Vizualni višejezični prevoditelj zasnovan na načelima računalnih igara

Radoš, Marin

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:310946

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-04**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

Vizualni vižejezični prevoditelj zasnovan na načelima računalnih igara

Diplomski rad

Marin Radoš

Osijek, 2017.

Sadržaj

1. UVOD 1
2. KORIŠTENJE NAČELA RAČUNALNIH IGARA U UČENJU 2
2.1. Uvod u problematiku učenja uz pomoć igrifikacije 2
2.2. Igrifikacija 2
2.3. Vizualni višejezični prevoditelj 3
2.3.1. Prepoznavanje u vizualnom višejezičnom prevoditelju 3
2.3.2. Prevođenje u vizualnom višejezičnom prevoditelju 3
2.3.3. Učenje na načelu rada računalnih igara 3
2.4. Ideja rada 4
2.4.1. Pregled postojećih sličnih rješenja 4
2.4.2. Unaprjeđenja koja donosi rad 4
2.5. Prikaz koncepta idejnog rješenja 4
3. PRIJEDLOG PROGRAMSKE ARHITEKTURE I KORIŠTENI PROGRAMSKI ALATI 6
3.1. Arhitektura programskog rješenja 6
3.1.1. MVVM arhitektura 6
3.1.2. Uzorak koordinatora 7
3.2. Korištene tehnologije i alati 9
3.2.1. Komplet za razvoj programskih rješenja za iOS 9
3.2.2. Razvojno okruženje Xcode 10
3.2.3. Uređaji i operacijski sustavi 10
3.3. Korišteni programski jezici i okviri 10
3.3.1. Programski jezik Swift 10
3.3.2. Mobilna platforma Realm 11
3.4. Korišteni API-ji 11
3.4.1. IBM Watson vizualno prepoznavanje 11
3.4.2. IBM Watson prevođenje jezika 11
4. PROGRAMSKO RJEŠENJE 13
4.1. Odabir jezika i prepoznavanje objekata na fotografiji 13
4.2. Spremanje u lokalnu bazu mobilnog uređaja 16
4.3. Korisničko sučelje za učenje kroz rješavanje kvizova 17
5. KORIŠTENJE I TESTIRANJE APLIKACIJE 20
5.1. Korištenje kamere za fotografiranje i galerije za odabir slika 20
5.2. Izbor jezika 22
5.3. Odabir ispravnog prepoznavanja i spremanje u bazu podataka 22
5.4. Odabir i rješavanje kvizova 23

5.5. Analiza točnosti vizualnog prepoznavanja 26
5.6. Testiranje aplikacije automatskim testovima 27
5.7. Testiranje aplikacije korištenjem aplikacije 28
5.8. Rukovanje greškama 28
5.9. Korisničko iskustvo kroz korištenje aplikacija 31
6. ZAKLJUČAK 32
LITERATURA .....
SAŽETAK .....
ABSTRACT .....
ŽIVOTOPIS .....
PRILOZI .....

1. UVOD

U ovom radu prikazuje se problem učenja kroz igrifikaciju. Kao rješenje predložena je aplikacija koja koristi vizualno prepoznavanje i prevođenje kako bi korisnik stvorio vlastite kvizove za provjeru znanja. Namjena aplikacije je, uz pomoć učenja na načelu rada računalnih igara, omogućiti adaptivnu igrifikaciju. Aplikacija je namijenjena mlađem uzrastu i učenicima, no jednako je dostupna svima. Svaki korisnik aplikacije može sam stvarati svoj sadržaj kroz koji uči jezik. Kako bi se rješenje ostvarilo, bit će korišteni alati za vizualno prepoznavanje i prevođenje implementirani u aplikaciju za mobilni uređaj iPhone. Aplikacija će biti razvijna u iOS okruženju i programskom jeziku Swift.

U drugom poglavlju opisan je problem učenja na načelu rada računalnih igara te je predloženo rješenje. Predloženo rješenje je uspoređeno s ranijim sličnim rješenjima te je iznesen kocept i ideja kako ostvariti programsko rješenje. U trećem poglavlju navedeni su alati i tehnologije koje će omogućiti razvitak aplikacije kao što su programska razvojna okruženja, programski jezik te programski okviri. U četvrtom poglavlju opisano je programsko rješenje te su detaljno prikazani važni dijelovi programskog koda. Objašnjene su metode kojima se došlo do rješenja. U petom poglavlju prikazana je gotova aplikacija. Opisano je korištenje aplikacije iz perspektive korisnika. Također su opisane metode testiranja programskog rješenja, te rukovanje greškama koje su pronađene za vrijeme testiranja.

2. KORIŠTENJE NAČELA RAČUNALNIH IGARA U UČENJU

U ovom poglavlju bit će opisana problematika učenja na načelu računalnih igara te ideja i prijedlog rješenja za lakše učenje jezika kroz igrifikaciju.

2.1. Uvod u problematiku učenja uz pomoć igrifikacije

Postoje načini na koji se djecu potiče na učenje jezika od rane dobi, jedan od takvih primjera je edukacijska animirana serija Dora Istraživačica (eng. *Dora the Explorer*) u čijoj izvornoj inačici, prema [3] djeca na engleskom jeziku uče španjolski kroz Dorinu interakciju sa drugim likovima i samim gledateljem. U današnje vrijeme većina djece ima pristup nekakvom uređaju koji je povezan s internetom te imaju pristup edukacijskom sadržaju i igrama. Prema [1], u 2014. godini 69% ispitane djece koristilo je mobilni uređaj, od toga dvije trećine imaju pametni telefon. 66.2% ispitanika koji su pristupali internetu preko uređaja koristila su uređaj u određenoj mjeri za samostalno igranje igara, a 77% je koristilo pristup internetu za učenje. Do dvanaeste godine 80% djece koja pristupa internetu skinulo je nekakvu aplikaciju.

Na listi najboljih aplikacija za učenje jezika, prema [2], nalaze se tek dvije aplikacije koje uključuju elemente igrifikacije ili učenje namijenjeno djeci. Jedna od tih aplikacija je, trenutno najpopularnija aplikacija za učenje jezika, Duolingo, koja angažira korisnika davanjem bodova i napretkom kroz jezike, no aplikacija nije dovoljno prilagođena kako bi bila zanimljiva djeci. Druga aplikacija s liste je Mindsnacks koja koristi zanimljiv dizajn prilagođen djeci kako bi olakšala učenje jezika kroz igru. Nijedna od tih aplikacija ne angažira djecu da sama otkrivaju svijet oko sebe i prilagode sama sebi učenje kroz znatiželju.

2.2. Igrifikacija

Prema [4], igrifikacija u edukaciji je proces korištenja razmišljanja i mehanike igara s ciljem poticanja uključivanja učenika i rješavanje problema. Igrifikacija koristi tehnike igara da bi aktivnosti učenja učinila privlačnijima i zabavnijima. Većina djece u današnje vrijeme ima pristup računalu, tabletu ili mobilnom uređaju što igre čini lako dostupnima. Današnje generacije su generacije igrača video igara te su od malena okružene napretkom i razvojem informatičkih tehnologija. Potrebno je integriranje informacijsko komunikacijskih tehnologija u učenje jer tehnologija čini prirodno okruženje učenika i mladih u današnjim generacijama. Igrifikacija je prilika za povećanje interesa, motivacije, povezivanja te osnaživanja komunikacije djece i mladih.

Igrifikacija cilja na ohrabrivanje učenika, mijenjanje navika i unaprjeđenje rješavanja problema. Prema [5], mobilno i sveprisutno učenje potiče učenje izvan formalnih konteksta kao što su učionice. Iako je dokazano da inteligentni sustavi podučavanja poboljšavaju uspjeh učenika i učenje, postoje problemi u igrifikaciji na kojima je potreban daljni rad. Jedan od problema je neprikladno korištenje takvih sustava od strane učenika zbog dosade, monotonije ili nedostatka motivacije. Učenje i testovi bi se trebali neprestano mijenjati kako bi zadržali pozornost učenika te motivirali daljnje unaprjeđivanje znanja. Prema [6], uspješna implementacija igrifikacije je ona koja dovodi rezultate učenja do najviše moguće točke. Kako bi se to ostvarilo potrebno je prilagoditi igrifikaciju svakom pojedincu. Najbolji oblik igrifikacije bio bi onaj gdje je učenje u potpunosti prilagođeno učeniku.

2.3. Vizualni višejezični prevoditelj

Vizualni višejezični prevoditelj je aplikacija za mobilne uređaje iPhone i operacijski sustav iOS koja omogućuje učenje jezika kroz igru korištenjem metode igrifikacije uz prilagodbu učenja svakom individulanom korisniku. U nastavku slijedi opis pojedinih važnih dijelova aplikacije.

2.3.1. Prepoznavanje u vizualnom višejezičnom prevoditelju

Korisnik aplikacije može fotografirati bilo koji objekt ili scenu te se uz pomoć algoritma za vizualno prepoznavanje sadržaja fotografije korisniku predloži opis uslikanog sadržaja na engleskom jeziku. Ukoliko je korisnik zadovoljan tim opisom, može ga zadržati ili ponoviti fotografiranje. Implementacija prepoznavanja u aplikaciju opisana je u potpoglavlju 4.1.

2.3.2. Prevođenje u vizualnom višejezičnom prevoditelju

Vizualni višejezični prevoditelj prevodi opise sadržaja fotografije koje je prepoznao algoritam za prepoznavanje na jezik odabran od strane korisnika. Jezici na koje je moguće prevesti sadržaj su španjolski, njemački, francuski, talijanski i portugalski. Implementacija prevođenja u aplikaciju opisana je u potpoglavlju 4.1.

2.3.3. Učenje na načelu rada računalnih igara

Fotografija, engleski opis i prijevod na odabrani jezik se spremaju u lokalnu bazu uređaja te se od njih kreiraju kvizovi kroz koje se uči odabrani jezik. S obzirom na svoje znanje, korisnik može odabrati različite težine kvizova koje predstavljaju veći izazov. Težine kvizova se reguliraju složenostima riječi i prikazom fotografija.

2.4. Ideja rada

Za razliku od dosadašnjih rješenja, aplikacija angažira korisnika da se kreće i samome sebi prilagodi igru kroz korištenje kamere i kvizova različite težine. Korisnik može fotografirati bilo koji predmet ili scenu za koju želi saznati prijevod na odabrani jezik koji uči. Aplikacija će prepoznati o kojem objektu se radi te ponuditi korisniku prijevod i opciju spremanja podataka. Na osnovu tih podataka stvaraju se kvizovi s bodovanjem. Korisnik, s obzirom na svoje trenutno znanje jezika, može odabrati jednu od različitih težina kvizova. Aplikacija je primarno namijenjena mlađim uzrastima, no može se koristiti u bilo kojoj životnoj dobi.

2.4.1. Pregled postojećih sličnih rješenja

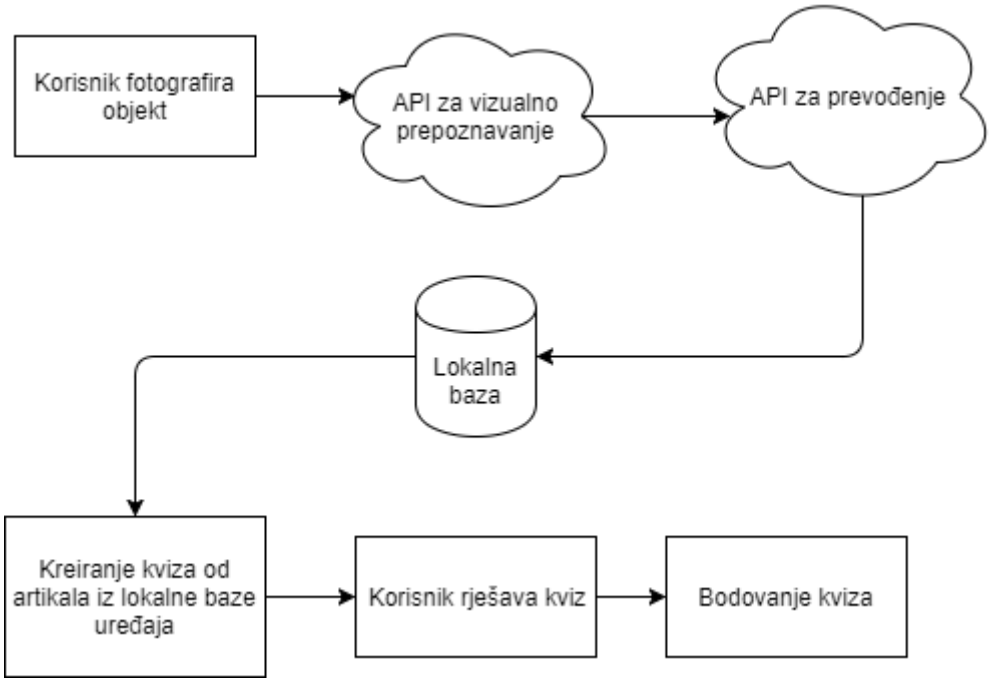
Prema [2], postoje aplikacije za učenje jezika kroz igrifikaciju koje su namijenjene mlađem uzrastu kao što su Mindsacks i Duolingo. Navedene aplikacije su dizajnirane kako bi njihovo korisničko sučelje bilo zanimljivo mlađim uzrastima te koriste elemente igrifikacije za motiviranje učenika. Jednako tako postoje i aplikacije koje koriste vizualno prepoznavanje za prepoznavanje sadržaja fotografija. Jedan od primjera je aplikacija CamFind koja, prema [7], ima preko milijun preuzimanja na Google trgovini. Osim nje, postoji pregršt aplikacija koje koriste neki od poznatih usluga kao što su IBM Watson Visual Recognition API, Google Vision API ili Microsoft Computer Vision API.

2.4.2. Unaprjeđenja koja donosi rad

Nijedna od ranije navedenih aplikacija nije spojila vizualno prepoznavanje i učenje na načelu rada računalnih igara. Aplikacije za učenje jezika obično imaju predefinirane kvizove ili pitanja, a vizualno prepoznavanje obično nije povezano s učenjem za mlađe uzraste. Vizualni višejezični prevoditelj omogućuje prilagodbu igrifikacije svakom individualnom korisniku. Korisnik uz pomoć vizualnog prepoznavanja sam stvara svoje kvizove koje zatim pokušava riješiti. Osim toga, s obzirom na svoje znanje, može odabrati između više težina kvizova što omogućuje postupno unaprjeđenje znanja.

2.5. Prikaz koncepta idejnog rješenja

Prikaz koncepta idejnog rješenja koristeći vizualno prepoznavanje, prevođenje i kreiranje kvizova iz lokalne baze na osnovi fotografiranog sadržaja prikazan je na slici 2.1.



Sl. 3.1. *Koncept idejnog rješenja*

Nakon što korisnik fotografira objekt koji želi dodati u svoje kvizove, slika se pošalje na uslugu za vizualno prepoznavanje koja prikaže skup najvjerojatnijih rješenja. Korisnik izabire rješenje koje smatra točnim te se taj izraz šalje usluzi za prevođenje na odabrani jezik. Nakon što je korisnik zadovoljan s prepoznavanjem i prevođenjem objekta, sprema fotografiju, originalni izraz i prevedeni izraz u lokalnu bazu uređaja. Kada se dovoljno objekata nalazi u bazi, korisnik može stvoriti kviz. Kviz se nasumično generira iz lokalne baze te korisnik riješi kviz. Nakon završetka kviza boduje se korisnikovo postignuće.

3. PRIJEDLOG PROGRAMSKE ARHITEKTURE I KORIŠTENI PROGRAMSKI ALATI

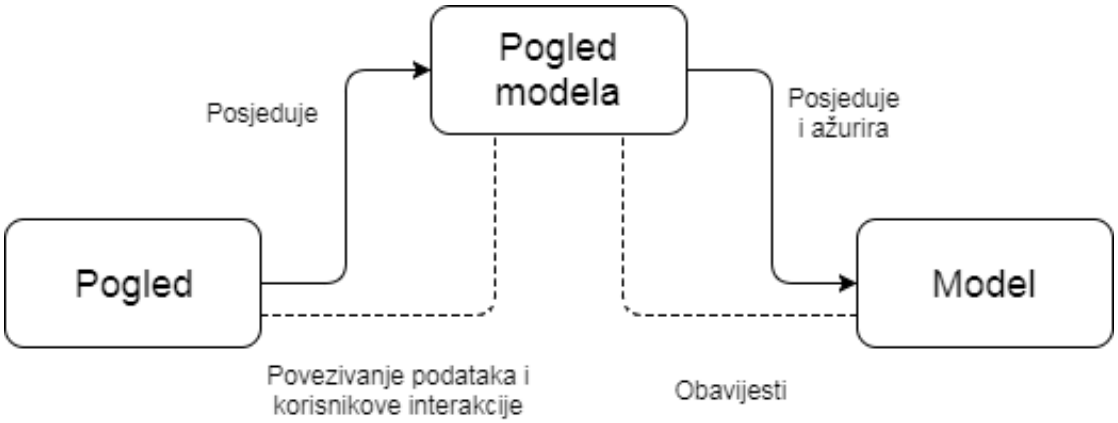
U ovom poglavlju opisana je predložena programska arhitektura MVVM-C te korišteni programski alati kao što su programski jezik, programski okviri i razvojna okruženja.

3.1. Arhitektura programskog rješenja

Za arhitekturu programskog rješenja korištena je MVVM-C ili Model-View-ViewModel-Coordinator programska arhitektura [29]. Ona se sastoji od Model-View-ViewModel arhitekture i uzorka koordinatora koji služi za navigaciju između kontrolera pogleda, to jest različitih zaslona aplikacije.

3.1.1. MVVM arhitektura

Prema [8], MVVM ili Model-View-ViewModel programska arhitktura sastoji se od modela, pogleda (eng. *view*) i pogleda modela (eng. *view model*). U iOS okruženju pogled predstavlja kontroler pogleda (eng. *view controller*) u koji se piše sav kod koji je vezan uz korisničko sučelje, na primjer gumbovi, oznake, ikone i slično. Model predstavlja modele baze, dok model pogleda sadrži svu logiku koja povezuje događaje na pogledu i promjene na modelu. Pogled i model nemaju čvrstu vezu između sebe što je vidljivo na slici 3.1.



Sl. 3.1. *MVVM arhitektura*

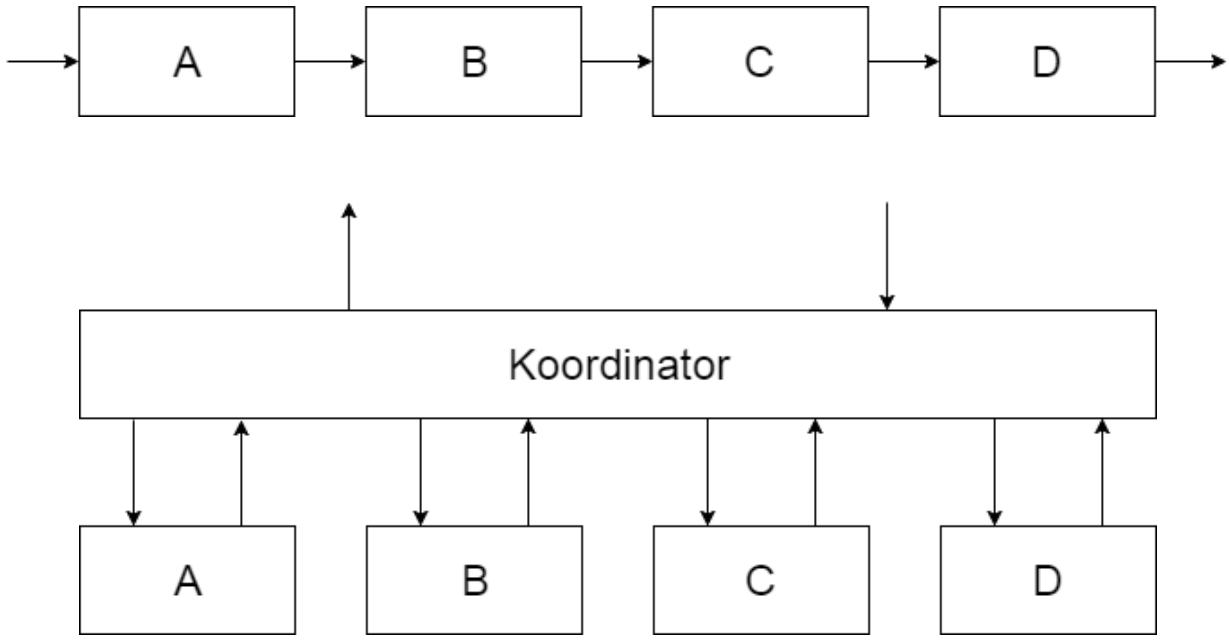
Model pogleda ne zna ništa o pogledu što programeru omogućuje lakše automatsko testiranje logike koja se nalazi u modelu pogleda. Pogled se također može testirati, no s obzirom da je ovisan o UIKit modulu bolje ga je testirati manualnom metodom.

Prema [8], prednosti MVVM-a naspram MVC arhitekture je to što je MVVM puno jasnije definiran, to jest jasnije je koji dio arhitekture vrši koju funkciju. Međutim, MVVM obično ima

nešto više programskog koda. U usporedbi s MVP arhitekturom, MVVM ima manje programskog koda, no oboje imaju jasno definirane zadaće svakog dijela arhitekture. Iako se MVVM arihtektura ne ističe kao predvodnik u bilo kojem aspektu arhitekture, jasno pokriva nedostatke koje imaju MVC i MVP arhitekture.

3.1.2. Uzorak koordinatora

Prema [9], koordinator je objekt koji upravlja navigacijom između različitih pogleda i zaslona aplikacije. Postoji jedan koordinator više razine koji upravlja cijelom aplikacijom. On stvara druge koordinatore kao svoju djecu. Svaki koordinator zadužen je za jedan dio aplikacije, što znači da se koordinatori trebaju rasporediti po logično povezanim cjelinama. Uzorak koodrinatora izolira svaki kontroler pogleda kao zasebnu jedinicu. Na ovaj način, svaki kontroler pogleda zna samo za sebe i niti za ijedan drugi kontroler pogleda. Ovim pristupom smanjuje se ovisnost kontrolera pogleda kao što je prikazano na slici 3.2.



Sl. 3.2. *Zavisnost kontrolera pogleda bez koordinatora (iznad) i sa koordinatorom (ispod)*

Koordinatori također čine kontroler pogleda pogodnima za višekratnu upotrebu. Jedan kontroler pogleda može se stvoriti i prikazati iz bilo kojeg drugog dijela aplikacije. Jedna klasa kontrolera prikazana je u prikazu programskog koda 3.1.

```
final class InfoCoordinator: NavCoordinator, Coordinator {

    private var navigationController = BaseNavigationController()

    func start() -> UIViewController {
        let vc = InfoViewController.instance()

        navigationController.viewControllers = [vc]
        vc.navigationBarDisplayMode = .always
        return navigationController
    }
}
```

Programski kod 3.1. Koordinator

Klasa InfoCoordinator, prikazana prikazom koda 3.1., naslijeđuje protokol Coordinator koji ima na sebi samo jednu metodu, *start*, koja pokreće prvi zaslon tog koordinatora. Prema [14], protokol definira nacrt metoda, svojstava i drugih zahtjeva koje klasa koja nasljeđuje protokol mora ispuniti. Za svaku klasu ili tip koji zadovoljava te zahtjeve kaže se kako se on prilagođava (eng. *conform*) protokolu kojeg nasljeđuje. Metoda ima povratni tip *UIViewController* koji je sastavni dio UIKit okvira. Prema [11], UIKit je programski okvir razvijen od strane Apple-a kako bi programeri mogli izgraditi i upravljati korisničkim sučeljem koje je upravljano događajima. Prema [12], *UIViewController* je klasa koja pruža infrastrukturu za upravljanje pogleda aplikacije. Svaki pogled iOS aplikacije je tipa *UIViewController* ili neke njegove podklase. Metoda *start* vraća kontroler pogleda zaslona kojeg treba pokazati na paljenju aplikacije što se vidi iz njezinog povratnog tipa. Progamski kod prikazan je u prikazu programskog koda 3.2.

```
public protocol Coordinator: class {

    @discardableResult
    func start() -> UIViewController
}
```

Programski kod 3.2. Coordinator protokol

Kontroler pogleda InfoViewController zove metodu *instance* koja poziva metodu *createFromStoryboard*, koja omogućuje stvaranje kontrolera pogleda iz storyboarda bez seguea. Metoda *instance* ima povratni tip *Self*, što, prema [13], znači da vraća opći (eng. *generic*) tip iz metode *createFromStoryboard*, a ona vraća opći tip *UIViewController* podklase koja dijeli ime sa storyboard datotekom. Metoda *instance* prikazana je u prikazu programskog koda 3.3.

```
open class func instance() -> Self {
        if let vc = createFromStoryboard(type: self) {
            return vc
        } else {
            print("WARNING: can't create view controller from storybard:\(self)")
            return self.init()
        }
    }

    private class func createFromStoryboard<T: UIViewController>(type: T.Type) -> T? {

        let storyboardName = String(describing: type)

        let bundle = Bundle(for: T.self)

        guard bundle.path(forResource: storyboardName, ofType: "storyboardc") != nil else {
            return nil
        }

        let storyboard = UIStoryboard(name: storyboardName, bundle: bundle)

        guard let vc = storyboard.instantiateInitialViewController() else {
            print("no vc in storyboard(hint: check initial vc)") ; return nil
        }

        return vc as? T
    }
```

Programski kod 3.3. Instance i createFromStoryboard metode kontrolera pogleda

3.2. Korištene tehnologije i alati

Programsko rješenje ostvareno je u obliku aplikacije za mobilni uređaj iPhone i operacijski sustav iOS. Većina korištenih alata usko su vezani uz razvoj mobilnih iOS aplikacija i uređaje te tehnologiju tvrtke Apple Inc. Uz njih, korištene su druge tehnologije, kao što su IBM Watson usluge te mobilna platforma Realm.

3.2.1. Komplet za razvoj programskih rješenja za iOS

Prema [17], iOS komplet za razvoj programskih rješenja (eng. *Softvare Development Kit*, *SDK*) omogućuje razvoj mobilnih aplikacija za iOS operacijski sustav tvrtke Apple Inc. On sadrži alate koji programerima omogućuju pristup raznim funkcijama i uslugama iOS uređaja kao što su sklopovski i programski atributi. Također sadrži i simulator uređaja iPhone koji oponaša izgled i dojam uređaja na računalu kako bi programeri mogli isprobavati aplikacije za vrijeme razvoja. iOS komplet za razvoj programskih rješenja podržava nekoliko programskih jezika. Najkorišteniji jezici su Swift i Objective-C. Neki od temeljnih dijelova iOS SDK su Cocoa Touch okvir za korisničko sučelje i Core Services skup sučelja koji se nalaze ispod korisničkog

sučelja.

3.2.2. Razvojno okruženje Xcode

Xcode je integrirano razvojno okruženje za MacOS operacijski sustav. Prema [21], on sadrži paket alata za razvoj programskih rješenja (eng. *software*) za MacOS, iOS, watchOS i tvOS. Xcode je razvijen od strane tehnološke tvrtke Apple Inc, a trenutna inačica je Xcode 9. Xcode podržava programske jezike C, C++, Objective-C, Java, AppleScript, Python, Ruby, ResEdit i Swift. Osim uređivača s automatskim popunjavanjem i prevoditelja programskih jezika, Xcode ima ugrađene alate za simulaciju iOS uređaja. Za vrijeme razvoja aplikacije ona se može pokrenuti i testirati na simulatoru raznih iPhone i iPad uređaja.

3.2.3. Uređaji i operacijski sustavi

Kako je Xcode razvojno okruženje za MacOS operacijski sustav, aplikacija je izrađena i testirana u Xcode-u na uređaju MacBook Pro i operacijskom sustavu MacOS Sierra. Aplikacija je, nakon izrade i testiranja na macOS sustavu, izgrađena na mobilni uređaj iPhone 7 koji na sebi ima operacijski sustav iOS 10. Aplikacija je nakon testiranja na uređaju MacBook Pro ponovno testirana na uređaju iPhone 7.

3.3. Korišteni programski jezici i okviri

U ovom radu korišten je programski jezik Swift koji je trenutno aktualan u programiranju iOS aplikacija. Platforma Realm korištena je za spremanje podataka u lokalnu bazu uređaja. Korišteni su okviri za vizualno prepoznavanje i prevođenje koji su dio usluga IBM Watson.

3.3.1. Programski jezik Swift

Swift je programski jezik za opću namjenu. Razvijen je od strane tehnološke tvrtke Apple Inc. i prvi put je predstavljen 2014. godine na Worldwide Developers Conference (WWDC). Prema [19], Swift je razvijen za iOS, macOS, watchOS, tvOS i Linux platforme te je dizajniran kako bi radio s Apple Cocoa i Cocoa Touch okvirima. Nasljednik programskog jezika Objective-C, Swift, dijeli metode sa svojim prethodnikom, no smatra se unaprjeđenjem u svim pogledima. Osim uljepšavanja sintakse (eng. *syntactic sugar*), Swift ima poboljšano rukovanje datotekama, rukovanje memorijom, performanse te uvodi opcionalne tipove i produžetke. Swift se može koristiti kao objektno-orijentiran ili funkcionalan programski jezik, no jedna od njegovih ključnih odlika je to što je ujedno i protokolno-orijentiran (eng. *protocol-oriented*). Protokoli su u drugim jezicima često nazvani sučeljem (eng. *interface*), oni jamče da će metode, svojstva ili

strukture pratiti i implementirati zadane zahtjeve. Protokoli su u Swiftu podržani produžetcima metoda i općim tipovima (eng. *generics*) što im omogućava korištenje za široku raznolikost slučajeva. Inačica Swifta korištena u aplikaciji je Swift 3 koja je službeno objavljena u rujnu 2016. godine.

3.3.2. Mobilna platforma Realm

Mobilna platforma Realm omogućuje spremanje podataka u stvarnom vremenu u lokalnu bazu koja se se nalazi na mobilnom uređaju. Prema [20], aplikacija koja koristi ovaj pristup može čitati podatke dok nije spojena na mrežu te se uskladiti s podatcima kada se ponovno spoji na mrežu. Mobilna baza podataka Realm rukuje stanjem mreže, Javascript Object Notation (*JSON*) dekodiranjem i rješavanjem konflikata.

3.4. Korišteni API-ji

Aplikacijsko programsko sučelje (eng. *Application programming interface*, *API*) je skup definicija, protokola i alata za izradu aplikacijskog rješenja. IBM je razvio mnoštvo usluga uz pomoć svog Watson računalnog sustava koji je dostupan kao set API-a i proizvoda na načelu program kao usluga (eng. *Software as a Service*, *SaaS*).

3.4.1. IBM Watson vizualno prepoznavanje

Usluga za vizualno prepoznavanje (eng. *visual recognition*), IBM Watson, koristi algoritme dubinskog učenja kako bi analizirala i prepoznala scene, objekte, lica i drugi sadržaj na slikama. Prema [22], usluga nakon analize sadržaja usluga daje odgovor s ključnim riječima koje pružaju informacije o tom sadržaju. IBM Watson ugrađene klase same po sebi daju vrlo precizne rezultate prepoznavanja, a mogu se dodati i vlastiti prilagođeni klasifikatori kako bi se napravilo daljnje treniranje algoritma za prepoznavanje s vlastitim slikama. IBM Watson API za prepoznavanje korišten je u ovom radu jer je dostupniji korisnicima od ostalih usluga kao što su Google Vision API ili Microsoft Cloud Vision API. U potpoglavlju 4.1 objašnjeno je programsko implementiranje ove usluge.

3.4.2. IBM Watson prevođenje jezika

Prema [23], usluga za prevođenje jezika (eng. *language translator*), IBM Watson, koristi se za izradu aplikacija koje identificiraju jezik unesenog teksta i koristi lingvistički model za prevođenje teksta na drugi jezik. Lingvistički model može se prilagoditi za pojedinu industriju ili druge vrste specijaliziranih terminologija kako bi se optimizirao za individualne potrebe. Usluga

trenutno podržava prevođenje na arapski, portugalski, francuski, njemački, talijanski i španjolski jezik. Ova usluga korištena je zbog dostupnosti, a, kako je korištena IBM Watson usuga za vizualno prepoznavanje, njihov drugi API bio je lakši za implementaciju. U potpoglavlju 4.1 objašnjeno je programsko implementiranje ove usluge.

4. PROGRAMSKO RJEŠENJE

Programsko rješenje ostvareno je pomoću arhitekture i alata navedenih u poglavlju 3. Aplikacija ima nekoliko važih i istaknutih odlika, kao što su spremanje podataka u lokalnu bazu uređaja Realm, prepoznavanje fotografija i prevođenje pojmova te bodovanje kvizova.

4.1. Odabir jezika i prepoznavanje objekata na fotografiji

Odabir jezika vrši se izborom jednog od jezika iz tabličnog pogleda. Jezik se sprema kao tip rječnik (eng. *dictionary*) u bazu podataka UserDefualts. Prema [10], rječnik pohranjuje poveznice između ključeva koji su obično tipa String i vrijednosti. Svaka vrijednost povezana je jedinstvenim ključem koji služi kao identifikator za tu vrijednost unutar rječnika. S obzirom da se vrijednosti u rječniku pretražuju po ključnim identifikatorima, rječnici nemaju određeno sortiranje kao neke druge vrste tipova kolekcija. Prema [15], klasa UserDefaults omogućuje interakciju sa sustavom zadanih postavki. Taj sustav omogućuje aplikaciji spremanje lokalnih postavki za aplikaciju na uređaj. Sustav se pokreće pokretanjem aplikacije te se pomoću metoda klase UserDefaults učitavaju podatci ili spremaju promjene. U prikazu programskog koda 4.1 vidi se svojstvo trenutnog jezika (eng. *current language*) koje se sprema u UserDefaults bazu pomoću metode *set(forKey:)* ili se dohvaća iz baze pomoću metode *object(for key:)*.

```
var currentLanguage: Language? {
    get {
        if let languageDictionary = object(forKey: "currentLanguage") as? [String: Any] {
            return Language(dictionary: languageDictionary)
        } else {
            return Language(name: "German", abrv: "de")
        }
    } set {
        set(newValue?.asDictionary, forKey: "currentLanguage")
    }
}
```

Programski kod 4.1. Spremanje trenutnog jezika u UserDefaults bazu

Svojstvo *currentLanguage* je tipa Language koji je Realm Object klasa, a ne struktura, kako bi se mogao spremati kao svojstvo drugog Realm objekta koji se sprema u lokalnu Realm bazu. U prikazu programskog koda 4.2 vide se svojstva objekta Language, ime (eng. *name*) i kratica (eng. *abbreviation, abrv*). Svojstva u objektu Realm moraju biti definirana kao *dynamic var* kako bi se mogla spremati u bazu Realm.

```
class Language: Object {
    dynamic var name: String = ""
    dynamic var abrv: String = ""

    convenience init(name: String, abrv: String) {
        self.init()
        self.name = name
        self.abrv = abrv
    }
}
```

Prikaz koda 4.2. Language Realm objekt

Kod pozivanja metode *set(forKey:)* na UserDefaults klasi, Realm objekt Language sprema se kao rječnik. Objekt se pretvori u rječnik pomoću metode asDictionary koja je vidljiva u prikazu programskog koda 4.3.

```
var asDictionary: [String: Any] {
        return ["name": name,
                "abrv": abrv
        ]
    }
```

Programski kod 4.3. Metoda asDictionary koja pretvara Realm objekt u rječnik

Kada korisnik odabere fotografiju iz galerije ili fotografira predmet kamerom uređaja, fotografija se pošalje metodi *getRecognition*, prikazanom progamskim kodom 4.4, koja stvori instancu VisualRecognition klase. Pomoću metode *getImagePath*, koja ima povratni tip URL, dobiva se URL fotografije iz lokacije na uređaju. URL fotografije se proslijedi metodi *classify* koja je metoda na objektu *visualRecognition*. Metoda *classify* vraća rezultate prepoznavanja usluge vizualnog prepoznavanja IBM Watson. U ovom rješenju uzimaju se u obzir prva četiri najvjerojatnija rezultata. Ukoliko postoji manje od četiri rezultata korisnik će dobiti upozorenje. U protivnom se rezultati prikažu na zaslonu kao što je prikazano u potpoglavlju 5.3.

```
func getRecognition(onComplete: @escaping (([String])-> Void)) {
        onStartedActivity?()
        let visualRecognition = VisualRecognition(apiKey: apiKey, version: version)

        let url = getImagePath()
        visualRecognition.classify(
            imageFile: url,
            failure: { error in
                print(error as NSError)
            },
            success: { images in
            guard let firstImage = images.images.first,
            let firstClassifier = firstImage.classifiers.first else { return }

            let classes = firstClassifier.classes

            if classes.count < 4 {
                self.onError?("Too few recognition results, please try to take another picture.")
                return
            }

            let firstFour = classes.prefix(4)

            let recognitions = firstFour.map { $0.classification }

            DispatchQueue.main.async {
                self.onEndedActivity?()
                onComplete(recognitions)
            }
        })
    }
```

Programski kod 4.4. Metoda getRecognition koja vraća rezultate prepoznavanja

Kada korisnik odabere točan rezultat prepoznavanja, taj se rezultat šalje metodi *translate*, prikazanoj u programskom kodu 4.5. Metoda *translate* primi riječ, to jest, tekst rezultata te stvori instancu klase LanguageTranslator. Dohvati se trenutni jezik iz *PersistenceService* klase koja sprema trenutni jezik u bazu podataka UserDefaults. Metoda *translate* je metoda na objektu *languageTranslator*. Ona vraća rezultat prevođenja usluge za prevođenje IBM Watson. Rezultat prevođenja prikaže se na zaslonu kao što je prikazano u potpoglavlju 5.3.

```
func translate(_ word: String) {
        onStartedActivity?()
        let languageTranslator = LanguageTranslator(username: username, password: password)

        guard let targetLanguage = persistenceService.currentLanguage?.abrv else { return }

        languageTranslator.translate(word,
                                     from: defaultLanguage.abrv,
                                     to: targetLanguage, failure: { (error) in
                                       print(error)
        }) { (translation) in
           guard let firstElement = translation.translations.first else { return }

           DispatchQueue.main.async {
              self.onEndedActivity?()
              self.onTranslation?(firstElement.translation)
           }
        }
    }
```

Programski kod 4.5. Metoda translate koja vraća rezultat prevođenja

4.2. Spremanje u lokalnu bazu mobilnog uređaja

Spremanje podataka u lokalnu bazu mobilnog uređaja učinjeno je pomoću mobilne platforme Realm. Kada se korisnik aplikacije odluči spremiti fotografiju i prijevod s kojima je zadovoljan zove se metoda *saveItem* koja koristi mobilnu platformu Realm koja je uvezena u projekt pomoću zaglavlja import RealmSwift. Realm sadrži metodu write koja sprema određeni Realm objekt u lokalnu bazu uređaja. Spremaju se originalni tekst prepoznatog objekta sa fotografije, prijevod na odabrani jezik, jezik za koji se prevodi te sama fotografija. Prikaz metode *saveItem* nalazi se pod prikazom programskog koda 4.6.

```
func saveItem(original: String, translation: String, image: Data) {
        let realm = try! Realm()

        try! realm.write {
            let newItem = QuizQuestion()

            newItem.originalText = original
            newItem.translatedText = translation
            newItem.language = persistenceService.currentLanguage
            newItem.image = image

            realm.add(newItem)
        }
        onItemSaved?()
        let when = DispatchTime.now() + 1
        DispatchQueue.main.asyncAfter(deadline: when) {
            self.onFinished?()
        }
    }
```

Programski kod 4.6. Spremanje u lokalnu bazu uređaja pomoću metode Realm write

4.3. Korisničko sučelje za učenje kroz rješavanje kvizova

Korisničko sučelje sastoji se od kontrolera pogleda koji je tipa *UIPageViewController*, koji u sebi sadržava druge kontrolere pogleda tipa *UIViewController*. *UIPageViewController* služi za iteraciju kroz veći broj kontrolera pogleda kao kroz stranice (eng. *pages*). Prema [16], kontroleri pogleda se dodaju u kontroler stranica pomoću metode *setViewControllers* koja prima argument liste kontrolera pogleda, koji je smjer listanja te je li prijelaz animiran. Metoda za dodavanje prvog kontrolera pogleda u kontroler stranica prikazana je prikazom programskog koda 4.7.

```
if let firstVC = questionViewControllers.first {
            self.navigationItem.title = "\(currentIndex + 1)/10"
            setViewControllers([firstVC],
                               direction: .forward,
                               animated: true,
                               completion: nil)
            currentIndex += 1
        }
```

Programski kod 4.7. Dodavanje prvog kontrolera u kontroler stranica

Nakon što se napravi prijelaz na sljedeću stranicu pomoću prelaska prstom po zaslonu, provjerava se je li to zadnja stranica ili se treba dodati još jedna stranica u kontroler kao u prikazu programskog koda 4.8.

```
func goToNextQuestion(_ isCorrect: Bool) {
        if isCorrect {
            points += 1
        }
        if currentIndex == questionViewControllers.count {
            viewModel.goToScore(points: points, image: viewModel.withImage, difficulty:
viewModel.pickedDifficulty)
        } else {
            self.navigationItem.title = "\(currentIndex + 1)/10"
            let vc = questionViewControllers[currentIndex]
            setViewControllers([vc],
                               direction: .forward,
                               animated: true,
                               completion: nil)
            currentIndex += 1
        }
    }
```

Programski kod 4.8. Provjera i navigacija na sljedeću stranicu

Kada dođe do zadnjeg kontrolera pogleda u kontroleru stranica, aplikacija navigira na zaslon s bodovima gdje se računaju bodovi s obzirom na težinu testa i točne odgovore. Uvjeti bodovanja vidljivi su u prikazu programskog koda 4.9.

```
func getTotalPoints() {

    var pointsWithModifier = 0

    switch difficulty as Difficulty {
    case .easy:
      if !withImage {
        pointsWithModifier = points * 3
        modifier = "x3"
      } else {
        pointsWithModifier = points
        modifier = "x1"
      }
    case .hard:
      if withImage {
        pointsWithModifier = points * 2
        modifier = "x2"
      } else {
        pointsWithModifier = points * 4
        modifier = "x4"
      }
    }
    persistenceService.totalScore += pointsWithModifier
  }
```

Programski kod 4.9. Računanje bodova s obzirom na težinu kviza i broj točnih odgovora

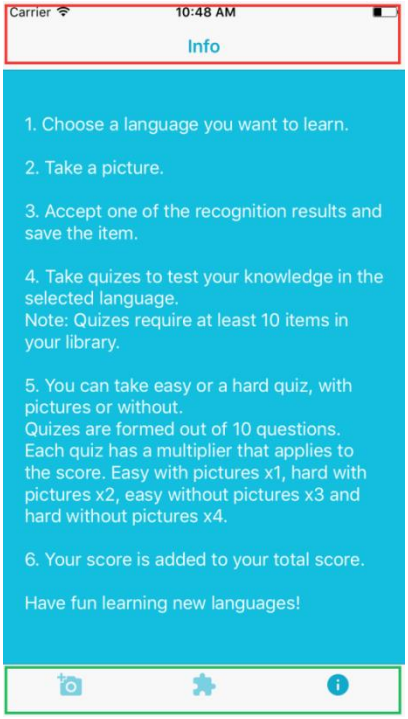
Bodovi se zatim dodaju ukupnim bodovima koji se spremaju u bazu UserDefaults, kao što je prikazano u prikazu programskog koda 4.10.

```
var totalScore: Int {
    get {
      return integer(forKey: "totalScore")
    } set {
      set(newValue, forKey: "totalScore")
    }
  }
```

Programski kod 4.10. Spremanje ukupnih bodova u UserDefaults bazu

5. KORIŠTENJE I TESTIRANJE APLIKACIJE

Svaki zaslon aplikacije sastoji se od glavnog sadržaja aplikacije, na slici 5.1 taj sadržaj je bijeli tekst na plavoj pozadini s informacijama kako koristiti aplikaciju. Uokvirena crvenom bojom na slici 5.1 je navigacijska traka (eng. *navigation bar*) na kojoj je vidljiv naslov svakog zaslona na sredini trake te se na lijevoj strani trake nalazi gumb za vraćanje na prethodni zaslon ako postoji ta opcija. Na slici 5.1. zelenom bojom uokvirena je traka kartica (eng. *tab bar*) koja omogućuje navigaciju između većih cjelina aplikacija. Lijevi gumb sa slikom fotoaparata vodi na zaslon za korištenje kamere opisan u potpoglavlju 5.1, srednji gumb sa slikom puzle vodi na izbornik kvizova opisan u cjelini 5.4, a desni gumb vodi na zaslon sa informacijama prikazan slikom 5.1.

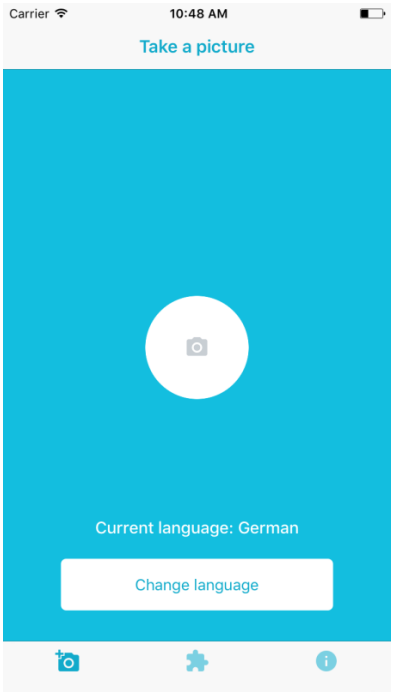


Sl. 5.1. *Zaslon sa informacijama i uputama kako korsistiti aplikaciju. Uokvireno crveno je navigacijska traka, a zeleno traka kratica*

Zaslon s informacijama se sastoji od šest točaka koje objašnjavaju korisniku aplikacije kako učinkovito koristiti aplikaciju.

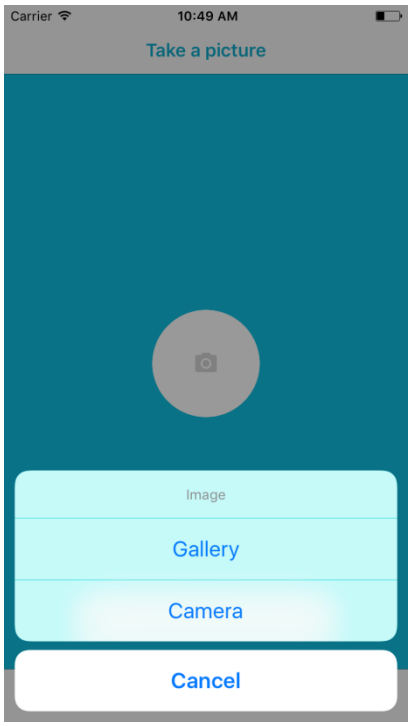
5.1. Korištenje kamere za fotografiranje i galerije za odabir slika

Početni zaslon aplikacije je prikazan slikom 5.2. Zaslon se sastoji od središnjeg gumba za odlazak na izbornik za fotografiranje prikazan slikom 5.3. Pri dnu zaslona nalazi se gumb koji na dodir vodi korisnika za izbornik jezika prikazan slikom 5.4. Iznad tog gumba nalazi se oznaka s navedenim trenutno odabranim jezikom.



Sl. 5.2. *Početni zaslon aplikacije sa izborom kamere i promjene jezika*

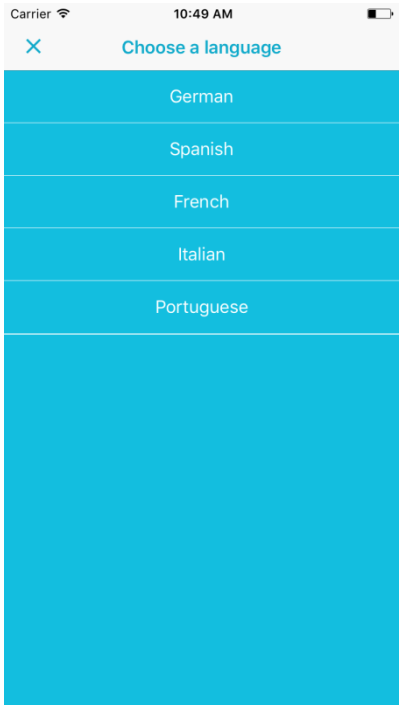
Izbornik za fotografiranje prikazan slikom 5.3 otvara tri opcije. Prva je opcija otvaranje galerije uređaja kako bi se odabrala ranije fotografirana fotografija. Druga opcija otvara kameru mobilnog uređaja kako bi korisnik fotografirao predmet koji želi. Treća je opcija poništenje ovog izbornika.



Sl. 5.3. *Izbornik kamere i galerije*

5.2. Izbor jezika

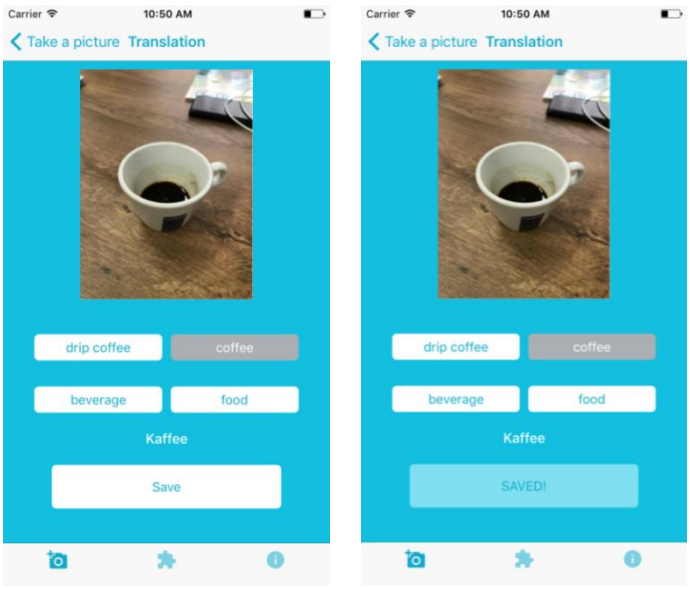
Izbornik jezika prikazan slikom 5.4. sadrži sve trenutne opcije jezika koje se nalaze u aplikaciji. Dodirom na jednu od opcija mijenja se trenutni jezik u aplikaciji, što znači da se mijenja prijevod u zaslonu za prevođenje i jezik za kvizove. Pritiskom na gumb označen s X na lijevoj strani navigacijske trake ovaj zaslon se poništava i korisnik se vrati na početni zaslon.



Sl. 5.4. *Izbor s tablicom za izbor jezika*

5.3. Odabir ispravnog prepoznavanja i spremanje u bazu podataka

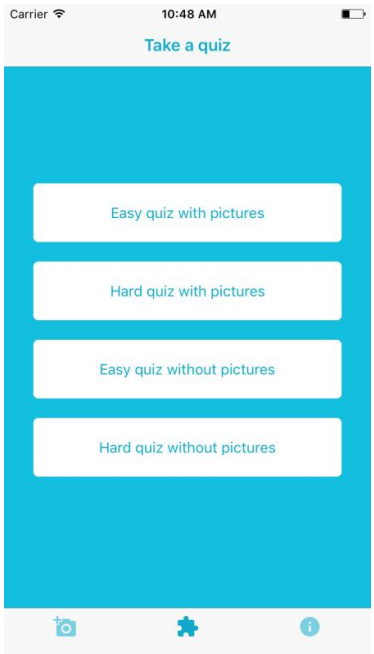
Nakon što korisnik odabere fotografiju iz galerije ili fotografira novu kamerom, aplikacija ga odvede na zaslon s prepoznavanjem i prevođenjem koji je prikazan na slici 5.5. Na zaslonu je prikazana odabrana fotografija. Ispod nje nalaze se četiri gumba koji predstavljaju najvjerojatnije prepoznavanje predmeta. Prvi gumb je označen te se za riječ ili riječi koje su ispisane na njemu ispod gumbova ispiše prijevod za te riječi. Odabirom drugog gumba promijeni se i prijevod. Kada je korisnik zadovoljan prepoznavanjem i prijevodom može taj predmet spremiti u lokalnu bazu uređaja kako bi se kasnije koristio u kvizovima. Za vrijeme spremanja gumb za spremanje postane neaktivan, kao što je prikazano na desnoj strani slike 5.5, te se zatim korisnik vrati na početni zaslon.



Sl. 5.5. *Izbor ispravnog prepoznavanja i spremanje u bazu*

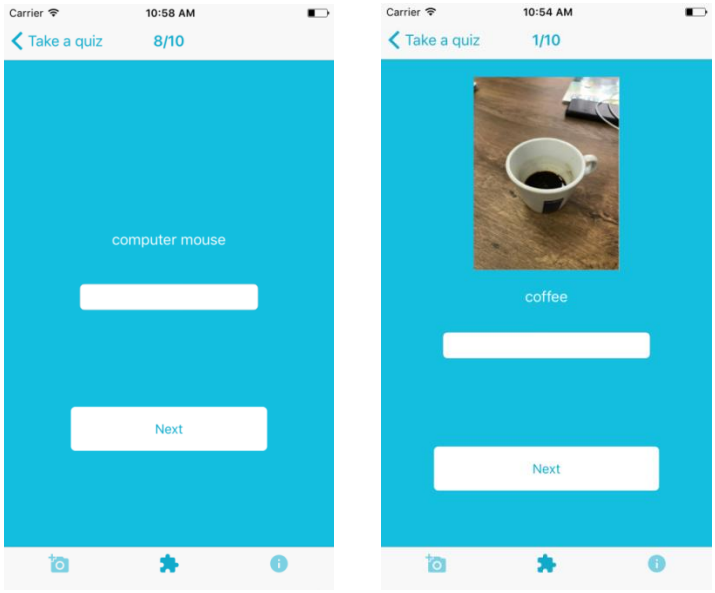
5.4. Odabir i rješavanje kvizova

Na slici 5.6 prikazan je izbornik kvizova. Na ovaj zaslon korisnik dolazi dodirom na srednji gumb na traci kartica. Izbornik daje korisniku izbor između četiri tipa kviza: lagan kviz sa slikama, težak kviz sa slikama, lagan kviz bez slika i težak kviz bez slika. Svaki od tih kvizova nosi različit broj bodova što je korisniku opisano na zaslonu s informacijama koji je prikazan slikom 5.1. Odabirom jedne od opcija korisnik započinje kviz ako u lokalnoj bazi uređaja postoji dovoljno artikala za stvaranje kviza. Testovi su uvijek stvoreni od deset nasumičnih artikala.



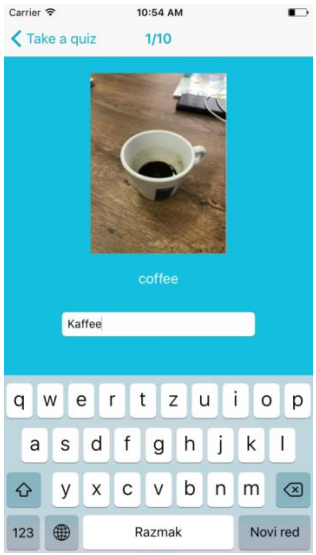
Sl. 5.6. *Izbornik kvizova*

Kvizovi mogu biti sa slikama ili bez njih. Zaslon za kviz bez slika sastoji se od broja pitanja na navigacijskoj traci, originalnog teksta na sredini zaslona, polja za unos prijevoda te gumba za odlazak na sljedeće pitanje. Zaslon za kviz sa slikom ima jednake elemente, jedina razlika je što se na gornjem dijelu zaslona nalazi fotografija koju je korisnik spremio zajedno s tekstom prepoznavanja i prijevodom. Zaslon kviza sa slikom prikazan je desnoj strani slike 5.7, a zaslon kviza bez slike na lijevoj strani iste slike.



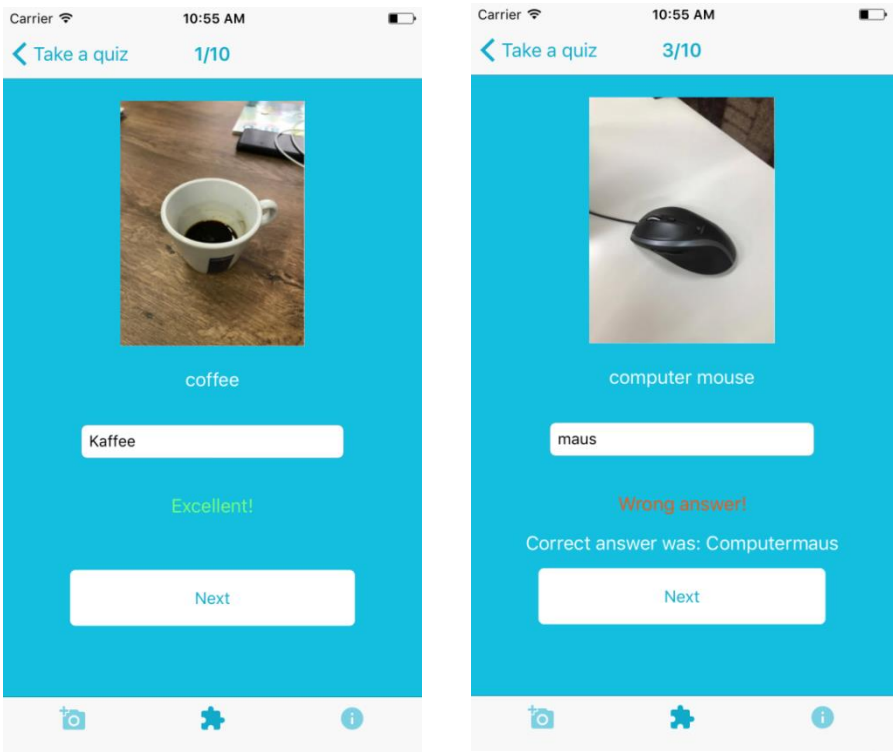
Sl. 5.7. *Kviz bez slike (lijevo) i kviz sa slikom (desno)*

Kao što je prikazano slikom 5.8., dodirom na polje za unos odogovora podiže se tipkovnica uređaja i korisnik može unijeti svoj odgovor. Pritiskom na potvrdu na tipkovnici ona se spušta.



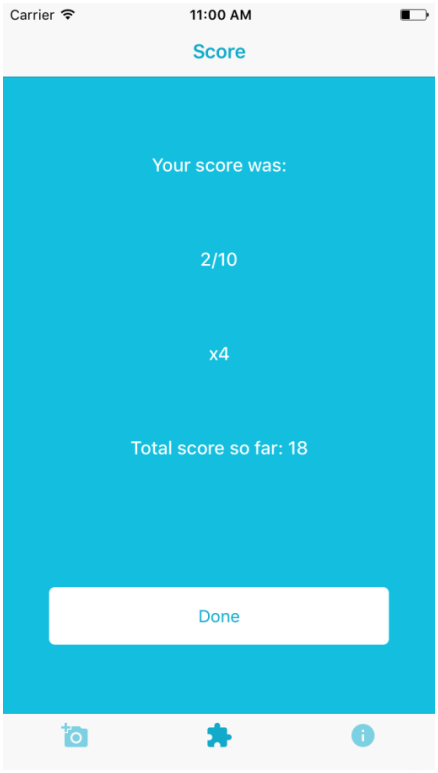
Sl. 5.8. *Ispunjavanje odgovora u kvizu*

Dodirom na gumb za odlazak na sljedeće pitanje aplikacija prikazuje zeleni tekst kao potvrdu točnosti odgovora ili crveni tekst koji ukazuje na to kako je odgovor netočan. Uz crveni tekst prikazano je i točno rješenje. Točan odgovor prikazan je na lijevoj strani slike 5.9, a netočan na desnoj. Nakon nekoliko sekundi korisniku je prikazano sljedeće pitanje.



Sl. 5.9. *Točan odgovor (lijevo) i netočan odgovor (desno)*

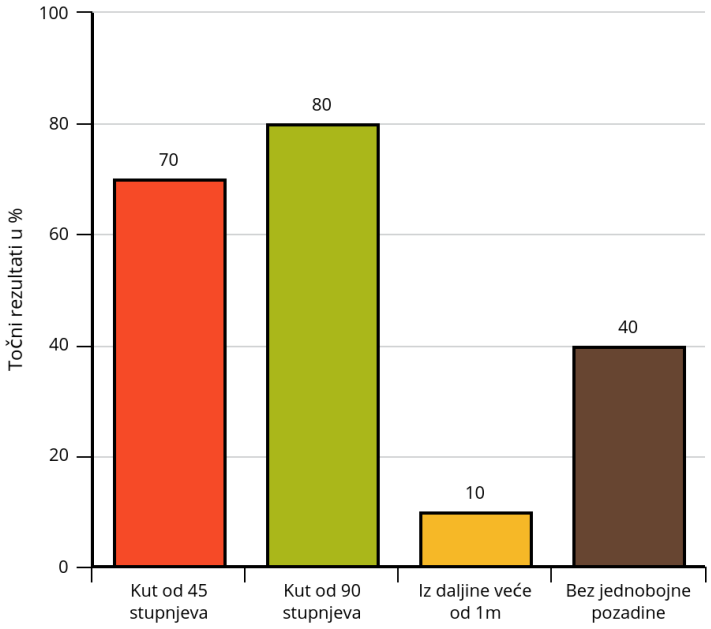
Ukoliko korisnik dođe do zadnjeg pitanja i dodirne gumb za sljedeće pitanje odveden je na zaslon s rezultatom kviza prikazan slikom 5.10. Zaslon se sastoji od ukupnog broja točnih odgovara od deset mogućih, modifikatora sa kojim se broj točnih odgovara množi prije nego se doda ukupnom broju bodova te ukupnog broja bodova. Modifikator ovisi o vrsti kviza koju je korisnik odabrao. Ukupan broj bodova su svi bodovi koje je do sada korisnik ostvario koristeći aplikaciju te je zajednički rezultat za sve jezike i kvizove.



Sl. 5.10. *Zaslon s bodovanjem*

5.5. Analiza točnosti vizualnog prepoznavanja

Točnost prepoznavanja analizirana je na skupu od deset predmeta različitih veličina i oblika. Predmeti su fotografirani na jednobojnoj pozadini pod kutem od otprilike 45 stupnjeva te 90 stupnjeva. Nakon toga su na istoj toj pozadini fotografirani na udaljenosti većoj od jednog metra. Na posljetku su fotografirani bez jednobojne pozadine. Rezultati su prikazani slikom 5.11.



Sl. 5.11. *Točnost prepoznavanja u postotku*

Većina rezultata se pokazala točnima kada su predmeti bili fotografirani na jednobojnoj pozadini po kutem od 45 ili 90 stupnjeva. Kada su predmeti fotografirani iz daljine, rezultat vizualnog prepoznavanja je u gotovo svim slučajevima bio vezan uz pozadinu. Rezultat bi bio soba ili stan. Rezultati za fotografije bez jednobojne pozadine su dali točne rezultate samo u slučajevima većih objekata kao što su vrata ili stol.

5.6. Testiranje aplikacije automatskim testovima

Prema [25], automatski testovi jedinica koda (eng. *unit test*) su metoda testiranja manjih jedinica izvornog koda programskog rješenja kako bi se odredila njihova ispravnost te provjerila funkcionalnost. U arhitekturi MVVM ovakvo testiranje obično se vrši na logici, to jest dijelu koji se nalazi u pogledu modela. Svaka metoda se posebno testira dok cijela klasa nije pokrivena testovima u cijelosti. Jedan takav primjer testa nalazi se u prikazu programskog koda 5.1. Prikazani test testira metodu *changeCurrentLanguage* koja mijenja trenutni jezik za prijevod u aplikaciji. Test provjerava je li jezik uspješno promijenjen nakon poziva metoda. Za ovakvo testiranje koriste se lažni podatci kako se ne bi radilo sa stvarnim podatcima aplikacije jer to narušava sigurnost podataka. Prema [27], za testiranje aplikacija napisanih u programskom jeziku Swift najčešće se koristi okvir XCTest s kojim se mogu integrirati testne mete za projekt.

```
func testChangesLanguageSuccessfully() {

    let expect = expectation(description: "onComplete called")

    viewModel.onComplete = { _ in
        XCTAssertEqual(self.viewModel.persistenceService.currentLanguage?.name, "French")
        XCTAssertEqual(self.viewModel.persistenceService.currentLanguage?.abrv, "fr")
        expect.fulfill()
    }

    viewModel.changeCurrentLanguage(to: Language(name: "French", abrv: "fr"))

    waitForExpectations(timeout: 0.2, handler: nil)
}
```

Programski kod 5.1. Test metode changeCurrentLanguage

5.7. Testiranje aplikacije korištenjem aplikacije

Prema [26], ručno testiranje aplikacije korištenjem je način testiranja programskog sučelja gdje osoba koja testira sučelje igra ulogu krajnjeg korisnika. Tester pokušava upotrijebiti sva svojstva aplikacije kako bi osigurao ispravno ponašanje. Takvo testiranje može i ne mora slijediti plan testiranja. Testiranje korištenjem također se može automatizirati kako bi se simuliralo korisnikovo ili testerovo korištenje aplikacije. To se obično radi za veće aplikacije i repetitivne testove, no s obzirom na veličinu aplikacije, u ovom radu se testiranje korištenjem nije automatiziralo.

Pogledi kontrolera, to jest korisničko sučelje i interakcija s njim, testirano je korištenjem aplikacije. Aplikacija se iznova koristi i provjeravaju se rubni slučajevi koji se mogu pojaviti kako bi se prouzročila greška ili nelogičnost u aplikaciji. U ovom slučaju, aplikacija je dana na testiranje vanjskom suradniku koji radi na osiguravanju kvalitete. Pronađene greške su otklonjene ili se njima rukuje unutar aplikacije.

Otkriveno je nekoliko manjih grešaka koje su otklonjene. Veće greške su pokrivene rukovanjem greškama kako bi se spriječilo rušenje aplikacije. Jedna od većih otkrivenih grešaka je rušenje aplikacije kada sustav za vizualno prepoznavanje ne vrati dovoljno rezultata, to jest manje od četiri rezultata. Druga greška je rušenje aplikacije kada u lokalnoj bazi uređaja ne postoji dovoljno objekata s kojima se može kreirati kviz. Minimalni broj objekata je deset, jer se kvizovi sačinjavaju od deset pitanja.

5.8. Rukovanje greškama

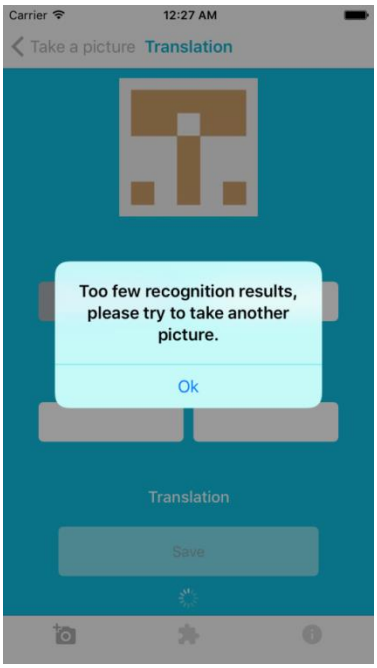
Prema [28], rukovanje greškama (eng. *error handling*) je proces reagiranja na i oporavka od

uvjete grešaka u aplikaciji. Uspješno izvršvanje nekih operacija unutar aplikacije nije uvijek zajamčeno, kao ni korisnost njihovog izlaza. Kada neka operacija ne uspije potrebno je rukovati greškom koja se pojavi. Prvo se treba provjeriti uvjet greške, a zatim je potrebno odrediti što će se prikazati korisniku ukoliko je došlo do greške. U prikazu programskog koda 5.2. provjerava se uvjet greške koji prikazuje grešku ako postoji manje od četiri rezultata prepoznavanja. Funkciji povratnog poziva onError predana je poruka koja će biti prikazana korisniku. U drugom dijelu istog prikaza koda vidljivo je kreiranje upozorenja koje će se prikazati na zaslonu korisnika.

```
// Uvjet greške i poruka koja se prikazuje na upozorenju u pogledu modela
        if classes.count < 4 {
            self.onError?("Too few recognition results, please try to take another picture.")
            return
        }

// Prikaz upozorenja na zaslonu korisnika u kontroleru pogleda
    viewModel.onError = { [weak self] error in
        let alert = UIAlertController(title: error, message: nil, preferredStyle: .alert)

        alert.addAction(UIAlertAction(title: "Ok", style: .default, handler: { action in
            self?.viewModel.cancel()
        }))

        self?.present(alert, animated: true, completion: nil)
    }
```

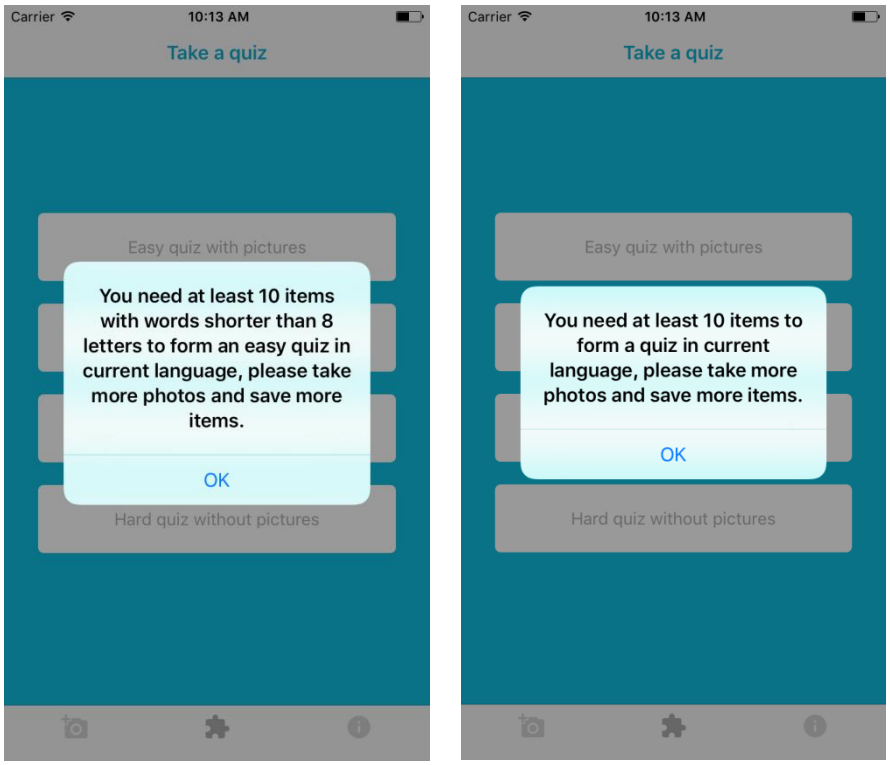
Programski kod 5.2. Rukovanje greškom za premalo rezultata prepoznavanja

Upozorenje koje korisnik vidi prikazano je na slici 5.12. Korisnik se dodirom na gumb upozorenja vraća na početni zaslon aplikacije.



Sl. 5.12. *Upozorenje za premalo rezultata prepoznavanja*

Osim upozorenja sa slike 5.12, pronađeno je još nekoliko grešaka kojima treba rukovati. Na slici 5.13 prikazano je rukovanje greškama u slučaju kada lagan ili težak kviz nemaju dovoljno artikala u lokalnoj bazi uređaja za kreiranje kviza.



Sl. 5.13. *Upozorenje za premalo artikala u bazi za lagani (lijevo) i teži (desno) kviz*

Korisnik ne može stvoriti kviz dok nema barem deset predmeta u bazi, ako stvara težak kviz, ili deset predmeta čiji je prijevod kraći od osam slova, ako stvara lagani kviz.

5.9. Korisničko iskustvo kroz korištenje aplikacija

Aplikacija je, za razliku od nekih drugih sa sličnim konceptom, interaktivnija i zanimljivija. Dok druge aplikacije imaju već određene zadatke, vizualni višejezični prevoditelj, zahtijeva korisnikovu uključenost u stvaranju daljnjih izazova. Aplikacija je izuzetno dobra za početnike, ali i za korisnike s nekim predznanjem određenog jezika. Također, zahtjeva od korisnika da sam traži predmete ili izraze koje želi naučiti izreći ili zapisati na drugom jeziku što daje dojam uključenosti korisnika, kao da sam stvara tu aplikaciju. Dobra je alterativa lijepljenju papirića po namještaju kada se uči strani jezik. Uz to, na kraju se može provjeriti naučeno kroz kvizove koje je korisnik sam kreirao. Samim ponavljanjem kviza, ako prvotni rezultat nije zadovoljavajući, korisnik stječe znanje koje mu treba.

6. ZAKLJUČAK

U ovom radu opisan je problem učenja na načelu rada računalnih igara. Predloženo je rješenje koje omogućuje učenje jezika kroz testove koje učenik sam stvara. Kao programsko rješenje načinjena je mobilna aplikacija za uređaj iPhone koja omogućuje korisniku fotografiranje sadržaja, vizualno prepoznavanje tog sadržaja i spremanje sadržaja u lokalnu bazu uređaja. Iz te baze aplikacija stvara kvizove kroz koje korisnik uči jezik. Na taj način korisnik aplikacije sam stvara sadržaj koji koristi za učenje. Učenje je personalizirano te motivira učenika kroz bodovanje.

Rezultati testiranja pokazuju da je potrebno rukovanje greškama kako se aplikacija ne bi rušila. Rukovanje greškama sa upozorenjima je dodano u aplikaciju. Korisničko iskustvo kroz korištenje aplikacije potvrđuje da je aplikacija prilagodiva svakom individulanom korisniku.

U budućem radu aplikacija se može proširiti dodatnim svojstvima kao što je registracija korisnika i usporedba rezultata s drugim korisnicima. Također se može dodati opcija brisanja određenih pojmova iz baze pojmova kada je korisnik siguran kako ih je svladao.

LITERATURA

[1] NTT DOCOMO Japan, Children's use of mobile phones, A special report 2014., Japan 2015.

[2] M. Krzeminska, LinguaLift, 10 best language learning apps, https://www.lingualift.com/blog/best-language-learning-apps/, pristupljeno: lipanj, 2017.

[3] Dora the Explorer, Wikipedia, https://en.wikipedia.org/wiki/Dora_the_Explorer, pristupljeno: lipanj, 2017.

[4] I. M. Ružić, M. Dumančić, Igrifikacija u odgoju i obrazovanju, Veleučilište Baltazar Zaprešić, , Zaprešić, Hrvatska, 2015.

[5] D. Codish, G. Ravid, Adaptive Approach for Gamification Optimization, IEEE/ACM 7th International Conference on Utlity and Cloud Computing, Izrael, 08-11 12 2014, pp. 609-610

[6] C. Gonzalez, A. Mora, P. Toledo, Gamification in Intelligent Tutoring Systems, TEEM Conference 2014, Salamanca, Španjolska, 01-03 10 2014, pp. 221-225

[7] CamFind application, http://camfindapp.com/, pristupljeno: rujan, 2017.

[8] B. Orlov, Medium, iOS Architecture Patterns: Demystifying MVC, MVP, MVVM and VIPER, https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52, pristupljeno: kolovoz, 2017.

[9] S. Khanlou, Khanlou blog, Coordinators Redux, http://khanlou.com/2015/10/coordinators-redux/, pristupljeno: kolovoz, 2017.

[10] Swift Collection Types , Apple Developer Swift Documentation, https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/CollectionTypes.html, pristupljeno: kolovoz, 2017.

[11] UIKit, Apple Developer Swift Documentation, https://developer.apple.com/documentation/uikit, pristupljeno: rujan, 2017.

[12] UIViewController, Apple Developer Swift Documentation, https://developer.apple.com/documentation/uikit/uiviewcontroller, pristupljeno: rujan, 2017.

[13] Swift Types, Apple Developer Swift Documentation, https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Types.html#//apple_ref/doc/uid/TP40014097-CH31-ID455, pristupljeno: rujan, 2017.

[14] Swift Protocols, Apple Developer Swift Documentation, https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html, pristupljeno: rujan, 2017.

[15] UserDefaults , Apple Developer Swift Foundation Documentation, https://developer.apple.com/documentation/foundation/userdefaults, pristupljeno: kolovoz, 2017.

[16] UIPageViewController, Apple Developer Swift Documentation, https://developer.apple.com/documentation/uikit/uipageviewcontroller, pristupljeno: kolovoz, 2017.

[17] iOS SDK, Wikipedia, https://en.wikipedia.org/wiki/IOS_SDK, pristupljeno: lipanj, 2017.

[18] Apple Inc., About Swift, https://swift.org/about/#swiftorg-and-open-source, pristupljeno: lipanj, 2017.

[19] Swift (programming language), Wikipedia, https://en.wikipedia.org/wiki/Swift_(programming_language), pristupljeno: lipanj 2017.

[20] Realm Mobile Platform, https://realm.io/products/realm-mobile-platform/, pristupljeno: lipanj 2017.

[21] Xcode, Wikipedia, https://en.wikipedia.org/wiki/Xcode, pristupljeno: lipanj 2017.

[22] Overview of the IBM Watson Visual Recognition service, https://www.ibm.com/watson/developercloud/doc/visual-recognition/index.html, pristupljeno: lipanj 2017.

[23] Overview of the Language Translator Service, https://www.ibm.com/watson/developercloud/doc/language-translator/index.html, pristupljeno: lipanj 2017.

[24] Watson Services, IBM, https://www.ibm.com/watson/developercloud/services-catalog.html, pristupljeno: lipanj 2017.

[25] Unit testing, Wikipedia, https://en.wikipedia.org/wiki/Unit_testing, pristupljeno: rujan, 2017.

[26] Manual Testing, Wikipedia, https://en.wikipedia.org/wiki/Manual_testing, pristupljeno: rujan, 2017.

[27] XCTest, Apple Developer Swift Documentation, https://developer.apple.com/documentation/xctest, pristupljeno: rujan, 2017.

[28] Error handling, Apple Developer Swift Documentation, https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/ErrorHandling.html, pristupljeno: rujan, 2017.

[29] MVVM-C With Swift, Marco Santarossa, https://marcosantadev.com/mvvmc-with-swift/, pristupljeno: rujan, 2017.

SAŽETAK

VIZUALNI VIŽEJEZIČNI PREVODITELJ ZASNOVAN NA NAČELU RADA RAČUNALNIH IGARA

U ovom radu opisan je problem učenja jezika na načelima rada računalnih igara metodom igrifikacije. Analizirana su ranija rješenja za učenje jezika kroz igru, te je predloženo rješenje za vizualno prepoznavanje objekata i prevođenje zapisanog tekstualnog oblika u neki drugi jezik. Na ovaj način personalizirano je učenje kroz igrifikaciju, svaki korisnik sam stvara svoje kvizove. Koristeći IBM Watson API za vizualno prepoznvanje i prevođenje te programski jezik Swift, programski okviri implementirani su u iOS aplikaciju. Aplikacija je testirana automatskim i ručnim testovima. Rezultati testiranja pokazuju da je potrebno rukovanje greškama kako se aplikacija ne bi rušila. Rukovanje greškama sa upozorenjima je dodano u aplikaciju. Korisničko iskustvo kroz korištenje aplikacije potvrđuje da je aplikacija prilagodiva svakom individulanom korisniku.

Ključne riječi: automatski testovi, igrifikacija, mobilna IOS aplikacija, prevođenje jezika, Swift, vizualno prepoznavanje.

ABSTRACT

VISUAL MULTILINGUAL TRANSLATOR BASED ON THE PRINCIPLE OF COMPUTER GAMES

In this paper, the problem of learning language via games was shown using the gamification method. Earlier options for learning languages through gaming were analyzed, and an algorithm for visual recognition and translation to other languages was proposed. This way learning is personalized through gamification, every user creates its own quizes. Using IBM Watson API for visual recognition and translation, and programming language Swift, frameworks were implemented into an iOS application. Application was tested by unit testing and manual testing. Testing results show that error handling is necessary so application would not crash. Error handling with alerts is added to the application. User experience through application use confirms that application is adaptive for every individual user.

Keywords: gamification, Swift, language translation, unit testing, visual recognition.

ŽIVOTOPIS

Marin Radoš rođen je 13.02.1993. u Osijeku. Pohađao je osnovnu školu „Vladimir Nazor“ u Čepinu. Nakon osnovne škole upisao je „Elektrotehničku i prometnu školu Osijek“ u Osijeku, smjer tehničar za računarstvo. Trenutno studira na „Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek“ u sklopu Sveučilišta Josipa Jurja Strossmayera u Osijeku gdje je upisao diplomski studij, smjer informacijske i podatkovne znanosti. Radi kao student iOS developer u tvrtci COBE d.o.o.

PRILOZI

Prilog 1. Završni rad u .pdf formatu

Prilog 2. Završni rad u .docx formatu

Prilog 3. Programski kod rada u obliku Xcode projekta