Mobilna aplikacija za poticanje aktivnog sudjelovanja studenata u nastavi

Markić, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA APLIKACIJA ZA POTICANJE AKTIVNOG SUDJELOVANJA STUDENATA U NASTAVI

Završni rad

Luka Markić

Osijek, 2022.

SADRŽAJ

1. UVOD

Poticanje studenata i interaktivno sudjelovanje u nastavi oduvijek je bilo jedan od glavnih ciljeva unaprjeđenja nastave. Razvojem tehnologije, posebice mobilne, pruža se prilika unaprjeđenja nastave primjenom mobilnog učenja koje pruža razne prednosti poput prenosivosti, jednostavne upotrebe i jednostavnog pristupa informacijama. Cilj ovog rada je odrediti zahtjeve za razvoj aplikacije u poticanju studenata na aktivnije sudjelovanje u nastavi i realizacija takve aplikacije. S tim ciljem razvoj je oslonjen na tehnologije uvjeravanja i psihologiju društvenog utjecaja.

Pristup mobilnoj aplikaciji je moguće ostvariti na tri načina: kao administrator, predavač i student pri čemu je funkcija administratora stvaranje kolegija, računa predavača, dodjela kolegija predavaču i drugo. Predavač pruža sadržaj, postavlja provjere znanja i domaće zadaće te ocjenjuje iste. Student je taj koji odabire kolegije i pristupa postavljenom sadržaju. Budući, kako svaki korisnik ima vlastite afinitete, ali i različiti proces učenja, izvršena je prilagodba sadržaja računu korisnika. Student rješavanjem testova, domaćih zadaća i slično ostvaruje odgovarajući uspjeh koji je popraćen odgovarajućom povratnom informacijom vodeći se smjernicom dijaloga podrške. Odnosno, student zaprima različitu poruku ili prijedlog, pozitivnu ili negativnu, završetkom provjere. Oslanjajući se na tehnologiju samokontrole i nadzora ugrađen je statistički pregled uspjeha budući kako društveni utjecaj usporedbe može studente potaknuti na završetak određenog zadatka, kao i ostvarenje boljih rezultata. U konačnici, oslanjajući se na tehnologiju prijedloga, korisnik biva pravovremeno obaviješten putem obavijesti (engl. *notification*) o postavljenoj provjeri znanja ili domaćoj zadaći. Također, ta obavijest biva postavljena kao napomena u kalendaru pregleda obveza unutar kojeg je omogućen unos vlastitih obveza.

1.1. Zadatak završnog rada

U teorijskom dijelu rada potrebno je proučiti i opisati zahtjeve za razvoj aplikacija za poticanje studenata za aktivnije sudjelovanje u različitim oblicima nastave (predavanja, laboratorijske, auditorne i konstrukcijske vježbe) i tehnologije za izradu mobilnih aplikacija koristeći Flutter razvojno okruženje. U praktičnom dijelu rada potrebno je izraditi mobilnu aplikaciju koja u sučelju omogućuje studentima uvid u sve oblike zadataka koje im je postavio nastavnik (pitanja, zadaće, zadaci, testovi i sl.) po pojedinom kolegiju te rješavanje istih. Također, studenti trebaju imati uvid u bodovanje i prikaz statistike i vlastite aktivnosti na kolegiju, pretraživanje i sortiranje sadržaja te u informacije o rokovima i obvezama.

2. ZAHTJEVI RAZVOJA MOBILNE APLIKACIJE ZA POTICANJE AKTIVNOG SUDJELVANJA I KORIŠTENE TEHNOLOGIJE

2.1. Mobilno učenje

Kako sposobnost studenta nije određena samo njihovom intelektualnom sposobnosti, već i drugim faktorima kao što su: radne navike, tehnike učenja, kognitivno učenje i rješavanja problema [1]. Kao jedan način pospješivanja sposobnosti studenta i njegove motivacije je korištenje mobilnih uređaja odnosno primjena mobilnog učenja. Budući da su mobilni uređaji dio naše svakodnevnice to pruža jednostavnu upotrebu u edukacijske svrhe. Početkom prošlog desetljeća povećanjem dostupnosti upotreba pametnih uređaja je se znatno povećala. S tim povećanjem povećala je se i njihova upotreba u visokom obrazovanju, njihova privlačnost vezana je uz mnoge faktore, od čega su: prenosivost, lagan i jednostavan pristup informacija s različitih lokacija u bilo kojem trenutku [2]. Mobilno učenje je moguće definirati kao tip učenja gdje se predavači i učenici ne vežu uz točno, unaprijed određenu lokaciju, a pri tome koriste prednosti mobilne tehnologije ili bilo koji tip učenja koji primjenjuje bežičnu tehnologiju i/ili mobilni uređaj [3]. Mobilno učenje ne može zamijeniti klasični način podučavanja, ali pruža alternativu učenju izvan učionice i povećava interakciju [4]. Primjenom mobilnih uređaja omogućuje se pristup u bilo kojem trenutku bilo gdje jednostavnim pristupom internetu. Pristupačnost sustava može potaknuti predavače i učenike na komunikaciju, razmjenu informacija, koja se sastoji od učenja, raznih aktivnosti i standardnog elektronskog učenja (engl. *E-learning*) [5]. Postavlja se pitanje koliko primjena mobilnih aplikacija zapravo utječu na akademski uspjeh i na koji način je potrebno dizajnirati aplikaciju kako bi se ostvarilo proaktivno sudjelovanje.

2.1.1. Utjecaj mobilnog učenja na akademski uspjeh

Na sveučilištu Dokuz Eylul u Turskoj provedeno je istraživanje koje je imalo za cilj utvrditi koliki je zapravo utjecaj mobilnog učenja na akademski uspjeh. Istraživanje je provedeno nad 36 ispitanikom, od čega je njih 15 dobilo mobilni uređaj, a 21 nije. U oba slučaja kolegiji su bili podijeljeni 50% na teorijsku i 50% na praktičnu komponentu. Sadržaj (prezentacije, video, audio, domaće zadaće i drugi) za obje grupe bio je dostupan putem sustava za upravljanje učenjem (putem mrežnog pretraživača), pri čemu je prva grupa koristila dobivene mobilne uređaje. Usporedba rezultata obje grupe je obavljena koristeći test sume rangova poznatija kao Mann-Whitney U metoda. Test sume rangova je neparametarski test za provjeravanje odgovaraju li dva uzorka

populaciji s istim medijanom [6]. Provjera je obavljena u tri stadija: prije, poslije i 6 mjeseci nakon istraživanja u svrhu utvrđivanja postojanosti rezultata. Kao što je vidljivo iz dolje prikazane tablice ne postoji veliko odstupanje prije istraživanja, no nakon provedenog istraživanja odstupanje je značajno i to u korist mobilnog učenja [7].

Grupa	Test	Broj ispitanika	Prosjek	Zbroj	Rang	Sig.
I.	Prije	15	25.20	378.00	132.00	.086
II.	istraživanja	26	18.58	483.00		
I.	Poslije	15	31.13	467.00	43.00	.000
II.	istraživanja	26	15.15	394.00		
I.	Nakon 6.	15	28.67	430.00	80.00	.002
II.	istraživanja	26	16.58	431.00		

Tablica 2.1. Prikaz utjecaja primjene mobilnog učenja na akademski uspjeh primjenom testa sume rangova [7]

Time možemo zaključiti kako primjena mobilnih uređaja u edukacijske svrhe osim na povećanje motivacije utječe i na akademski uspjeh. Nadalje, osim samog akademskog uspjeha cilj je potaknuti studente na aktivno sudjelovanje i odrediti način na koji je to moguće ostvariti.

2.1.2. Smjernice oblikovanja mobilne aplikacije

Određivanje osnovnih smjernica razvoja mobilne aplikacije za poticanja aktivnog sudjelovanja studenata ostvareno je osloncem na tehnologije uvjeravanja. Prema tehnologijama uvjeravanja (engl. *Persuasive Technologies*) potrebno je voditi brigu o nekoliko tehnologija odnosno načina dizajna kojim se potiče na upotrebu mobilne aplikacije.

Određivanje smjernica razvoja temelji se na sljedećim tehnologijama uvjeravanja:

- Tehnologija tuneliranja (engl. *Tunnelling technology*), predstavlja unaprijed određeni slijed aktivnosti ili događaja koji slijedno vode korisnika. Primjerice, ako je prijava (ili stvaranje korisničkog računa) dobro dizajnirana ona će potaknuti korisnika na završetak prijave, u suprotnom ako je kompleksna i neintuitivna može dovesti do neželjenog učinka.
- Tehnologija kroja (engl. *Tailoring technology*) zasnovana je na prilagodbi sadržaja korisniku.
- Tehnologija prijedloga (engl. *Suggestion technology*) pruža prijedloge, primjerice skočni prozor koji upozorava korisnika na poduzimanje određene radnje (želimo li spremiti, izbrisati dokument i sl.).

- Samokontrola (engl. *Self-monitoring*) potiče korisnika na ostvarenje određenog cilja, primjerice ukoliko postoji jasan pregled sadržaja i rok predaje projekta, domaće zadaće i slično veća je vjerojatnost kako će ona biti predana pravovremeno.
- Tehnologija nadzora (engl. *Surveillance technology*) pruža korisniku pregled vlastitog uspjeha, ali i uspjeha drugih.
- Uvjetna tehnologija (engl. *Conditional technology*) zasnovana na nagradnom konceptu, što može utjecati na promjenu pristupa i načina ponašanja [8] [9].

Osim navedenih tehnologija veliku ulogu u procesu poticanja imaju dijalog podrške i društveni utjecaj na kojim su neke od tehnologija zapravo zasnovane. Dijalog podrške može biti u obliku pohvale, podsjetnika, izgleda i društvene uloge. Pohvale trebaju ovisno o uspjehu korisnika rezultirati različitom povratnom informacijom, dok podsjetnik treba potaknuti korisnika pravovremenim i redovitim obavijestima. Aplikacija treba imati pristupačan i jednostavan izgled, ali sama aplikacija treba preuzeti društvenu ulogu odnosno biti vodilja.

Gledano sa strane društvenog utjecaja primjenjuje se sljedeće:

- Društveno učenje (engl. *Social learning*), osoba je u mogućnosti korištenjem mobilnog uređaja uočiti ponašanje drugih korisnika. Ovo može motivirati korisnika na ostvarenje cilja primjerice samom željom za sudjelovanjem.

- Društvena usporedba (engl. *Social comperison*), korisnik usporedbom vlastitih rezultata s primjerice prosjekom ostalih može biti potaknut na unaprjeđenje svog daljnjeg uspjeha.

- Utjecaj norme (engl. *Normative influence*), mobilna aplikacija može potaknuti korisnika na ostvarenje željenog cilja pružanjem određene norme koje je potrebno ispuniti, npr, ponovno pokretanje kviza dok se ne postigne željeni rezultat.

- Natjecateljski duh (engl. *Competition*), kao što je ranije navedeno mobilna aplikacija može potaknuti na natjecateljski duh (npr. usporedbom rezultata).

- Priznanje (engl. *Recognition*), mobilna aplikacija može potaknuti korisnika na aktivno sudjelovanje nekim oblikom priznanja, odnosno ranije navedena poruka pohvale.

Za utvrđivanje valjanosti navedenih teorija provedeno je istraživanje korištenjem mobilne aplikacije za poticanje aktivnog sudjelovanja u nastavi LevelUp, gdje su neke od povratnih informacije prikazane tablicom 2.2. [10].

Postavljeno pitanje	Povratna informacija
Je li mislite da možete biti potaknuti dizajnom aplikacije na sudjelovanje u nastavi?	Većini ispitanika sami dizajn nije igrao veliku ulogu, ali je svakako predstavljao prednost.
Je li mislite je li osoba ili ljudi bolje motiviraju od tehnologije?	Većina je odgovorila u korist ljudi, odnosno većina smatra kako bi korištenje tehnologije dovelo do socijalne interakcije i negativno utjecalo na mentalno zdravlje
Koje sve aplikacije koriste izvan škole u edukacijske svrhe?	Neki od odgovora su bili: Ebooks, YouTube, općenito Internet i Gmail.

Tablica 2.2. Prikaz povratnih informacija primjerne mobilne aplikacije LevelUp [10]

Prema istraživanju zaključeno je kako korisnicima potreban jednostavna dizajn s popratnim sadržajem kao što su dokumenti, video, audio zapisi, poveznice i slično. Također važnu ulogu igra i komunikacija primjerice slanje elektroničke pošte profesoru i drugima. Mobilna aplikacija ne predstavlja primarni oblik obrazovanja, ali svakako može povećati motivaciju studenata.

Nakon obrađenog moguće je zaključiti kako za razvoj aplikacije se moraju ispoštovati sljedeći zahtjevi: aplikacija mora obavljati korisne zadatke, sadržaje treba biti prilagođen prema korisniku, potrebno je omogućiti prikaz vlastitih uspjeha i uspjeha drugih, primijeniti dijalog podrške, pružiti obavijesti o odgovarajućim obvezama, omogućiti pregled tih obveza i pružiti jednostavan i intuitivan dizajn.

2.2. Korištene tehnologije

Za razvoj mobilne aplikacije kao razvojno okruženje korišten je Android Studio, aplikacija je napravljena koristeći komplet za razvoj softvera Flutter, odnosno objektno orijentirani programski jezik Dart. Budući da aplikacija omogućuje prijavu korisnika i pohranu sadržaja kao uslugu za pohranu podataka korištena je usluga Firebase.

2.2.1. Android Studio kao razvojno okruženje

Android Studio je integrirano razvojno okruženje (engl. *Integrated Development Environment* - IDE) za razvoj Android aplikacija, temeljeno na IntelliJ IDEA. Dok IntelliJ IDEA predstavlja integrirano razvojno okruženje napisano u Java programskom jeziku za razvoj računalne programske podrške napisane u programskim jezicima poput Java, Kotlin, Groovy i ostalim programskim jezicima temeljenih na JAR formatu. Povrh IntelliJ moćnog uređivača koda i alata za razvoj programa, Android Studio nudi još više značajki koje olakšavaju izradu aplikacije, neke od njih su: brz emulator, ujednačeno okruženje razvoja za sve Android uređaje i promjena koda

bez potrebe ponovnog pokretanja aplikacije [11] [12]. Android Studio od verzije 2020.3.1 (Arctic Fox) pruža potpuno integrirano razvojno sučelje za Flutter i za Dart programski jezik. Nakon instalacije razvojnog okruženja Android Studio potrebno je instalirati Flutter i Dart priključke što će omogućiti prepoznavanje Dart sintakse, pružanje prijedloga i druge mogućnosti [13].

2.2.2. Dart programski jezik

Google prvi put predstavlja Dart programski jezik na GOTO konferenciji 2011. godine, dok prva konačna verzija Dart 1.0 izlazi koncem 2013. godine. Prvotno je gledan ako zamjena JavaScript jeziku, ali nije bio dobro prihvaćen u programerskoj zajednici. Međutim, zahvaljujući razvoju Fluttera i njegovog oslonca na Dart, njegova upotreba se značajno povećala. Razlog zašto Flutter koriti Dart je taj što poboljšava proces razvoja mobilnih aplikacija i omogućuje alternativu visoke izvedbe postojećim više-platformskim okvirima. Osim toga Dart pruža fleksibilnost kada je u pitanju njegovo prevođenje (engl. *compilation*) pružajući prevođenje prije izvođenja (engl. *ahead-of-time* - AOT) i prevođenje točno na vrijeme (engl. *just-in-time* - JIT). Također omogućuje visoke performanse, a budući da podržava AOT način prevođenja Flutter ne zahtjeva primjerice, čekanje na pretvorbu nenativnog Flutter koda u nativni čime se omogućuje bolji odaziv aplikacije i njezino brže pokretanje. Također sadrži skupljač smeća (engl. *Garbage collector*), te svojom fleksibilnosti, otpornosti i modernim pristupom olakšava učenje [14].

```
void printPersonsData(Person person) {
  print('Osoba: ${person.toString()}'); // Ispis na konzolu.
}
void main() {
  Person person = Person(name: 'Luka', surname: 'Markic');
  printPersonsData(person);
}
class Person{

  String name;
  String surname;

  Person({this.name = '', this.surname = ''});

  @override
  String toString (){
    return '$name $surname';
  }
}
```

Programski kôd 2.1. Prikaz jednostavnog Dart programskog koda s klasom Person i metodom za ispis atributa te klase

Programski kod 2.1. prikazuje jednostavni Dart programski kod kojim je prikazan način pisanja metoda, ispisa te budući da je Dart objektno orijentirani programski jezik omogućeno je stvaranje klasa s pripadajućim podatkovnim članovima.

2.2.3. Flutter kao razvojni softver

Flutter je Googleov prijenosni okvir korisničkog sučelja (engl. *User Interface Framework*) koji služi za razvoj modernih, nativnih, reaktivnih aplikacija za iOS i Android platformu. Također pruža podršku za internetski preglednik (Hummingbird) i za ugradbene uređaje (Raspberry Pi, automobile, kućanske uređaji i druge). Otvorenog je koda koji je objavljen na GitHub platformi gdje njegovom razvoju najviše doprinosi Google, ali je također podržan od zajednice. Oslanja se na Dart programski jezik, a kao pogon obrade (engl. *Rendering engine*) koristi Skia 2D koja radi na različitim vrstama sklopovlja i programske podrške, a također je korištena u raznima internetskim preglednicima (Google Chrome, Mozilla Firefox i drugi). Jedna od funkcionalnosti Fluttera je ta što koristi Dart kako bi stvorio vlastito korisničko sučelje čime nema potrebe za korištenjem odvojenih jezika kao što su označni jezici ili vizualni dizajneri. Flutter gradi izgled aplikacije tako da reflektira stanje aplikacije, odnosno kada se stanje (podatak) promijeni izgled je ponovno izgrađen. Ono što je glavno obilježje razvojnog softvera Flutter su male aplikacije (engl. *widgests*). One predstavljaju osnovne gradivne jedinice te njihovim kombiniranjem se stvaraju objekti i daje se ponašanje tim objektima te se stvara stablo malih aplikacija (engl. *Widget tree*). Postoje dvije osnovne vrte malih aplikacija StatelessWidget i StatefulWidget. StatelessWidget se korist kada nema promjene stanja, u suprotnom se koristi StatefulWidget [15].

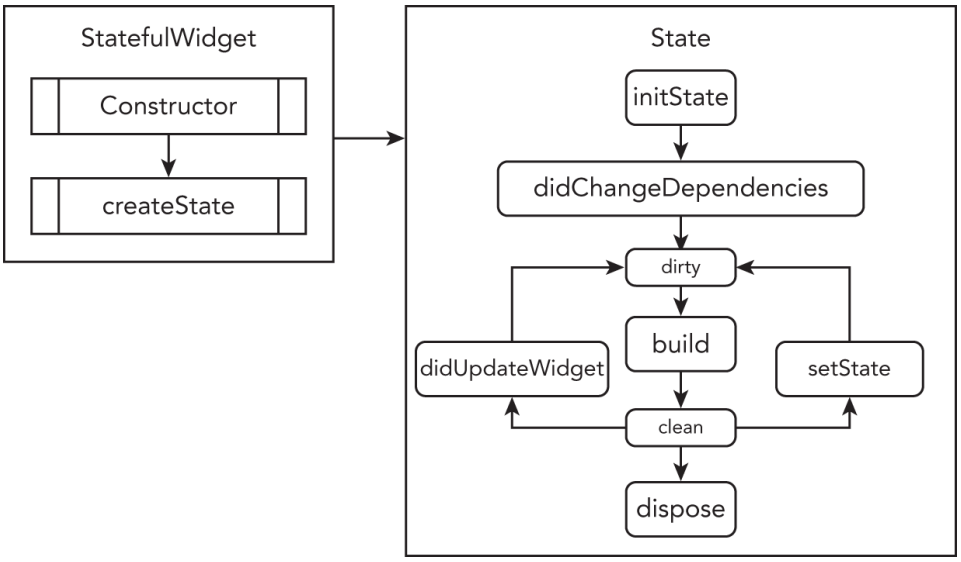
Prikazani programski kodovi 2.2. i 2.3. prikazuju razliku između StatelessWidgeta i StatefulWidgeta *TextBox*. StatelessWidget koristi se za prikazivanje nepromjenjive tekstualne vrijednosti, dok se StatefulWidget koristi za prikaz promjene varijable *text*, pritiskom gumba varijabla mijenja svoju vrijednost. Kako bi se osigurao prikaz te promjene na zaslonu dodatno koristimo metodu *setState* koja će obavijestiti radni okvir kako je došlo do promjene varijable koja može utjecati na korisničko sučelje što će rezultirati ponovnim stvaranja zaslona.

```
class TextBox extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.all(5),
      child: const Text ('Nepromjenjivi sadržaj'),
    );
  }
}
```

Programski kôd 2.2. Prikaz StatelessWidgeta TextBox

```
class TextBox extends StatefulWidget{
  @override
  State<TextBox> createState() => _TextBoxState();
}

class _TextBoxState extends State<TextBox> {

  String text = 'Početna vrijednost';

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Container(
          padding: const EdgeInsets.all(5),
          child: Text(text),
        ),
        ElevatedButton(onPressed: (){
          setState(() {
            text = 'Nova vrijednost'; //Promjena vrijednosti
          });
        }, child: const Text('Promjeni vrijednost'))
      ],
    );
  }
}
```

Programski kôd 2.3. Prikaz StatefulWidgeta TextBox s prikazom primjenjive vrijednosti atributa text



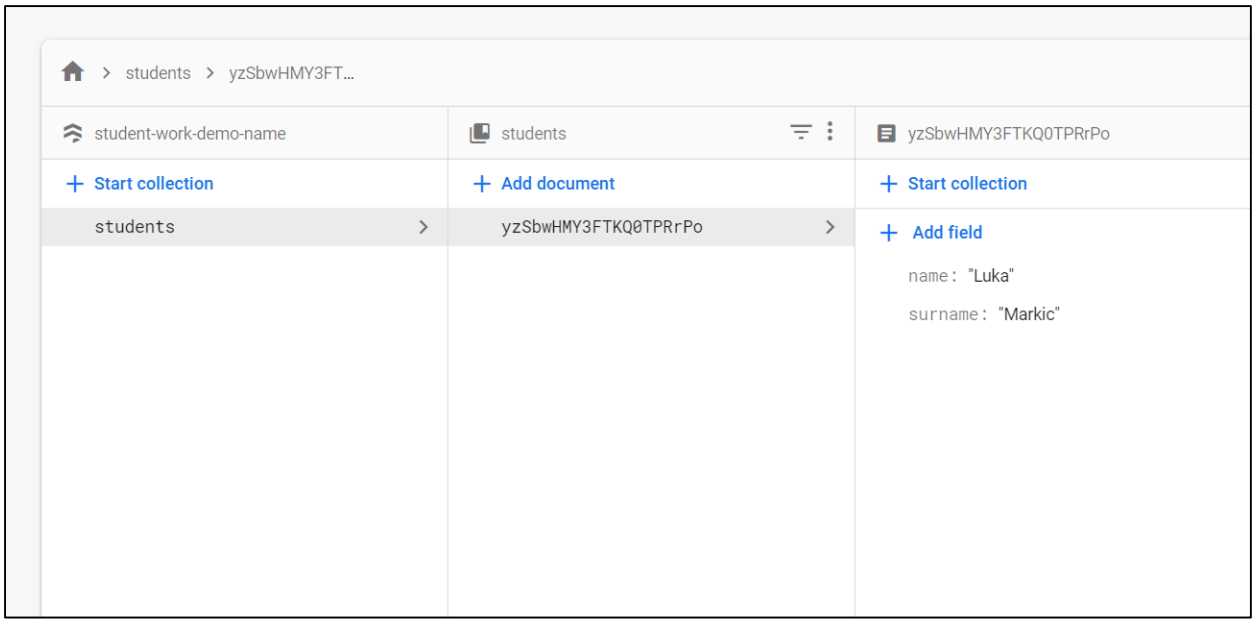
Slika 2.1. Prikaz životnog ciklusa StatefulWidgeta [16]

Gore prikazana slika 2.1. prikazuje životni ciklus StatefulWitgeta. Nakon što se stvori novi StatfulWidget poziva se obvezna metoda *createState* koja vraća instancu stanja. Pri prvom stvaranju poziva se metoda *initState* koja se pokreće prije nego je mala aplikacije stvorena. Nadalje, poziva se metoda *didChangeDependecies* za provjeru promjena ovisnosti, nakon čega slijedi metoda *build* koja zapravo vrši obradu svih malih aplikacija ovisnih o njoj. Metode *didUpdateWidget* i *setState* obavještavaju radni okvir kako je došlo do promjene time što označe

malu aplikaciju kao prljavu (engl. *dirty*). Metoda *dispose* zapravo predstavlja suprotnost od metode *initState* koja se poziva kada je potrebno objekt i njegovo stanje ukloniti iz postojećeg stabla malih aplikacija [16]. Osim ove dvije male aplikacije postoje mnoge druge poput: unosa, izgleda, dodataka, slika, ikona, tekstualnih, stilističkih elemenata i drugih. [17].

2.2.4. Firebase kao usluga za pohranu podataka u oblaku

Firebase je platforma za razvoj mobilnih aplikacija koju koristi preko milijun tvrtki diljem svijeta. Objavljena je u travnju 2012. godine i zbog svojih suvremenih sposobnosti i tehnologija 2014. godine kupila ju je tvrtka Google LLC. To je zapravo platforma pozadinske podrške kao usluge (engl. *Backend-as-a-Service* - BaaS) koja je namijenjena razvoju aplikacija pružajući pozadinske usluge pohrane kao što su: baze podataka u stvarnom vremenu, pohrana u oblaku, autentifikacija, strojno učenje, obavijest o padu sustava, udaljeno upravljanje i spremanje statističkih podataka. Firebase autentifikacija pruža sigurnu prijavu podržavajući anonimnu prijavu te prijavu putem elektroničke pošte i zaporke, telefonskog broja i putem Google, Apple, Microsoft, Yahoo, Twitter, Facebook i GitHub računa [18] [19]. Svaki korisnik dobiva jedinstveni ključ preko kojega ih je moguće razlikovati čime se osigurava jedinstvenost i omogućuje se prilagođavanje aplikacije računa korisniku. Osim autentifikacije Firebase pruža i bazu podatak u oblaku (Cloud Firestore), koja koristi fleksibilnu, prilagodljivu NoSQL bazu podataka u oblaku za spremanje i sinkronizaciju podataka korisnika i omogućuje razvoj na strani poslužitelja [20]. Sljedeća slika prikazuje primjer Cloud Firestore baze podataka s pregledom sadržaja kolekcije *students*.



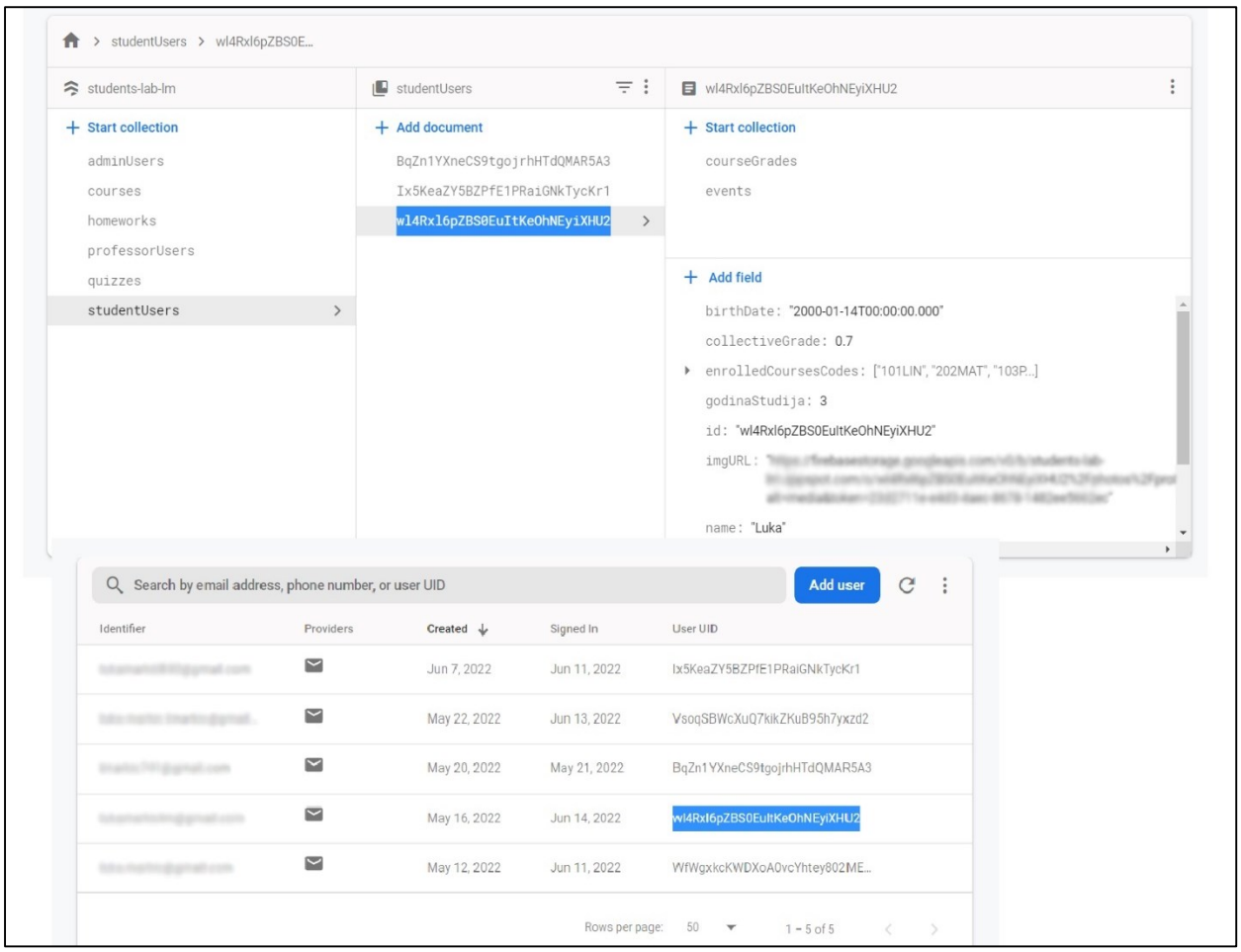
Slika 2.2. Prikaz Cloud Firestore baze podataka s preglednom na sadržaj kolekcije students

Budući kako je Cloud Firestore NoSQL dokumentno orijentirana baza podataka, podaci nisu raspoređeni u redove i stupce kao kod klasične SQL baze podataka. Podaci se spremaju o obliku dokumenata i kolekcije, pri čemu svaki dokument sadrži polja parova ključ-vrijednost (engl. *key-value*). Svaki dokument treba pripadati odgovarajućoj kolekciji i posjedovati odgovarajući identifikacijski ključ čime omogućujemo identifikaciju dokumenta prema njegovoj lokaciji u bazi podataka [18]. Slika 2.2. prikazuje pregled baze podataka u oblaku s kolekcijom *students* u kojoj se nalazi dokument s odgovarajućim identifikacijskim ključem, a unutar kojeg se nalazi polje parova ključ-vrijednost. Primjerice ključ prvog polja je *name*, dok je vrijednost tog ključa niz znakova „Luka“. No kako Cloud Firestore ne podržavanja spremanje datoteka poput fotografija, audio i video uradaka, dokumenata za njihovu pohranu korišten je Cloud Storage [21]. Njegovom upotrebom omogućeno je podizanje i dohvat sadržaja uz osnovne funkcionalnosti stvaranja, pisanja, ažuriranja i brisanja korištenjem jednostavnih i izravnih API-a [22].

3. RAZVOJ MOBILNE APLIKACIJE

3.1. Prijava u sustav

Budući kako je jedan od zahtjeva razvoja aplikacije prilagodba sadržaja korisniku samo je intuitivno ostvariti navedeno prilagodbom računu korisnika. U prethodnom odlomku istaknuto je kako svaki korisnik jednoznačno određen jedinstvenim ključem, što znači da je time osigurana jedinstvenost. Prijava je ostvarena upotrebom prijave elektroničkom poštom i zaporkom, pri čemu će Firebase vodi brigu o zabrani ponovne upotrebe iste adrese elektroničke pošte i stvorenom računu dati jedinstveni ključ koji je naknadno korišten u povezivanju sadržaja s računom korisnika. Jedinstveni ključ iskorišten je za dodavanje informacije o korisniku kao što su njegovo ime, prezime i druge na načina da se u odgovarajuću kolekciju NoSQL baze podataka u oblaku dodaje dokument s identifikacijskom oznakom jednakom jedinstvenom ključu korisnika.



Slika 3.1. Prikaz povezivanja podataka baze podataka u oblaku s odgovarajućim korisničkim računom preko korisnikove identifikacijske oznake

Budući kako je aplikaciji moguće pristupiti na tri načina: kao klijent (student), administrator i kao predavač, korisnici su raspoređeni u odgovarajuće kolekcije slijedno: *studentUsers*, *adminUsers* i

professorUsers. Time je olakšana njihova pretraga i omogućeno je utvrđivanje tip računa. Slika 3.1. prikazuje povezivanje podataka baze podataka u oblaku s odgovarajućim korisničkim računom, te je potrebno samo poznavati oznaku korisnika i kolekciju kojoj pripada za dohvat informacija. Preusmjeravanje na odgovarajući zaslon ostvareno je pomoću metode *getUserType* i klase *FrontPage* prikazane sljedećim programskim kodovima.

```
Future<UserType> getUserType(String? docId) async {
  try {
    var collectionRef = FirebaseFirestore.instance.collection('studentUsers');
    var doc = await collectionRef.doc(docId).get();
    if(doc.exists) {
      return UserType.student;
    } else{
      collectionRef = FirebaseFirestore.instance.collection('adminUsers');
      var doc = await collectionRef.doc(docId).get();
      if(doc.exists) {
        return UserType.admin;
      }else{
        return UserType.professor;
      }
    }
  } catch (e) { throw e; }
}
```

Programski kôd 3.1. Prikaz asinkrone metode za dohvat tipa korisnika

```
class FrontPage extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
        future: getUserType(AuthService().user?.uid),
        builder: (context, AsyncSnapshot snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const LoadingScreen();
          } else if (snapshot.hasError) {
            return const Center(
              child: ErrorMessage(),
            );
          } else {
            if (snapshot.data == UserType.student){
              return StudentFrontPage();
            }
            else if(snapshot.data == UserType.admin) {
              return AdminFrontPage();
            }else{
              return ProfessorFrontPage();
            }
          }
        }
    );
  }
}
```

Programski kôd 3.2. Prikaz klase FrontPage koja na temelju rezultata metode getUserType preusmjerava korisnika na odgovarajući zaslon

Programski kod 3.1. prikazuje jednostavnu metodu *getUserType,* koja provjerava je li postoji dokument s odgovarajućom oznakom u kolekciji čiji je povratni tip *Future<UserType>*. *UserType* je *enum* koji sadrži tri vrijednosti: *student*, *admin* i *professor*, a služi lakšem raspoznavanju. *Future* je poseban tip podatka koji je specifičan za Dart programski jezik. To je asinkroni tip podatka koji zapravo vraća buduću vrijednost, a ne statički tip podatka. Statička vrijednost dobivena je koristeći *FutureBuilder* kao što je prikazano programskim kodom 3.2.. Važna značajka koju pruža Firebase usluga je provjera je li korisnik prijavljen, na taj način se izbjegava nepotrebno i zamorno ponovno prijavljivanje korisnika pri ponovnom pokretanju i korištenju aplikacije. Za izvršenje provjere korištena je metoda *authStateChanges* nad instancom klase *FirbeaseAuth*, ako metoda vrati podatak onda je korisnik prijavljen u suprotnom će korisnik biti proslijeđen na zaslon prijave, što je prikazano programskim kodom 3.3..

```
StreamBuilder(
  stream: FirebaseAuth.instance.authStateChanges(),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const LoadingScreen();
    } else if (snapshot.hasError) {
        return const Center(
          child: ErrorMessage(),
      );
    } else if (snapshot.hasData) {
        return FrontPage();
    } else {
        return const LoginScreen();
    }
  },
)
```

Programski kôd 3.3. Prikaz provjere prijave korisnika i preusmjeravanja na odgovarajući zaslon

3.2. Pohrana i dohvat podataka iz baze podataka

Za olakšano upravljanje podatcima korištene su klase unutar kojih se nalaze metode za dohvat podatka iz baze podataka i njegovo postavljanje u bazu podataka. Primjerice, programski kod 3.4. prikazuje isječak klase *ProfileService* koja sadrži razne metode no prikazana su dva osnovna primjera metoda. Metoda *addProfileData* postavlja podatke korisnika u bazu podataka, dok metoda *getProfileDataStudent* dohvaća podatke iz baze podataka u oblaku. Za postavljanje podatka u bazu podataka potrebno ih je zapisati u JSON obliku što je tekstualni oblik zapisa ključ-vrijednost, a upravo je to oblik koji odgovara zapisu podataka (polja) dokumenta neke kolekcije baze podataka u oblaku. Budući kako postoje razni modeli odnosno klase koje je potrebno zapisati u i iz JSON oblika, za olakšani i ubrzani pristup korišten je paket JsonSerializable [23]. Gdje željenu klasu treba prepisati (engl. *override*) i unutar nje postaviti metode za prebacivanje u JSON oblik i iz takovog oblika u oblik odgovarajuće klase. Nakon što su podaci u JSON obliku, podatci

će biti postavljeni u bazu podataka jednostavnim unosom putanje pri čemu je prvo potrebno unijeti naziv kolekcije, a zatim identifikaciju oznaka dokumenta i u konačnici korištenjem metode *set* odgovarajući dokument će biti postavljen. Za dohvat podataka unesena je putanja kolekcije i korištenjem *await* naredbe budući da se radi o *Future* vrijednosti dohvaća se dokument. Zatim je dobiveni sadržaj prebačen iz JSON oblika u oblik odgovarajuće klase.

```
class ProfileService{
  final FirebaseFirestore _db = FirebaseFirestore.instance;

  Future<void> addProfileData(String collectionName, ProfileUser user) async{
    CollectionReference collectionReference =
FirebaseFirestore.instance.collection(collectionName);
    Map<String, dynamic> uploadedData = user.toJson();
    await collectionReference.doc(user.id).set(uploadedData);
  }

  Future<ProfileStudent> getProfileDataStudent(String uid) async{
    CollectionReference collectionReference =
     FirebaseFirestore.instance.collection('studentUsers');
   var documentSnapshot = await collectionReference.doc(uid).get();
   Map<String, dynamic> data = documentSnapshot.data()! as Map<String,dynamic>;
    return ProfileStudent.fromJson(data);
  }
  //Nastavak klase*****
}
```

Programski kôd 3.4. Prikaz isječka klase ProfileService

3.3. Koncept i izrada kolegija

Kolegij je osnovni element aplikacije uz koji je vezan određeni sadržaj, predavači i upisani studenti. Za oblikovanje kolegija koristi se model prikazan ispisom sljedećeg programskog koda.

```
@JsonSerializable()
class Course{
  String title; double ECTS;
  String? imgURL; int color;
  String code; int semester;
  Course(
      {
        this.title = 'Course', this.ECTS = 0,
        this.imgURL, this.color = 0Xff0000FF,
        required this.code, this.semester = 1,
      }
      );
  @override
  factory Course.fromJson(Map<String, dynamic> json) => _$CourseFromJson(json);
  Map<String, dynamic> toJson() => _$CourseToJson(this);
}
```

Programski kôd 3.5. Prikaz klase Course koja predstavlja model kolegija

Klasa *Course* sadrži sljedeće atribute pomoću kojih je opisana: jedinstveni identifikator *code*, broj semestra, naziv, broj ECTS bodova i cjelobrojna vrijednost tematske boje kolegija. Također sadrži

metode za prebacivanje iz i u JSON oblik, koje služe za zapis i dohvaćanje podataka iz baze podataka u oblaku. Za implementaciju pojedinog dijela kolegija poput predavanja, auditornih vježbi i drugih korišten je model segmenta koji je prikazan programskim kodom 3.6..

```
@JsonSerializable()
class Segment{
  String title; String code; List<String>? linkURLs;
  List<Map<String, dynamic>> documents;
  List<Map<String, dynamic>>? quizSegmentModels;
  List<Map<String, dynamic>>? homeworkSegmentModels;
  Segment(
      {
        this.title = 'Segment', this.code = '0_XXX',
        this.documents = const <Map<String, dynamic>>[],
        this.linkURLs, this.quizSegmentModels, this.homeworkSegmentModels,
      }
     );
  @override
  factory Segment.fromJson(Map<String, dynamic> json) =>
  _$SegmentFromJson(json);
  Map<String, dynamic> toJson() => _$SegmentToJson(this);
}
```

Programski kôd 3.6. Prikaz klase Segment koja predstavlja model segmenta

Kao što je vidljivo programskim kodom 3.6. model segment sadrži listu dokumenta, provjera, domaćih zadaća i poveznica čime je sadržaj povezan uz točno određen segment, a budući da je određeni segment sadržan kao dokument podkolekcije određenog kolegija omogućeno je jednoznačno određivanje sadržaja. Nadalje, za povezivanje kolegija s određenim predavačem pa tako i studentom potrebno je samo postaviti identifikaciju oznaku tog kolegija u listu niza znakova koja se nalazi unutar dokumenta baze podataka koja odgovara određenom korisniku. Za bolje razumijevanje pogledati sliku 3.1. na kojoj je prikazan korisnik s listom *enrolledCoursesCodes* unutar koje su sadržani identifikacijske oznake određenih kolegija. Ti identifikatori korišteni su za dohvat podataka kolegija što je prikazano sljedećim programskim kodom.

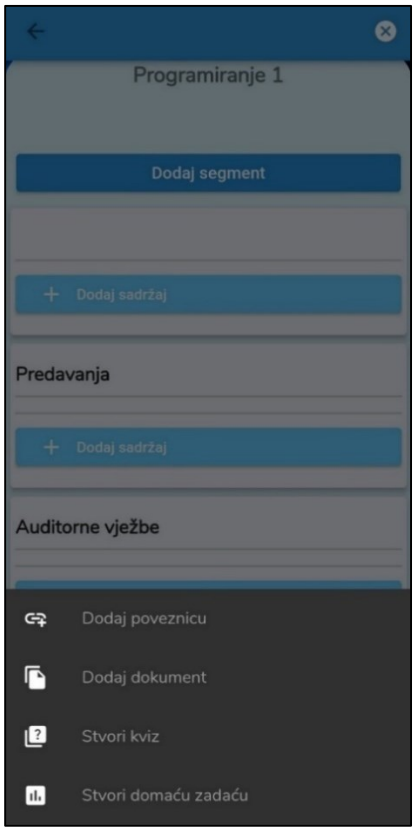
```
Future<Course> getCourseData(String code) async{
  CollectionReference collectionReference =
   FirebaseFirestore.instance.collection('courses');
  var documentSnapshot = await collectionReference.doc(code).get();
  Map<String, dynamic> data = documentSnapshot.data()! as Map<String, dynamic>;
  return Course.fromJson(data);
}
```

Programski kôd 3.7. Prikaz metode za dohvaćanje podataka kolegija prema identifikacijskoj oznaci

3.4. Postavljanje i uređivanje sadržaja kolegija

Kako bi se zadovoljili daljnji zahtjevi razvoja aplikacije potrebno je pružiti smislen i koristan sadržaj aplikaciji te omogućiti njegovo upravljanje. Gledano s pristupa aplikaciji predavač je onaj koji pruža sadržaj kolegija postavljanjem dokumenata, poveznica, stvaranjem kvizova, domaćih

zadaća i slično. U prethodnom poglavlju pojašnjen je raspored sadržaja, tj. svaki dodani sadržaj odgovara određenom segmentu određenog kolegija. Dodavanje sadržaja omogućeno je upotrebom skočnog dijaloga pritiskom na gumb kao što je prikazano na slici 3.2..



Slika 3.2. Snimka zaslona izbornika dodijele sadržaja segmentu kolegija.

Pohrana poveznica na određenu mrežnu stranicu (u bazi podataka) ostvarena je jednostavnom pohranom adrese kao zapisa niza znakova (engl. *String*) unutar liste. Dok kod pohrane dokumenta potrebno je spremiti sami dokument u pohranu podataka koju pruža Firebase usluga zvana Cloud Storage, no kako bi se kasnije pristupilo tom dokumentu potrebno je poznavati njegovu mrežnu adresu koja se sprema kao ujednačeni lokator sadržaja (engl. *Uniform Resource Locator* - URL). Pohrana datoteke ostvarena je prikazanom metodom *uploadFile* programskim kodom 3.8. čiji su parametri podatak tipa *File* i putanja do odredišta *destination* u zapisu niza znakova, dok je povratni tip *Future* tip podataka *UplaodTask?*. Programski kod 3.9. prikazuje primjenu navedene metode, gdje je varijabla *file* odabrana datoteka, a varijabla *filePath* putanja do odredišta gdje će podatak biti pohranjen. Nakon pohrane provjerava se je li dokument uspješno postavljen, odnosno je li vrijednost varijable *task* različita od nule. Ukoliko je uvjet ispunjen mrežna adresa dohvaćena je metodom *getDownloadURL*, koju pruža klasa *UploadTask*, što je omogućeno kroz proširenje *firebase_storage* [24].

```
Future<UploadTask?> uploadFile(File file, String destination) async{
  try{
    final ref = storage.ref(destination);
    return  ref.putFile(file);
  } on FirebaseException catch(e){
    print(e);
    return null;
  }
}
```

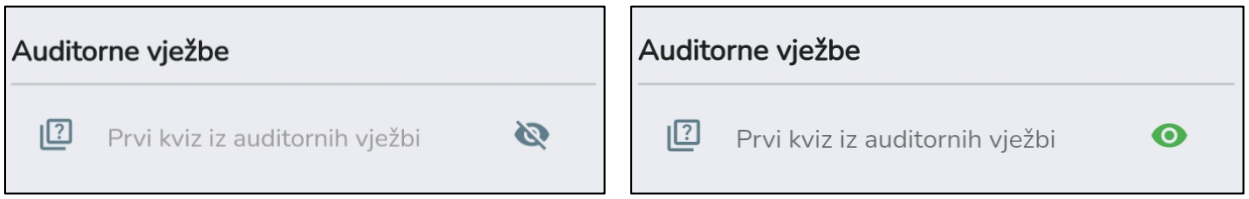
Programski kôd 3.8. Prikaz metode za postavljanje podatka u Firebase pohrani u oblaku

```
var task = await firebaseStorageService.uploadFile(file!, filePath);
String? url;
 if(task != null){
   url = await (await task)?.ref.getDownloadURL();
 }
```

Programski kôd 3.9. Prikaz upotrebe uplaodFile metode i dohvat URL-a datoteke

3.4.1. Provjera znanja kao sadržaj kolegija

Kao jedan od zahtjeva razvoja aplikacije bila je provjera znanja s povratnom informacijom s ciljem povećanja motivacije studenta. Provjera znanja je ostvarena u obliku kviza, pri čemu se kviz provjere znanja rješava s vremenskim ograničenjem i s određenim brojem pitanja s ponuđenim odgovorima koje postavlja predavač. Predavač stvaranjem novog kviza stvara novi dokument u kolekciji *quizzes* te unutar dokumenta unosi identifikacijske oznake kolegija i segmenta kojem pripada. Na taj način kviz pripada točno određenom segmentu odgovarajućeg kolegija, što je posebice važno kod ocjenjivanja i pretrage kviza. Dok su pitanja kviza postavljena u podkolekciji *questions* odgovarajućeg pitanja. Kako bi student mogao pristupiti kvizu kviz mora biti vidljiv što se jednostavno ostvaruje pritiskom na ikonu (Slika 3.3.) koja na to ukazuje, što je omogućeno sa strane predavača.



Slika 3.3. Snimke zaslona aplikacije s prikazom kviza određenog kolegija i njegove vidljivosti sa strane predavača

Pritiskom na ikonu svi studenti upisani na kolegij zaprimaju poruku obavijesti čime je ostvarena jedna od smjernica poticanja na sudjelovanje u nastavi oslanjajući se na dijalog podrške. Za ostvarenje navedenog također je korištena usluga Firebase, preciznije značajka Firebase Cloud Messaging. Tehnologija se temelji na token vrijednosti, svakom uređaju dodijeljena je posebna vrijednost te se ta vrijednost postavlja u dokument baze podataka odgovarajućeg korisnika. Vrijednost se postavlja pri stvaranju računa korisnika, ali također pri pokretanju aplikacije

(Programski kôd 3.10.). Time će korisnik zaprimiti obavijest na posljednjem uređaju kojeg je koristio.

```
String? token = await NotificationService().getNotificationToken();
```

Programski kôd 3.10. Prikaz dohvata token vrijednosti

Pri uklanjanju sadržaja provjere znanja odnosno kviza treba voziti brigu o potpunom brisanju. Drugim riječima, osim brisanja polja dokumenta uklanja se i podkolekcija ukoliko ona postoji inače dokument neće biti u potpunosti uklonjen, što je zapravo prikazano programskim kodom 3.11. brisanjem kviza.

```
Future<void> removeQuiz(String quizID) async{
  var ref = collectionReference.doc(quizID).collection('questions');
  var snapshot = await ref.get();
  var questions = snapshot.docs;
  for(var question in questions){
    question.reference.delete();
  }
  await collectionReference.doc(quizID).delete();
}
```

Programski kôd 3.11. Prikaz metode za uklanjanje kviza

```
Future<void> removeQuizModelFromCourseSegment(String courseCode, String
segmentCode, QuizSegmentModel quizModel) async {
 //Remove quiz with questions
  QuizService().removeQuiz(quizModel.quizID);

  //Remove model
  Map<String, dynamic> data = quizModel.toJson();
  var ref =
collectionReference.doc(courseCode).collection('segments').doc(segmentCode);
  ref.update({
    'quizSegmentModels': FieldValue.arrayRemove([data]),
  });

  //Remove quiz mark
  var studentIDs = await
GradeService().getAllStudentIDsWithSpecificActivityMark(courseCode,
 segmentCode, quizModel.quizID);
  if(studentIDs.isNotEmpty){
    for(var studentID in studentIDs){
      GradeService().deleteActivityMark(studentID, courseCode, segmentCode,
       quizModel.quizID);
    }
  }
}
```

Programski kôd 3.12. Prikaz metode za potpuno uklanjane kviza i svih poveznica na njega

Osim brisanja podkolekcije potrebno je obrisati i ocjenu za svakog studenta, jer taj kviz više ne postoji, ali također i ukloniti instancu poveznice segmenta i određenog kviza ostvarene klasom *QuizSegmentModel*. Što je ostvareno primjenom metode *removeQuizModelFromCourseSegment*

prikazane programskim kodom 3.12.. Metoda kao ulazne parametre prima jedinstvenu oznaku kolegija i segmenta te instancu modela kviza segmenta kojem pripada. Za uklanjane kviza primjenjuje se metoda *removeQuiz* prikazana programskim kodom 3.11., uklanjanje modela segmenta vrši se na način brisanja samo modela iz liste modela segmenta kviza spremljenih unutar kolekcije *segments* kao atribut odgovarajućeg dokumenta (segmenta kolegija). Za uklanjanje ostvarenog uspjeha na kvizu korištena je metoda *getAllStudentIDsWithSpecificActivityMark* koja kao ulazni parametar prima jedinstvenu oznaku kviza, a kao rezultat daje listu niza znakova jedinstvenih oznaka studenata kojim je dodijeljena ocjena iz navedene aktivnost. Slijednim prolaskom kroz jedinstvene oznake studenata za svakog studenata pozivamo metodu *deleteActivityMark* koja kao ulazne parametre prima jedinstvenu oznaku kviza, kolegija, segmenta i studenta čiju ocjenu uklanjamo.

3.4.2. Domaća zadaća kao sadržaj kolegija

Uzimajući u obzir tehnologiju samokontrole implementirana je domaća zadaća s jasnim prikazom roka predaje i njezinog zadatka. Cilj je jasnim prikazom roka predaje potaknuti studenta na pravovremenu predaju, a samim time na sjecanje radnih navika i pravovremenog izvršenja obveza. Slično kao kod provjere znanja prilikom prikaza domaće zadaće u vidljivom stanju (Slika 3.3.) svim upisanim studentima dolazi obavijest o navedenoj obvezi koja također biva zabilježena unutar pregleda obveza. Predavač je onaj koji stvara domaću zadaću i nakon stvaranja ima uvid u sve predane zadaće. Jednostavnim izbornikom prikazani su svi studenti koji su predali zadaću, odnosno svi studenti čije su zadaće predane na ocjenjivanje. Student u svakom trenutku ima uvid u stanje njegove zadaće, odnosno je li ocjenjena ili predana na ocjenjivanje oslanjajući se pri tome na tehnologiju nadzora. Model domaće zadaće prikazan je programskim kodom 3.13., koji kao atribute sadrži vlastiti jedinstveni identifikator, identifikatore kolegija i segmenta kojem pripada, te naziv, opis, popratni dokument i krajnji rok predaje domaće zadaće.

```
class Homework{
  String id; String courseCode;
  String segmentCode; String title;
  String description; String? documentURL; DateTime deadline;
  Homework({
    this.id = '', this.courseCode = '', this.segmentCode = '', this.title = '',
    this.description = '', this.documentURL, required this.deadline,
   }
  );
}
```

Programski kôd 3.13. Prikaz klase Homework koja predstavlja model domaće zadaće

Domaća zadaća postavlja se kao jedinstveni dokument kolekcije *homeworks* NoSQL baze podataka. Nakon podstavljanja jedinstveni identifikator domaće zadaće koristi se za povezivanje

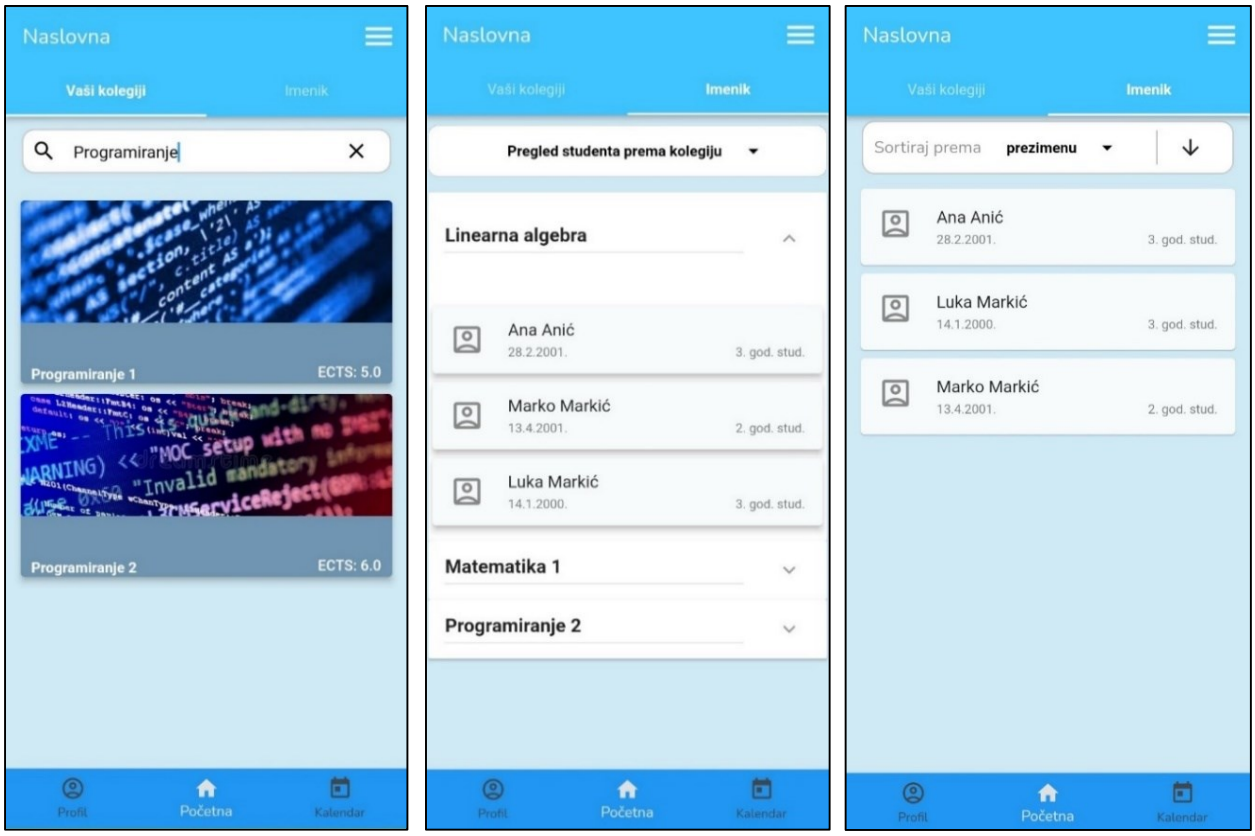
sa segmentom određenog kolegija, pretragu domaće zadaće kao i ocjenjivanje. Uklanjane domaće zadaće ostvareno je na sličan način kao i uklanjanje kviza (Programski kôdovi 3.11. i 3.12.). No također treba voditi brigu o brisanju odgovarajućeg podsjetnika i brisanju svih datoteka vezanih uz domaću zadaću pohranjenih u Firesbase Storage pohrani podataka. Brisanje datoteke iz Firebase Storagea ostvareno je jednostavnom metodom *deleteFileWithURL* prikazane programskim kodom 3.14. koja na temelju URL adrese datoteke uklanja datoteku iz pohrane.

```
Future<void> deleteFileWithURL(String URL) async{
  try{
      await FirebaseStorage.instance.refFromURL(URL).delete();
  }on FirebaseException catch(e){
    print(e);
    return null;
  }
}
```

Programski kôd 3.14. Prikaz metode za uklanjane datoteka iz Firebase Storagea na temelju URL-a

3.5. Implementacija pretraživanja, sortiranja i grupiranja sadržaja

S ciljem olakšane i efikasne pretrage sadržaja poput kolegija i studenata implementirana je pretraga sadržaja kao i sortiranje i grupiranje navedenog. Na sljedećoj slici prikazane su snimke zaslone aplikacije s navedenim funkcionalnostima.



Slika 3.4. Snimke zaslona aplikacije s prikazom pretraživanja, grupiranja i sortiranja sadržaja

Kada se radi s većim brojem podataka nužno je omogućiti neki oblik filtriranja sadržaja, što je istaknuto dijalogom podrške odnosno potrebno je ostvariti pristupačan, intuitivan i smislen dizajn aplikacije. Osim pretrage sa strane studenta isto je omogućeno sa strane predavača gdje je osim pretrage ponuđenih kolegija omogućena i pretraga svih studenta koji su upisani na kolegije koji su dodijeljeni predavaču. Funkcionalnost pretraživanja temelji se primjenom metode *contains* nad odgovarajućom listom elemenata te povratkom nove liste s filtriranim sadržajem.

```
void searchCourses(String query) {
  final courses = allCourses.where((course) {
    final titleLower = course.title.toLowerCase();
    final searchLower = query.toLowerCase();
    return titleLower.contains(searchLower;
  }).toList();

  setState(() {
    this.courses = courses;
  });
}
```

Programski kôd 3.15. Prikaz metode za pretraživanje kolegija

Programskim kodom 3.15. prikazana je metoda za pretraživanje prikazanih kolegija *searchCourses* s ulaznom vrijednosti upita pretraživanja u obliku niza znakova. Korištenjem metode *where* nad nizom svih ponuđenih kolegija i metode *contains* provjerava se je li traženi upit odgovara nekom od naziva kolegija. Zatim se novo dobivena lista kolegija postavlja kao trenutna lista kolegija koja se prikazuje na zaslonu koristeći uz to metodu *setState* kako bi se navedena promjena vidjela. Sortiranje sadržaja vrši se usporedbom vrijednosti atributa modela prikazanog sadržaja, primjerice prema prezimenu studenta (Programski kôd 3.16.).

```
students.sort((a, b) => a.surname.compareTo(b.surname))
```

Programski kôd 3.16. Prikaz sortiranja liste studenata prema prezimenu kao parametru sortiranja

Za ostvarivanje funkcionalnosti grupiranja korišten je model za grupiranje odgovarajućeg sadržaja, primjerice za grupirani prikaz kolegija korišten je model *CourseGroupedModel* prikazan programskim kodom 3.17..

```
class CourseGroupedModel{
  dynamic groupingByElement;
  List<Course> courses;
  bool isExtended;
  CourseGroupedModel({
    this.groupingByElement = '',
    this.courses = const [], this.isExtended = false,
  });
}
```

Programski kôd 3.17. Prikaz modela za grupirani prikaz kolegija

Klasa *CourseGroupedModel* kao atribute sadrži: listu kolegija koji sadrže atribut koji odgovara parametru grupiranja, parametar grupiranja i *boolean* vrijednost *isExtended* koja služi za evidentiranje proširenog prikaza pojedine grupe (Slika 3.4.) na zaslonu uređaja. Za dodavanje elementa u listu prema zadanom parametru grupiranja, odnosno u ovom primjeru kolegija u listu prema njegovom semestru koristi se metoda *groupCoursesBySemester* prikazana programskim kodom 3.18.. Metoda *groupCoursesBySemester* kao parametar prima listu svih kolegija te vraća listu grupa pripadajućeg semestara, vrši se dvostruki slijedni prolazak kroz kolegije s provjerom je li dani kolegij dodan u neku od grupa s ciljem izbjegavanja neželjenog i nepotrebnog multipliciranja grupa.

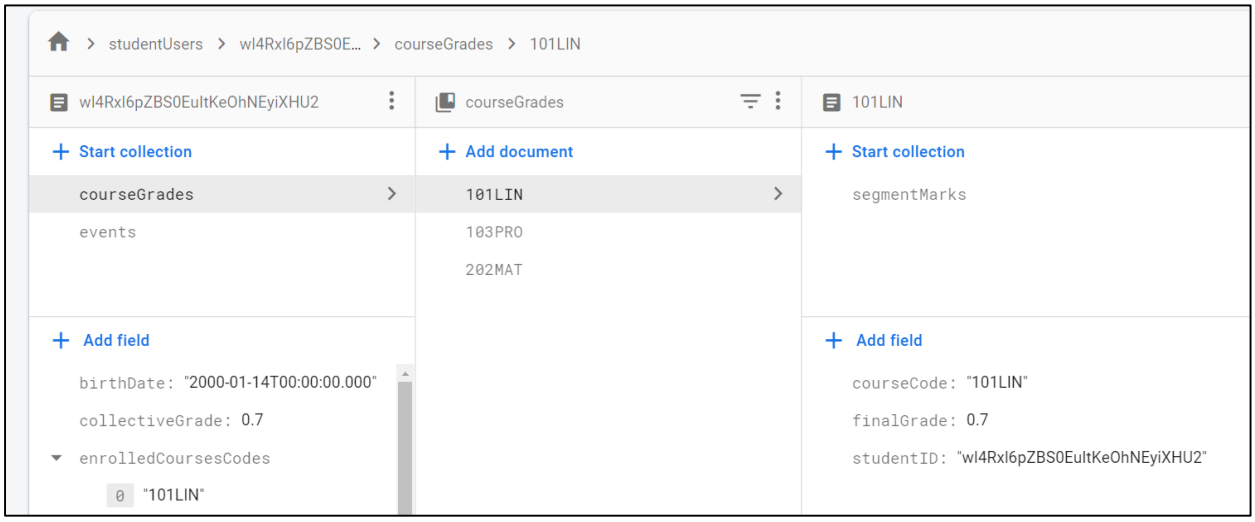
```
List<CourseGroupedModel> groupCoursesBySemester(List<Course> courses) {
  List<CourseGroupedModel> courseGroupedModels = [];
  List groupedByElements = [];
  for (var courseOne in courses) {
    var groupByElement = courseOne.semester;

    //Provjera je li vec sortirano po danom parametru
    if(!groupedByElements.contains(groupByElement)){
      groupedByElements.add(courseOne.semester);
      List<Course> modelCourses = [];
      for(var courseTwo in courses){
        if(courseTwo.semester == groupByElement)  modelCourses.add(courseTwo);
      }
      courseGroupedModels.add(CourseGroupedModel(groupingByElement:
       '$groupByElement. semestar', courses: modelCourses));
    }
  }
  return courseGroupedModels;
}
```

Programski kôd 3.18. Prikaz metode za grupiranje kolegija prema semestru

3.6. Pregled uspjeha i obveza

Također jedan od zahtjeva razvoja je pregled uspjeha, odnosno primjena tehnologije nadzora s utjecajem društvene usporedbe. Prvotno je potrebno razumjeti koncept ocjenjivana, uspjeh svakog segmenta poput predavanja, laboratorijskih vježbi i slično određen je prosjekom odrađenih aktivnosti kao što su provjere znanja i domaće zadaće. Uspjeh po pojedinom kolegiju predstavljen je aritmetičkom sredinom uspjeha svih segmenata. Te u konačnici prosječna uspješnost studenta predstavljana je aritmetičkom sredinom uspjeha po pojedinom kolegiju. Ocjene su dodijeljene kao podkolekcija određenog studenta, dok je njegova konačna ocjena postavljena kao polje tog dokumenta radi lakšeg pristupa podatku (Slika 3.5.). No kako bi uspjeh pojedinog studenta bio uspoređen s uspjehom drugih kolega upisanih na jednake kolegije kao taj student korištena je metoda *getAverageEnrolledStudentsGrade* prikazana programskim kodom 3.19.. Nakon što je dohvaćena lista studenta koji su upisane u navedene kolegije čije su identifikacijske oznake

primljene kao ulazni parametar izvršen je jednostavni prolazak kroz listu studenata. Prolaskom kroz listu provjerava se postojanje ocjene studenta te ukoliko je uvjet zadovoljen uvećana je cjelobrojna varijablu *length* za jedan i varijabli *sum* je pridijeljena vrijednosti uspjeha pojedinog studenta. Ukoliko je varijabla različita od nula, odnosno postoji barem jedan student koji ima odgovarajući uspjeh vratiti će se kvocijent vrijednosti varijabli *sum* i *lenght*, u suprotnom se ne vraća niti jedna vrijednost.



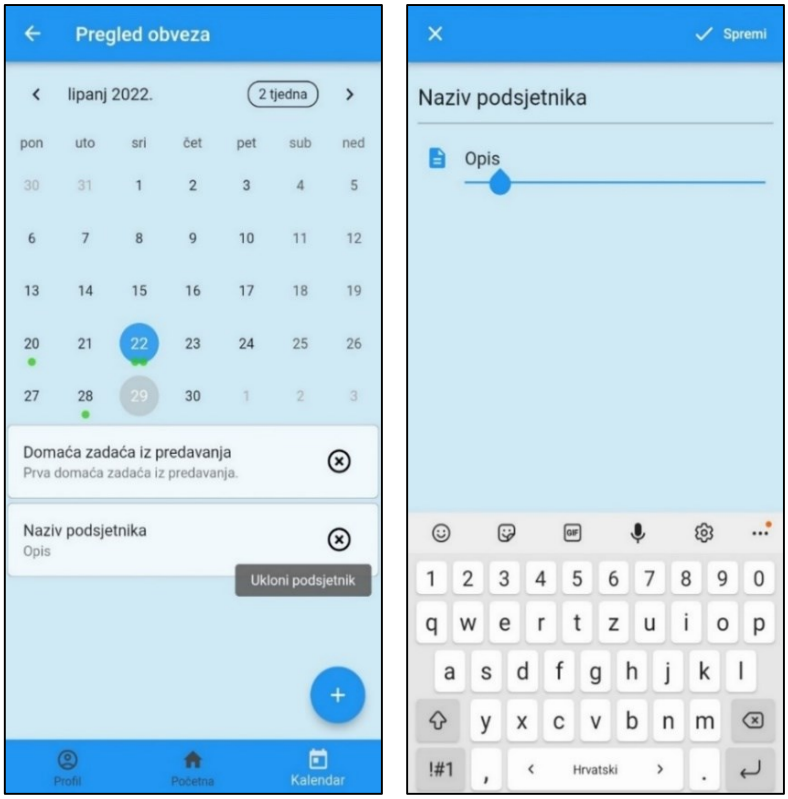
Slika 3.5. Prikaz pohrane uspjeha odgovarajućeg korisnika unutar Firebase baze podataka u oblaku

```
Future<double?> getAverageEnrolledStudentsGrade(List<String> courseCodes) async
{
  double sum = 0; int length = 0;
  List<ProfileStudent>? students = await
CourseService().getAllStudentEnrolledInCourses(courseCodes);
  if(students != null){
    for(var student in students){
      if(student.collectiveGrade != null){
        sum += student.collectiveGrade!;
        length++;
      }
    }
    if(length != 0) return sum / length;
  }
}
```

Programski kôd 3.19. Prikaz metode izračuna prosječnog uspjeha svih studenta koji su upisani na odgovarajuće kolegije

U konačnici vodeći se još jednom smjernicom izrade aplikacije implementiran je pregled obveza korištenjem kalendara, što je ostvareno korištenjem proširenja *table_calendar* [25]. Podaci o događajima su pohranjeni kao podkolekcija dokumenta baze podataka koji predstavlja podatke korisnika, budući kako svaki korisnik ima vlastite obveze. Obveze se postavljaju u obliku podsjetnika (događaja) kalendara, koje može postaviti predavač ili sam student postavljanjem vlastitog podsjetnika. Primjerice, predavač može postaviti podsjetnik krajnjeg roka predaje

domeće zadaće, a student postavlja određeni prilagođeni podsjetnik. Navedeno je prikazano slikom 3.6., gdje je prikazana snimka zaslona aplikacije s prikazom kalendara za prikaz obveza i snimka zaslona postavljanja podsjetnika.



Slika 3.6. Prikaz snimke zaslona aplikacije s prikazom kalendara za prikaz obveza i obrasca stvaranja podsjetnika

4. ZAKLJUČAK

Završnim radom ukratko je predstavljeno mobilno učenje, kao i njegov utjecaj na akademski uspjeh. Određene su smjernice dizajniranja mobilne aplikacije za poticanje aktivnog sudjelovanja studenata u nastavi. Kroz tehnologije uvjeravanja poput kroja, samokontrole, nadzora, prijedloga, oslanjajući se uz to na dijalog podrške i psihologiju društvenog utjecaja, stvorene su glavne smjernice razvoja mobilne aplikacije. Za realizaciju navedene aplikacije korišten je razvojni softver Flutter koji pak koristi objektno orijentirani programski jezik Dart dok je pohrana podataka i sadržaja u oblaku i slanje obavijesti ostvareno korištenjem usluge Firebase s kratkim objašnjenjem navedenih tehnologija. Prilagodba sadržaja korisniku se nametnula kao jedan od zahtjeva aplikacije što je ostvareno prilagodbom računu korisnika. Student ima mogućnost odabira kolegija, a na njima se nalazi odgovarajući sadržaj podijeljen prema segmentima (laboratorijske vježbe, predavanja i drugi). Podjela kolegija na segmente je posebice važna kod ocjenjivanja, a uspjeh segmenta određen je uspjehom riješenosti pripadajućih kvizova i domaćih zadaća, dok će uspjeh segmenata odrediti uspješnost riješenosti kolegija. Objavom provjera znanja, odnosno domaćih zadaća, student je obaviješten korištenjem obavijesti vodeći se dijalogom podrške. Nadalje, statističkim pregledom uspjeha student je u mogućnosti vidjeti uspješnost riješenosti pojedinog kolegija kao i uvid u prosjek uspješnosti drugih studenata koji su upisani u iste kolegije. Za olakšanu pretragu sadržaja poput kolegija i studenata implementirane su funkcionalnosti pretrage, sortiranja i grupiranja istih. U konačnici prikazom obveza putem kalendara omogućeno je dodavanje vlastitih podsjetnika, ali također prikaz drugih obveza poput krajnjeg roka predaje domaće zadaće.

LITERATURA

[1] A. Peko, V. Mlinarević i V. Buljubašić-Kuzmanović, Potreba unaprjeđivanja sveučilišne nastave, *Odgojne znanosti,* svez. 10, br. 1, str. 195-208, lipanj 2008.

[2] Y.L. Jeng, T.T. Wu, Y.M. Huang, Q. Tan i S.J.H. Yang, The Add-On Impact of Mobile Applications in Learning Strategies: A Review Study, *Educational Technology & Society,* svez. 13, br. 3, str. 3-11, travanj 2010.

[3] L. C. Jain, S.L. Peng, B. Alhadidi i S. Pal, Intelligent Computing Paradigm and Cutting-edge Technologies, Springer Nature Switzerland AG, 2020.

[4] M. Sharples, J. Taylor, i G. Vavoula, A Theory of Learning for the Mobile Age, u *The SAGE handbook of e-learning research, 2nd edition*, Sage, 2016, str. 63-81.

[5] J. Rossing, iLearning: The future of higher education? Student perceptions on learning with mobile tablets, *International Journal for the Scholarship of Teaching and Learning,* svez. 12, br. 2, str. 1-26, lipanj 2016.

[6] „Test sume rangova," [Mrežno]. Dostupno na: hr.wikipedia.org/wiki/Test_sume_rangova. [lipanj 2022].

[7] K. Demir i E. Akpinar, The effect of mobile learning applications on students' academic achievement and attitudes toward mobile learning, *Malaysian Online Journal of Educational Technology,* svez. 6, br. 2, str. 48-59, 2018.

[8] T. Fritz, E.M. Huang, G. Murphy, T. Zimmermann, Persuasive Technology in the Real World: A Study of Long-Term Use of Activity Sensing Devices for Fitness, u *CHI 2014*, 2014.

[9] B.J. Fogg, Persuasive Technology: Using Computers to Change What We Think and Do, Stanford, Kalifornija: Morgan Kaufmann Publishers, 2003.

[10] E. Eilu, R. Baguma, J.S. Pettersson, G. D. Bhutkar, Digital Literacy and Socio-Cultural Acceptance of ICT in Developing Countries, Springer Nature Switzerland AG, 2021.

[11] „Meet Android Studio," [Mrežno]. Dostupno na: https://developer.android.com/.... [lipanj 2022].

[12] „IntelliJ IDEA," [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/IntelliJ_IDEA. [lipanj 2022].

[13] „Set up an editor,“ [Mrežno]. Dostupno na: https://docs.flutter.dev/get-started/editor. [lipanj 2022].

[14] T. Bailey, A. Biessek, Flutter for Beginners: An introductory guide to building cross-platform, Packt Publishing, 2021.

[15] M. L. Napoli, Beginning Flutter: A Hands On Guide to App Development, Wrox Press, 2019.

[16] T. Goel, „Lifecycle of Flutter App,“ [Mrežno]. Dostupno na: https://mobikul.com/lifecycle-of-a-flutter-app/. [lipanj 2022].

[17] „Widget catalog,“ [Mrežno]. Dostupno na: https://docs.flutter.dev/development/ui/widgets. [lipanj 2022].

[18] A. Kumar, Mastering Firebase for Android Development: Build real-time, scalable, and cloud-enabled Android apps with Firebase, Packt Publishing, 2018.

[19] „Firebase,“ [Mrežno]. Dostupno na: https://docs.flutter.dev/development/data-and-backend/firebase. [lipanj 2022].

[20] „Cloud Firestore,“ [Mrežno]. Dostupno na: https://firebase.google.com/docs/firestore. [lipanj 2022].

[21] „Usage and limits,“ [Mrežno]. Dostupno na: firebase.google.com/docs/firestore/quotas. [lipanj 2022].

[22] H. Yahiaoui, Firebase Cookbook: Over 70 recipes to help you create real-time web and mobile applications with Firebase, Packt Publishing Ltd, 2017.

[23] „json_serializable,“ Google LLC, [Mrežno]. Dostupno na: https://pub.dev/packages/json_serializable . [lipanj 2022].

[24] Firebase, „firebase_storage,“ [Mrežno]. Dostupno na: packages/firebase_storage. [lipanj 2022].

[25] „table_calendar,“ [Mrežno]. Dostupno na: https://pub.dev/packages/table_calendar . [lipanj 2022].

SAŽETAK

Ovim završnim radom izrađena je mobilna aplikacija za poticanje aktivnog sudjelovanja studenata u nastavi i utvrđeni su osnovni zahtjevi njezinog razvoja. Oslanjajući se na tehnologije uvjeravanje i psihologiju društvenog utjecaja određeni su zahtjevi razvoja na osnovu kojih je je mobilna aplikacija razvijena i realizirana korištenjem razvojnog softvera Flutter i Firebase usluge za pohranu podataka u oblaku. Kroz mobilnu aplikaciju pruža se pregled kolegija i njihovog sadržaja, kao i njivo pretraživanje, sortiranje i grupiranje. Svakom kolegiju po danom segmentu dan je odgovarajući sadržaj u obliku dokumenta, provjere znanja, domaće zadaće i slično. Student sudjelovanjem, odnosno rješavanjem aktivnosti poput provjera znanja i domaćih zadaća ostvaruje određeni uspjeh na kolegiju, pri čemu je omogućen uvid u vlastiti uspjeh i statistički pregled istih. Pružajući uz to studentima pregled vlastitih obveza u svakom trenutku kao i funkcionalnost dodavanje vlastitih podsjetnika.

Ključne riječi: Dart, Društveni utjecaj, Firebase, Flutter, Mobilno učenje, Tehnologije uvjeravanja

ABSTRACT

This bachelor's thesis covers the development of the mobile application for encouraging students in active participation in curriculum activities and establishment of basic requirements of its development. Based on Persuasive Technologies and psychology of Social Impact, basic requirements for the app development are established on which is mobile application developed and realized with use of developing software Flutter and service for storing data in cloud Firebase. The mobile application provides an overview of courses and their content, as well as course search, sort and grouping functionality. To each course appropriate content is given for a given segment in form of document, tests, homework and likewise. By participating, or solving activities such as tests and homework, the student achieves a certain course grade, while providing insight into his/her own grades and general statistical overview. Additionally providing students at any time with overview of their obligations as well as functionality of adding personalized reminders.

Key words: Dart, Firebase, Flutter, Mobile learning, Persuasive Technologies, Social Impact

ŽIVOTOPIS

Luka Markić rođen je 14. siječnja 2000. godine u Mostaru, 2014. godine završava Treću osnovnu školu Mostar nakon čaga upisuje Srednju elektrotehničku školu Ruđera Boškovića u Mostar. Završetkom srednjoškolskog obrazovanja upisuje preddiplomski studij računarstva Fakulteta strojarstva, računarstva i elektrotehnike Sveučilišta u Mostaru, nakon čega odobrenim prelaskom se upisuje kao student preddiplomskog sveučilišnog studija računarstva Fakulteta elektrotehnike, računarstva i informacijski tehnologija Osijek. Član je neprofitne studentske udruge HUMRSPTZ (engl. *IAESTE Croatia*) u kojoj je obnašao funkciju Fundraising koordinatora lokalnog odbora Osijek dvije godine te je sudjelovao u izradi i provedbi raznih studentskih projekta organiziranih u udruzi.

Luka Markić

PRILOZI

Izvorni kod aplikacije

Dostupan putem poveznice: https://github.com/LukaMarkic/students_lab.git.