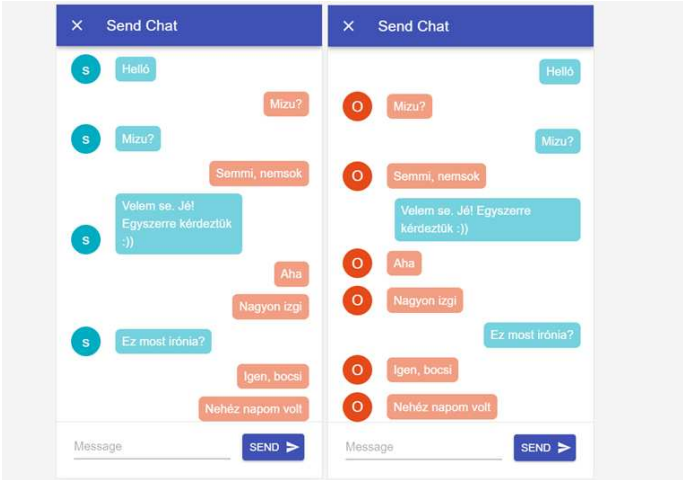
GraphChat

INeedMoreFreeTime

Laczkó Örs
Felkészítő tanár: Kovács Irén
Kerék Általános Iskola és Gimnázium, Budapest.

1. Bevezetés

Egy korábbi versenyre készített adatvizualizációs programom szolgált inspirációként a GraphChat ötletéhez. Az alapgondolat egy olyan weboldal volt, ami egy számítógépen tárolt .txt kiterjesztésű file-t, vagy egy szövegmezőbe írt számsorozatot olvas be, majd egy animációval megjeleníti, így segítve azokat, akik hosszú távon szeretnének valamilyen tevékenységet mérni (pl.: testmozgás, kalóriabevitel, testmagasság, testtömeg). Azonban hamar be kellett lássam, hogy nem túl felhasználó barát, hogy az egyetlen mentési mód egy programon kívüli file szerkesztése. Mivel a weboldal fejlesztése során még nem lehetett közvetlenül a számítógépen tárolt fájlokba tartalmat menteni, más megoldás után kellett néznem. Így jött a képbe a Google Cloud platformra épülő Firebase, a maga számtalan és folyton bővülő - kezdetben ingyenes- micro service-eivel, ahol a legmodernebb biztonságtechnikákkal vannak ötvözve különböző szerver oldali szolgáltatások (pl.: adatbázisok, file elosztók, hosting, stb.). A Firebase különböző szolgáltatásai óriási inspirációként hatottak, hogy hogyan bővíthetném az alapötletet. Ahogy egyre több micro service-szel ismerkedtem meg, úgy jöttek az ötletek, hogy mivel bővíthetném a programot. A jelenlegi elképzelésem, egy olyan webes applikáció, ahol az azonos érdeklődési körű felhasználók megoszthatnak egymással hasznos tippeket történeteken keresztül (pl.: hogy mit hol érdemes megvenni), tarthatják egymással a kapcsolatot (az applikációba épített csetnek köszönhetően - 1. ábra), vagy megoszthatnak egymással fontos adatokat privát üzenetben vagy csoportos üzenetben (pl.: egy edzőteremben, egy edzőcsoport a heti teljesítményüket). Emellett még számos apró beépített funkciót valósítottam meg, vagy vár még megvalósításra.



1. ábra: Az applikációba épített cset

2. Probléma megoldásának menete

2.1 React

Probléma megoldásának menete Mikor az alapötletből kiindulva elkezdtem a GraphChat-et fejleszteni, hamar rájöttem, hogy ekkora mennyiségű HTML-kódot összefésülni mind a szerver- mind a kliensoldali logikával rendkívül nehéz lenne. Nem beszélve arról, hogy mivel webes applikációról van szó, fontos, hogy az URL az adott tartalomra mutasson, és ez vagy azt jelentené, hogy az applikáción belül minden új felhasználói felületnél lenne egy plusz 1-2 másodperces várakozási idő, amíg a szükséges file-ok megérkeznek a szerverről, majd egy újabb 1-2 másodperces várakozási idő (készüléktől függő) amíg a tartalmat a böngésző rendereli, vagy azt, hogy egy külön JavaScript könyvtárat kéne ahhoz használnom, hogy az URL megváltozásakor, amíg az oldal domain nevén van a felhasználó, megakadályozza a re-renderelést, és ez után a DOM (document object model) felülírásával változtassa meg a felhasználói felületet. Egyik variációt se szerettem volna, így döntöttem végül egy JavaScript framework, - a Facebook által fejlesztett – ReactJS (2. ábra) mellett, ami amellett, hogy saját routing-gal rendelkezik - így megoldva az URL-s problémám -, úgy nevezett RxJS (Reactive Extensions for JavaScript) szintaxot használ, ami egyrészt lehetővé teszi a felhasználói felület komponensekre bontását, másrészt lehetőséget biztosít, arra, hogy közvetlenül a HTML kódba lehessen logikát és JavaScript változókat használni. További előny, hogy a komponensekre bontott felhasználói felületnek a részeit újra fel lehet használni, (például elég egyszer

implementálni az avatár kódját, és azt fel lehet használni mind a történetek, mind a cset beszélgetések során). A ReactJS jelenlegi verziója úgynevezett Virtual DOM-ot használ, ami azért előnyös, mert ha új tartalmat kell megjeleníteni - mert például érkezik egy új üzenet - először a Virtual DOM-ot frissíti, majd összehasonlítja a Virtual DOM-ot a HTML DOM-jával, és csak azt a részt frissíti, ami eltér, ezzel jelentősen csökkentve a gép terhelését.

2. ábra: ReactJS

2.2 Material-UI

A felhasználói felület fejlesztéséhez, a native hatás elérés érdekébe egy ReactJS-re épülő keretrendszert, a Material-UI-t használtam. A Material-UI egy nyílt forráskódú, előre definiált komponensekből álló könyvtár, amely keretrendszer-szerű funkciókkal rendelkezik (pl.: Hook API, CSS stílusokhoz).

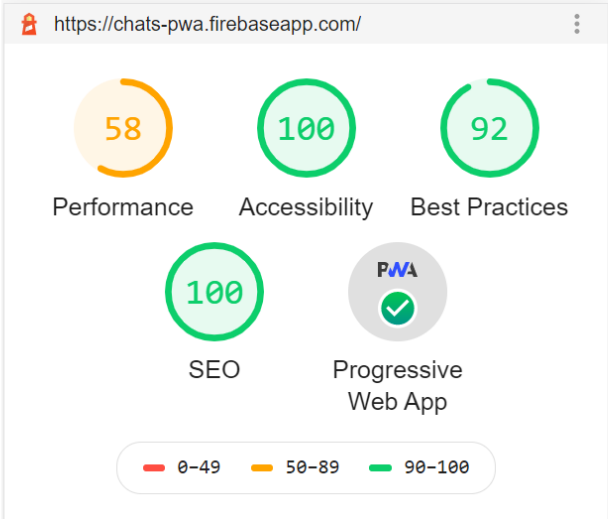
2.2 Firebase

A szerver oldali kiszolgáláshoz és a weboldal hostinghoz Firebase-t (3. ábra) használtam. A program jelenlegi állapotába számos Firebase szolgáltatást vesz igénybe: Firebase Authentication, a felhasználók bejelenetkeztetéséhez és kliens és szerver oldali megjegyzéshez. Firebase Realtime Database, a gyakran olvasott és írt adatok tárolásához (pl.: cset üzenetek, grafikonok), Firebase Firestore, a ritkábban olvasott, nagyobb tárhelyigényű adatok tárolásához (pl.: felhasználó profil), Firebase Analytics, a felhasználói élmény javításához és még sok mást.

3. ábra: Firebase

2.3 Service Worker

Azért, hogy teljes képernyős, native-hatású applikációként, kezdőképernyőről indítva is lehessen használni, Service Worker-t használtam, így az applikáció PWA (progressive web app) lett. (4. ábra)



4. ábra: Lighthouse PWA Analysis Tool-lal készített report az applikációról.

3. Elért eredmények

A felhasználó az applikáció jelenlegi állapotában képes cset üzeneteket küldeni és fogadni, valamint saját sztorit posztolni és másokét megnézni. Jelenleg a grafikonok mentését és vizualizálását implementálom react-chartist segítségével. Mivel egy egy rész implementálása rendkívül időigényes, az applikáció még közel sincs kész. Bár már több hónapnyi kemény munkám van benne, még mindig rengeteg ötletem van arra vonatkozóan, hogy mivel tudnám bővíteni. Amellett, hogy új funkciókkal szeretném bővíteni, a meglévőket is szeretném továbbfejleszteni: átláthatóvá és könnyen fenntarthatóvá tenni. Szeretném emellett a jelenlegi biztonsági szabályokat megerősíteni az adatbázisokon és a legtöbb kliensoldali adatbázissal kapcsolatos függvényt áthelyezni Firebase Cloud Functions-ba , szintén biztonsági okokból.