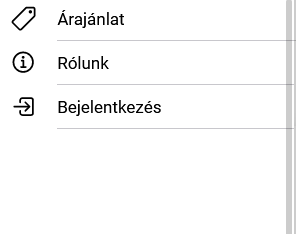
* Hogy csináljuk, hogy platform független legyen?
* Hol legyen az elhelyezkedése?
* Hogy csináljuk meg igényesre?
* Hogyan oldjuk meg az átirányításokat?

Két szó: Ionic + React. Igen egy frameworkel és egy könyvtárral nagyon egyszerűen, és gyorsan el lehet ezt érni. Lássuk hogyan!

A mi menünk egy slide menü lett. Bal oldalon helyezkedik el hamburger ikonként. A menü előhívása kétféleképpen történhet, mind mobilon, mind asztali számítógépen: Kattintással (Koppintással) vagy jobbra húzással.

Alapvetőleg 2 féle menürendszert különböztetünk meg: User és Admin oldali menürendszert.





*User oldali menü Admin oldali menü*

A menü kinézetéért az Ionic Framework a felelős. Ionic számtalan olyan előre megformázott tagekkel (IonMenu, IonMenuToggle, IonItem, IonSplitPane) rendelkezik, amivel gyorsan össze lehet hozni egy jól kinéző menürendszert. De egy jól kinéző menürendszer semmit se ér, ha nem tudja elvégezni a neki szánt feladatokat. Ez nem más, mint az irányítás.

Az irányítás elvégzésért felelős a React könyvtár. 2 lépésre van csak szükségünk:

1. lépés: Vegyük fel a gyökér fájlunk-ba (App.tsx) az összes utat (route), amit használni szeretnék (kép mutatja).



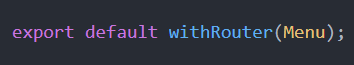
*Az összes route*

2. lépés: Egy onClick eseményt fűzünk minden egyes menüponthoz (IonItem). Majd megadjuk, hogy melyik oldal „renderelődjön”(history.push) le ha az adott IonItem-re rákattintunk. Ez biztosítja számunkra a lapok közti irányítást. Mivel mindig a megfelelő oldal renderelődik le ezért sosincs oldal frissítés, akkor mikor az egyik menüpontról a másikra ugrunk!



*Menü szerkezete+onClick esemény*

3.lépés: A react-router-dom npm modul által biztosított withRouter komponensel burkoljuk be az egész menürendszerünket.



*withRouter segítségével élesítjük a menürendszert*

**Cookie-k**

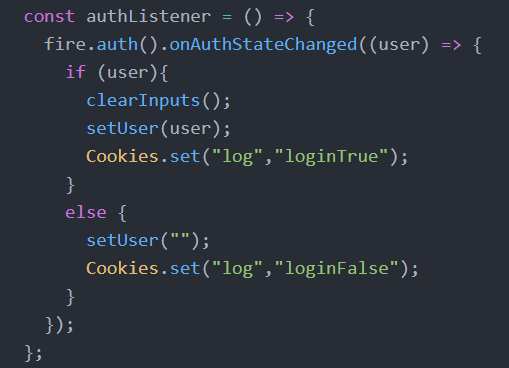
Már csak egy kérdéses rész maradt számunkra. **Hogyan különítjük el a user és az admin oldalt?** A webes technológia nagyon sok opciót ad arra, hogy hogyan kezeljük a bejelentkezett felhasználókat. Pl:. PHP:Session ID, JSON alapú rendszereknél JSON WebToken. Esetünkben csak egy felhasználót különböztetünk meg a többitől, ezért számunkra elegendő, ha ezt eltároljuk valahol. Nyílván egy olyan tárolót kellet keresnünk, amit dinamikusan, a rendszer futása közben állíthatunk + esetleges oldal frissítéseknél értéke nem veszik el. Erre a problémára egy könnyű megoldás a cookie-k használata. **Fontos** megjegyezni, hogy **csak a bejelentkezett user-nek** (esetünkben az egyetlen adminnak) **követeljük meg a cookie-k használatát**.

Telepítsük a cookie-js npm modul. Majd importáljuk be a projektunkbe.



*Cookies import*

Ezután a set metódus segítségével beállítunk egy változónak egy értéket. Ezt akkor tesszük meg, amikor a bejelentkezés sikeres volt. Innentől kezdve bárhonnan és bármikor elérhető ez a változó. Elérését a get metódus biztosítja.



*A log változó értékét loginTrue értékre változtatjuk*

*a set metódus segítségével*

### 2.4.4.2 Németh Ádám kódja (Árajánlat kérés, Login, Naptár)

A projekt részei, amelyeken dolgozott, a felhasználói autentikáció megvalósítása, az árajánlat kérés menü, illetve maga az adatbázis szerkezete és az ebbe való adatfeltöltés.

Az árajánlat kérése menü a választható munkanemeket felsoroló rádiógomb-csoportból, az adott munkakörhöz tartozó paramétereket, valamint egy e-mail címet bekérő beviteli mezőből, egy dátumválasztó naptárból és egy gombból áll, mellyel a felhasználó elküldi az árajálnat kérést az adatbázisba. Az oldalon mindig csak a kiválaszott munkaághoz tartozó paraméterek beviteli mezői jelennek meg dinamikusan, amíg egyik sincs kiválasztva, addig a „Kérem válasszon munkakört!” felirat látható a felhasználó számára. Ezt egy selected nevű állapottal és egy switch case vezérlési szerkezettel lett megvalósítva meg. Az adatbázisba beküldött kérésekhez szintén csak a releváns paraméterek kerülnek be mezőkként, az előzőhöz hasonló módon megoldva. A határidő bekéréséhez az ingyenesen elérhető React-Calendar package-et használtuk, amely a kiválasztott dátumot egy Date típusú állapotban tárolja el, az adatbázisba pedig timestampként kerül be.

A felhasználói autentikáció megvalósítását a Firebase rengeteg előre elkészített, könnyen használható funkcióval segíti. A bejelentekeztetéshez is többféle módszert nyújt, mint pl. az e-mail/jelszó, telefonszám, Facebook-fiók, stb., mi azonban egyelőre csak az e-mail-jelszó párossal való autentikációt választottuk ki. A Kódban, ezek bekérése két input mezőre, valamint egy gombra volt szükség. Erre rákattintva az alkalmazás elküldi az e-mail címet és a jelszót az adatbázis számára, ami pedig már létező felhasználó és helyes jelszó esetén elvégzi a bejelentkeztetést. Hiba esetén, mint pl. helytelen formátumú e-mail cím, nem létező felhasználó, letiltott felhasználó, helytelen jelszó, a bejelentkeztetést kezelő függvény a passwordError vagy az emailError állapotnak átadja a hibaüzenetet, ez pedig egy-egy, a beviteli mezők alatt lévő bekezdésben jelenik meg a felhasználó számára. A hibakezelő függvények szintén a Firebase beépített részei, használatukhoz csak meg kell hívnunk az adatbázishoz tartozó auth, illetve signInWithEmailAndPassword függvényeket, utóbbinak paraméterként átadva az e-mail címet és a jelszót tároló állapotokat, majd catch-elnünk és kezelnünk az általuk dobott hibát. Sikeres bejelentkezés esetén a felhasználó e-mail címe a user nevű állapotban tárolódik el és megjelenik előtte az üdvözlő oldal. Ezt egy, a user értékét vizsgáló elágazással valósítottuk meg. A user állapot alapvetően egy üres string. Amíg ez az állapot áll fent, addig a bejelentkező menü renderelődik az oldalra. Sikeres autentikáció után a Naptár menüpont jelenik meg.

### 2.4.4.3 Abafalvi Balázs kódja (Admin felület)

Admin oldal 2 szegmensből épül fel. Egyrészt itt érheti el a munkavállaló az árbeállításait. Másrészt itt vállalhat, utasíthat el és tehet várakozóra felhasználói kéréseket. Az árbeállítások egy listában tartalmazzák a külömböző munkák költségeit melyeket az admin saját igényei szerint állíthat. Ezt a firebase firestore segítségével tároljuk szerveroldalon. Ezek az adatok egy egyszerű json fileban vannak sorrendben eltárolva így könnyen és gyorsan elérhetjük őket. Árak változtatása esetén az oldal frissül és a megfelelő adatok mind az adatbázisba mind a képernyőn frissülnek. A második funkciója az admin oldalnak az árajánlatok elbírálása. Ez szintén a firestoreban eltárolt requesteket olvassa ki és helyezi el a képernyőre melyeket gestureok (jobb, bal swipe vagy double click/tap) segítségével tudunk elbírálni. Elbírálás frissíti az adatbázist: elfogadás esetén elfogadottá teszi, várakoztatás esetén változatlanul hagyja, elutasítás esetén törli a bejegyzést. A változatlanul hagyott requestek visszakerülnek a lista végére így ha végigért a legfrissebb kéréseken újra elbírálhatja az adott kérést vagy újra halaszthatja a döntést.

**Firebase hosting**

Firebase másik előnye az általuk szolgált hostolási lehetőség. Könnyen használható és meglehetősen sok funkcióval rendelkezőszolgáltatás. Lehetővé teszi a komplexebb projektek gyors elérését és könnyű menedzselését pár egyszerű paranccsal.

**On hold komponensek**

Távolságmeghatározásra használható google distance matrix api is implementálva van, de oldalon használata még nem elérhető. A felhasználó és munkavállaló közti távolság kiszámítására ad lehetőséget melyet későbbiekben implementálni tudunk, így a munkáltató tisztában van a kérdéses munka minél több paraméterével.