Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jaka Stavanja

Uporaba različnih biometričnih tehnik pri spletnem preverjanju znanja

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2018

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V izobraževanju so v zadnjem času vedno bolj razširjeni množični odprti spletni tečaji, kjer lahko uporabniki samostojno napredujejo po lekcijah in tudi enostavno sodelujejo v preverjanju znanja iz oddaljene lokacije (npr. od doma na lastnem računalniku). Pri tem se poraja vprašanje identifikacije uporabnika, saj je, ob odsotnosti osebnega stika, težko zagotoviti, da izbrano preverjanje znanja dejansko opravlja prava oseba. V okviru diplomskega dela raziščite področje uporabe različnih biometričnih tehnik pri spletnem preverjanju znanja, ki jih je mogoče uporabiti na strani odjemalca v kontekstu spletnega brskalnika. Na podlagi izsledkov implementirajte prototipni sistem, ki je v obliki vtičnika na voljo za odprtokodno platformo Moodle. Predlagano rešitev kritično ovrednotite in evaluirajte na dejanskih uporabnikih.

Rad bi se zahvalil mentorju doc. dr. Dejanu Lavbiču za strokovno pomoč in ideje, ko sam nisem vedel kako naprej. Za podporo ob študiju bi se rad zahvalil svoji družini, prijateljem, sošolcem, sosedom, sostanovalcu, vsem, ki ste me med študijem bodrili in vsem, ki me niste obremenjevali po nepotrebnem. V izogib zameram, bi se rad zahvalil tudi vsem, ki sem jih tu pozabil omeniti, a so mi kljub temu marsikdaj polepšali študentske dneve.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Motivacija 1

1.2 Pregled področja 2

1.3 Struktura dela 3

2 Pregled uporabljenih tehnologij **5**

2.1 Shema rešitve 5

2.2 Tehnologija za implementacijo logike za primerjavo stila tipkanja 5

2.3 Tehnologija za implementacijo logike za primerjavo slik obraza 7

2.4 Tehnologija za implementacijo osrednje točke API 7

2.5 Tehnologija za razvoj vtičnika za Moodle 8

3 Implementacija sistema **11**

3.1 Ideja 11

3.2 Postavljanje temeljev 12

3.3 Implementacija osrednje točke API in podatkovnih modelov . 15

3.4 Implementacija logike za primerjavo stila tipkanja 28

3.5 Implementacija logike za primerjavo slik obraza 31

3.6 Implementacija vtičnika za Moodle 34

4 Testiranje sistema **47**
4.1 Testiranje implementacije primerjave zapisa tipkanja 48
4.2 Testiranje prototipa sistema 51
4.3 Analiza SWOT 54

5 Zaključek **57**
5.1 Možne izboljšave in nadgradnje 57
5.2 Možnost uporabe sistema v prihodnosti 59
5.3 Sklepne ugotovitve 59

Literatura **62**

Seznam uporabljenih kratic

kratica	angleško	slovensko
MOOC	Massive Open Online Course	masovni prosto dostopni spletni tečaji
REST	Representational State Transfer	reprezentativno stanje prenosa
API	Application Programming Interface	aplikacijski programski vmesnik
HTML	Hypertext Markup Language	označevalni jezik za oblikovanje večpredstavnostnih dokumentov
CSS	Cascading Style Sheets	prekrivni slogi
PHP	PHP Hypertext Preprocessor	PHP predprocesor hiperteksta
SWOT	Strengths, Weaknesses, Opportunities, Threats	Prednosti, Slabosti, Priložnosti, Nevarnosti
SQL	Structured Query Language	strukturirani povpraševalni jezik
AJAX	Asynchronous JavaScript And XML	asihrona JavaScript in XML
XML	Extensible Markup Language	razširljivi označevalni jezik
DOM	Document Object Model	model objektov dokumenta
ORM	Object-relational mapping	objektno relacijsko preslikavanje
URL	Uniform Resource Locator	enolični krajevnik vira

YAML	YAML Ain't Markup Language	YAML ni označevalni jezik
TPR	True Positive Rate	delež resničnih pozitivnih primerov
FPR	False Positive Rate	delež lažnih pozitivnih primerov
AUC	Area Under Curve	površina pod krivuljo

Povzetek

Naslov: Uporaba različnih biometričnih tehnik pri spletnem preverjanju znanja

Avtor: Jaka Stavanja

Identifikacijo študentov pri reševanju kvizov preko spletne učilnice lahko trenutno izvajamo s preverjanjem posamezne osebe v živo. Primerjati moramo njen izgled s tistim iz slike na osebni oziroma študentski izkaznici. Če hočemo zmanjšati čas, porabljen za verifikacijo identitete, potrebujemo drug, učinkovitejši način overjanja. Časovni problem odpravimo s pomočjo razvoja vtičnika za programsko opremo Moodle in njemu pripadajoče zaledne logike. Ta zna s pomočjo spletnih obrazcev zajemati zapise stila tipkanja in slike obraza. Skupaj z vtičnikom razvijemo tudi zaledni sistem, ki zna primerjati slike obraza s pomočjo odprtokodne knjižnice OpenFace in računati razlike med zapisi stila tipkanja. Zanesljivost našega sistema na koncu tudi testiramo na prostovoljcih in ugotovimo, da je točnost sistema, ob uporabi obeh tipov primerjave, na prostovoljcih zelo visoka.

Ključne besede: biometrika, identiteta, spletno preverjanje znanja.

Abstract

Title: Use of biometric techniques in online exams

Author: Jaka Stavanja

The identification of students in exams is currently done by physically checking each and every student if their appearance and identity match the one on their student ID cards. If we wish to shorten the time needed to check students' identities, we need a more efficient way of verification. We solve this problem by developing a plugin for the Moodle learning management software, which is able to record prints of typing styles and images of faces by displaying an input form. Along with the plugin, we develop a backend system, that can distinguish faces using the OpenFace open-source library and typing styles. Finally, we test the system's reliability on volunteers and see that its accuracy is very high when using both comparison techniques.

Keywords: biometrics, identity, web, quiz, knowledge.

Poglavje 1

Uvod

Identifikacija oseb na preverjanjih znanja, ki se rešujejo preko spletne učilnice na fakultetnih računalnikih, še vedno poteka tako, da se tisti, ki spremlja reševanje kviza, na začetku odpravi do vsakega študenta posebej in preveri, če je prisotna oseba res tista za katero se predstavlja. To vzame izvajalcu kviza veliko časa, kar pa včasih vodi tudi k popolnem izogibanju preverjanja identitete študentov. V tem delu se bomo ukvarjali z izgradnjo vtičnika za spletno učilnico, ki teče na programski opremi Moodle. Ta vtičnik bo znal preko različnih biometričnih tehnik registrirati in nato preverjati uporabniško identiteto, brez da bi bilo treba vsako posamezno osebo preverjati v živo.

1.1 Motivacija

Ob začetnem pregledu področja ugotovimo, da se preverjanja znanja, ki se izvajajo preko spleta, hitro širijo in vedno več uporabljajo. Tako kot drugje po svetu, tudi na naši fakulteti. Po hitrem razmisleku ugotovimo tudi, da se na začetku kvizov, četudi so ti spletni, preverjanje identitete še vedno izvaja za vsakega posameznega študenta. To velikokrat izvajalcem kviza vzame veliko časa, kar pa lahko posledično s seboj prinese tudi površnost zaradi izgube koncentracije izvajalca kviza. Spletni kvizi se po navadi izvajajo v skupinah po največ trideset ljudi, da te lahko nadzorujemo. To pa pomeni,

da moramo za celotno generacijo rezervirati tudi več terminov in pripraviti več različnih oblik preverjanj znanja, da si med seboj ne bi izmenjevali rešitev. Če bi imeli na voljo alternativno metodo za preverjanje identitete pri spletnih kvizih, bi lahko le-te tudi izvajali kar v predavalnici, v enem samem terminu. V teh razmerah je to, zaradi časovne in prostorske stiske, težko izvedljivo.

Razvoj vtičnika za spletno programsko opremo Moodle ter pripadajočega zalednega sistema bi mnogim profesorjem in asistentom olajšal delo pri preverjanju identitete; le-to lahko to nalogo morda celo bolje opravi in hkrati skrajša čas, porabljen za celotno izvedbo kviza.

1.2 Pregled področja

Najsodobnejše rešitve za preverjanje identitete na digitalni način uporabljajo tako imenovani MOOC spletni portali, ki ponujajo plačljive in brezplačne študijske programe. Ti portali nimajo možnosti preverjanja študentov v živo, zato uporabljajo nekatere biometrične tehnike verifikacije oseb, ki jih bomo v tem delu pri svoji rešitvi tudi sami implementirali. Te so na kratko razložene tudi v publikaciji na temo MOOC programov, ki so jo napisali raziskovalci z univerze Stanford [18].

Portal Coursera, eden največjih ponudnikov MOOC programov, je od leta 2013 naprej za izdajo certifikatov opravljenih tečajev uporabljal biometrične tehnike identifikacije, natančneje, primerjavo stila tipkanja ter primerjavo slike obraza, zajete s spletno kamero, s sliko na osebni izkaznici, ki jo je uporabnik na začetku moral naložiti na portal Signature Track [7]. To tehniko so zaradi sprejetja kodeksa Coursera Honor Code [6] ukinili, saj svojim študentom v povprečju lahko zaupajo.

Primerjava zapisa stila tipkanja je sicer stara tehnika, ki sta jo že leta 1990 raziskovala Rick Joyce and Gopal Gupta [16]. Zaradi večletne uporabe na portalu Coursera, bomo tudi mi poskusili narediti podoben sistem za preverjanje identitete in raziskali, kako zanesljiv je lahko.

Izkaže se, da verifikacija s primerjavo stila tipkanja sama po sebi ni naj-

bolj zanesljiva, oziroma ni dovolj natančna, da bi jo uporabljali samostojno. Ravno zaradi njene slabše zanesljivosti bomo v našem sistemu kombinirali dve tehniki preverjanja identitete, ki bosta nedovoljene zamenjave ljudi zaznali z večjo verjetnostjo, kot če bi uporabljali samo eno izmed njiju. Svoje rešitve, kot je na primer primerjanje stila pisanja števil z miško, ponujajo tudi izključno v biometriko usmerjena podjetja, a so njihove rešitve zaprte in plačljive. V svojem sistemu bomo dve zgoraj navedeni tehniki, v podobnem sistemu, kot ga je uporabljala Coursera, implementirali kot kombinacijo zalednega sistema za preverjanje podatkov in vtičnika za spletno učilnico.

1.3 Struktura dela

Delo je strukturirano tako, da se v 2. poglavju najprej seznanimo s tehnologijami, ki jih bomo uporabili za izgradnjo vtičnika za Moodle in njemu pripadajoče zaledne logike, osrednje točke API in ostalih pomožnih programov. V 3. poglavju bomo predstavili implementacijo čelnega in zalednega dela sistema. V istem poglavju bomo razložili še nekatere pristope strojnega učenja in pridobivanja znanja iz podatkov, uporabljene v namene primerjanja zapisa tipkanja, ki jih bomo v nadaljevanju tudi sami implementirali ali uporabili obstoječe knjižnices. Našo implementacijo bomo nato v 4. poglavju tudi preizkusili na resničnih osebah, naš sistem poskušali izboljšati ter ga na koncu še enkrat analizirati. Zaključili bomo s 5. poglavjem, v katerem bomo podali zaključne ugotovitve o dodani vrednosti sistema, o kvaliteti njegovega delovanja in o možnostih za izboljšavo.

Poglavje 2

Pregled uporabljenih tehnologij

V tem poglavju bomo našteli in opisali vse tehnologije, uporabljene za izgradnjo našega sistema. Pri opisih bomo navedli tudi razloge za uporabo programskega jezika oziroma tehnologije.

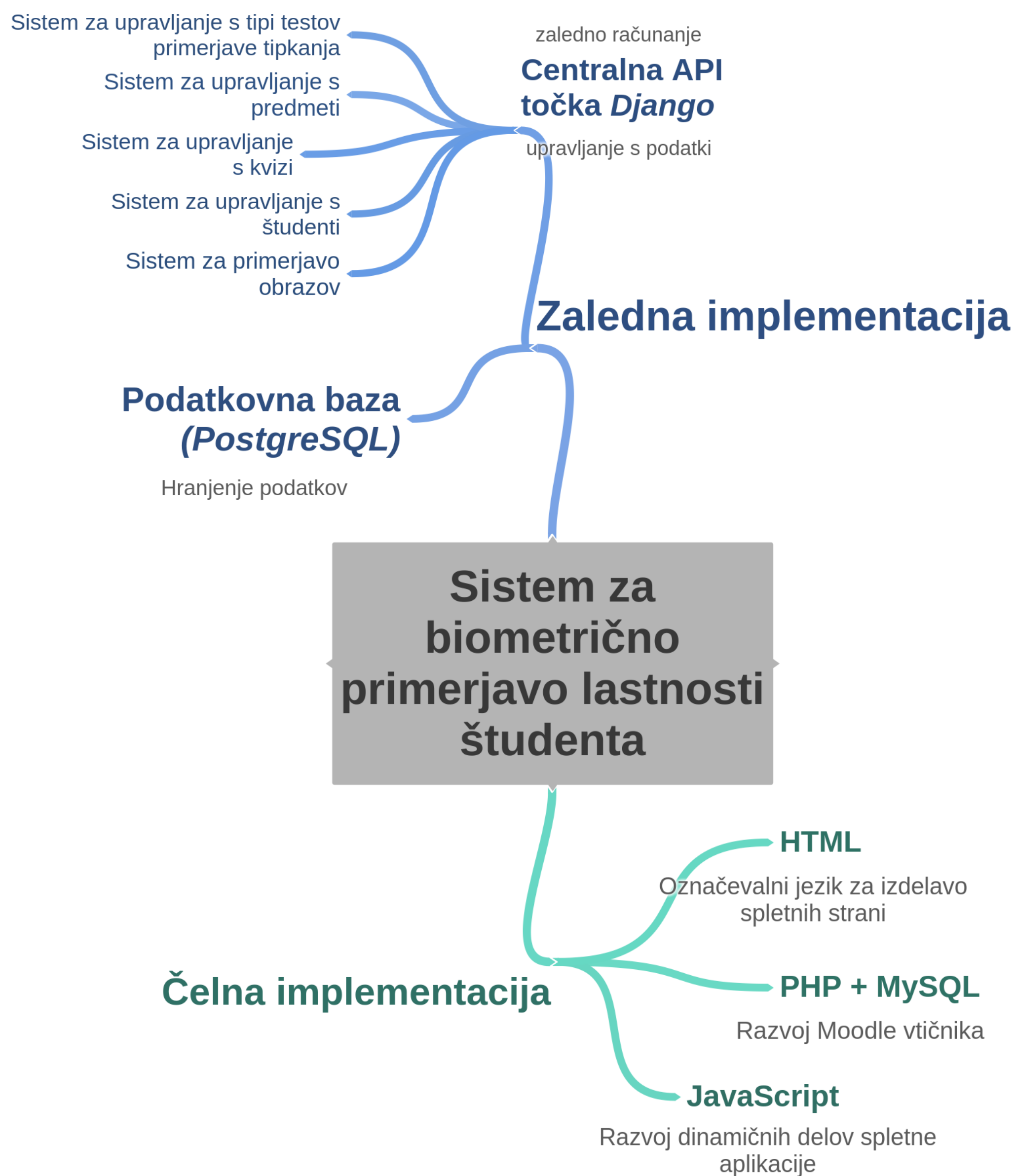
2.1 Shema rešitve

Rešitev bo sestavljena iz dveh delov, zalednega (točka API in nadzorna plošča) in čelnega (vtičnik), ki med seboj komunicirata preko protokola REST. Celotna struktura implementacije je prikazana na sliki 2.1.

2.2 Tehnologija za implementacijo logike za primerjavo stila tipkanja

2.2.1 Python

Logiko računanja razlik med stili tipkanja bomo implementirali v jeziku Python, čigar ekosistem ponuja številne knjižnice za strojno učenje in podatkovno rudarjenje, kar nam bo močno olajšalo delo [26]. Python je skriptni jezik, večinoma pa se uporablja za izvajanje različnih vrst programov na strežniku ali na osebnem računalniku izven brskalnika. Ima enostavno



Slika 2.1: Shema rešitve.

sintakso in omogoča hitro osvajanje znanja, tudi za programerje začetnike.

2.2.2 Knjižnice

Za računanje si bomo pomagali s knjižnico NumPy, ki ponuja kup pomožnih funkcij za hitrejše in bolj učinkovito programiranje splošne matematične ter algebraične logike [22].

2.3 Tehnologija za implementacijo logike za primerjavo slik obraza

Tudi pri implementaciji logike za primerjavo slik obraza bomo uporabili jezik Python, tokrat v sodelovanju s knjižnico OpenFace. Ponoven opis jezika zaradi preglednosti izpustimo.

2.3.1 OpenFace

OpenFace je odprtokodna knjižnica, ki s pomočjo globokih nevronskih mrež razlikuje in zaznava človeške obraze [1]. Napisana je v različnih jezikih, mi pa bomo, zaradi lažje integracije v naš sistem, uporabili različico napisano v jeziku Python. Knjižnico bomo uporabili za primerjavo razlik med sliko obraza, ki je zajeta s spletno kamero ter sliko obraza, ki je bila naložena ob vpisu v predmet. Tako bomo poskušali prepoznati reševalce kvizov, ki se izdajajo za drugo osebo.

2.4 Tehnologija za implementacijo osrednje točke API

2.4.1 Python knjižnica Django

Django je knjižnica, namenjena razvoju spletnih aplikacij v programskem jeziku Python, ki ponuja visokonivojsko abstrakcijo podatkovnih modelov

in enostaven ter hiter razvoj pogledov oziroma odgovorov na spletne zahtevke [9]. Poleg same knjižnice bomo uporabili tudi njen dodaten modul rest_framework, ki nam bo omogočil lažjo izdelavo točke API.

2.4.2 PostgreSQL

PostgreSQL je ena izmed najbolj razširjenih odprtokodnih implementacij relacijskih podatkovnih baz, ki za svoje poizvedbe uporablja jezik SQL [25]. Uporabili jo bomo namesto baze MySQL, saj v zadnjih letih močno pridobiva na ugledu in robustnosti, s pomočjo orodja Docker [10] pa jo je s knjižnico Django najlažje povezati. V bazo bomo zapisovali podatke, ki jih bomo pridobivali ob registraciji in testiranju uporabnikov ter podatke o različnih predmetih in kvizih, za katere bomo lahko shranili različne oblike testov tipkanja.

2.4.3 Protokol REST

Protokol REST je način implementacije komunikacije med odjemalcem in strežnikom preko mreže, ki ga je v svojem doktorskem delu leta 2000 opisal Roy Thomas Fielding [13]. Opredeljuje različne tipe spletnih zahtevkov iz strani odjemalca, kar je v večini primerov spletni brskalnik. Najbolj uporabljani štirje zahtevki so GET (pridobivanje podatkov), POST (spreminjanje in pridobivanje podatkov), PUT (shramba in považanje podatkov) in DELETE (brisanje podatkov). Preko zahtevkov GET in POST bomo tudi sami komunicirali s centralno Django točko API, ko bomo hoteli shraniti podatke ali pa te pridobiti znotraj vtičnika v spletni učilnici Moodle.

2.5 Tehnologija za razvoj vtičnika za Moodle

2.5.1 Moodle

Moodle je brezplačna odprtokodna programska oprema napisana v jeziku PHP. Na voljo je vsakomur, ki potrebuje sistem za spletno učilnico. Podpira

različne funkcionalnosti, kot so ustvarjanje različnih študijskih programov in njim pripadajočih modulov ter predmetov [20]. Predmeti lahko vsebujejo gradiva in ostale interaktivne vsebine. Ena od teh so tudi spletni kvizi, katerim lahko dodajamo vprašanja. Naš vtičnik bo ponudil nov tip vprašanja, ki bo znal vase vdelati obrazec, ki ga potrebujemo za izvedbo biometrične registracije oziroma primerjave.

2.5.2 HTML

HTML je označevalni jezik, uporabljen pri zasnovi oblike spletne strani, saj z njim povemo, kam bi radi postavili določene elemente, tako vizualno kot logično in kaj sploh predstavljajo ti elementi (to določimo s posebnimi tipi značk, ki jih ponuja jezik [28]). Z njim bomo zasnovali strukturo čelnega dela vtičnika za programsko opremo Moodle.

2.5.3 PHP

Spletna učilnica Moodle je napisana v jeziku PHP, kar pomeni, da morajo v istem jeziku biti zgrajeni tudi vsi njeni vtičniki. PHP je skriptni jezik, namenjen za izvajanje procesiranja hiperteksta (t.j. tekst, ki opisuje strukturo in vsebino spletne strani) in dodatne zaledne logike na strani strežnika [23]. Njegovo sintakso lahko enostavno vključimo kar med sintakso jezika HTML, ta pa se bo ob zahtevkih brskalnika prevedla v končno statično HTML kodo. Prva oblika jezika je izšla leta 1994, še danes pa ta poganja logiko večine spletnih strani na internetu. V kombinaciji s funkcijami za upravljanje s podatkovno bazo, bomo tudi tukaj morali uporabiti nekaj SQL poizvedb, saj Moodle v ozadju uporablja bazo MySQL.

2.5.4 CSS

CSS je stilna predloga na spletni strani, ki jo uporabljamo zato, da brskalniku povemo, kako bodo elementi iz HTML dokumenta vizualno izgledali in kje bodo postavljeni [8]. Za obliko čelnega dela našega vtičnika bo poskrbela tudi

zunanja knjižnica Semantic UI, ki vsebuje določene že vnaprej oblikovane komponente [27].

2.5.5 JavaScript

JavaScript je skriptni jezik, primarno namenjen izvajanju dodatne logike v brskalniku, ob obiskih spletnih strani. Z njim ponavadi spreminjamo HTML elemente znotraj strukture DOM in izvajamo zahtevke na zunanje strežnike. V našem konkretnem primeru ga bomo pri čelnem delu vtičnika uporabili za manipulacijo tekstov in izmenjavo podatkov s centralno točko API. To bomo izvajali s pomočjo klicev AJAX in uporabo knjižnice Axios [3].

Poglavje 3

Implementacija sistema

V tem poglavju bomo predstavili potek implementacije našega sistema. Najprej bomo postavili ogrodje in nato napisali logiko zalednega in čelnega dela sistema. V zalednem delu sistema bomo implementirali točko API, nadzorno ploščo za profesorje, logiko za primerjavo zapisov stila tipkanja in logiko za primerjavo obrazov. Čelni del bo zgrajen s pomočjo izdelave Moodle vtičnika, kateremu bomo najprej postavili ogrodje in omogočili nastavitve in prilagajanje znotraj spletne učilnice. Nato bomo izdelali še obrazce za registracijo in primerjavo ter sproti vse povezali z našim zalednim sistemom in podatkovno bazo.

3.1 Ideja

Zamislimo si zaledni sistem, ki profesorju omogoči ustvarjanje različnih obrazcev za primerjavo in zajem biometričnih lastnosti študentov na začetku vsakega preverjanja znanja. Vsak profesor lahko v sistemu ustvari več predmetov, ki jim vnaprej določi izbrano besedilo, katerega bo študent znotraj spletnega obrazca prepisoval na začetku vsakega preverjanja. Vsak tak predmet ima lahko več instanc kvizov oziroma preverjanj znanj, kar profesorju omogoči spremljanje rezultatov preverjanja biometričnih lastnosti za vsako posamezno spletno preverjanje znanja, ki ga bo izvajal pri nekem predmetu.

3.2 Postavljanje temeljev

Najprej izdelamo ogrodje sistema in postavimo njegove temelje tako, da bo nadaljnji razvoj v prihodnje karseda enostaven in se nam ne bo treba ukvarjati s konfiguracijskimi težavami. Za postavitev temeljev in okolja bomo uporabili programsko opremo Docker.

3.2.1 Docker

Da lahko učinkovito poganjamo sisteme med različnimi napravami, ki tečejo na različnih operacijskih sistemih, uporabimo funkcionalnosti programske opreme Docker [10] in podsistemov, ki jih ponuja. Njihova največja prednost je enostavno kopiranje instanc, najpogosteje osnovanih na podsistemu Linux. Na njih teče programska oprema, ki je neodvisna od glavnega strežnikovega operacijskega sistema.

Take mikrosisteme z lastnim datotečnim sistemom in prednaloženo programsko opremo ter knjižnicami imenujemo vsebniki [11]. Ti so popolnoma neodvisni od zunanjega operacijskega sistema in nam omogočajo vedno identične pogoje za razvoj, ne glede na to, kje aplikacijo poganjamo. Hkrati se z njihovo uporabo izognemo tudi vsem začetnim težavam, ki nas lahko doletijo. Pri gradnji obsežnejših sistemov se velikokrat srečamo s problemi konfiguracije potrebne programske opreme, na katerih temeljijo.

Za poganjanje zalednega dela naše aplikacije potrebujemo vzpostavljeno Docker okolje s potrebnimi vsebniki. Orodje docker-compose, ki je že vgrajeno v Docker, nam omogoča gradnjo več vsebnikov hkrati in s tem enostavno postavitev okolja za poganjanje in razvoj strežnika API. To orodje skrbi tudi za enostavno povezovanje več vmesnikov med sabo, s preprosto preslikavo njihovih IP naslovov in vrat. Najprej definiramo vsebnik za našo točko API. Ta zna poskrbeti, da je znotraj njega nameščena vsa programska oprema in vse potrebne knjižnice, da lahko razvijemo Django točko API, ki bo uporabljala funkcionalnosti iz knjižnice Openface.

Definiramo še drugi vsebnik, ki gosti PostgreSQL podatkovno bazo in bo

```
1 RUN ls
```

Izsek 3.1: Primer datoteke Dockerfile z ukazom RUN, ki znotraj vsebnika zažene ukaz ls.

hranila vse podatke o osebah, predmetih in kvizih. Vsebovala bo tudi vse informacije, potrebne za izvajanje preverjanja identitete reševalcev kvizov na čelnem delu. Ta vsebnik je tesno povezan s prvim, ki se nanj povezuje ob vseh SQL poizvedbah.

Vsebniki se gradijo s pomočjo datotek tipa Dockerfile. Primer ukaza iz take datoteke lahko vidimo na izseku 3.1. S pomočjo različnih ukazov navedemo programsko opremo, ki jo želimo naložiti znotraj vsebnika, da bo ta znal delovati tako, kot si sami želimo.

Predloge Dockerfile datotek in skripte za gradnjo vsebnikov lahko brezplačno prenesemo tudi iz portala Docker Hub [12], ki je portal z brezplačnimi, odprtokodnimi diskovnimi slikami in poenostavljenimi rešitvami, ki nam prihranijo pisanje svojih Dockerfile datotek, katere lahko naloži kdorkoli. Te slike vsebujejo že nameščeno programsko opremo, ki jo veliko ljudi pogosto potrebuje. Za potrebe gostovanja podatkovne baze uporabimo predlogo *postgresql* s tega spletišča, za izgradnjo vsebnika za Django točko API pa sami priredimo Dockerfile datoteko, ki nam jo za potrebe delovanja knjižnice Openface ponujajo njeni avtorji (poimenovali jo bomo *web*). Dodamo ji ukaze, ki poskrbijo, da so znotraj vsebnika nameščeni vsi potrebni moduli za delovanje knjižnice Django.

Ko imamo definirane datoteke tipa Dockerfile in izbrane slike iz portala Docker Hub, lahko sestavimo datoteko docker-compose.yml, ki v formatu YAML definira hierarhijo naših vsebnikov in njihove povezave (našo datoteko prikazuje izsek 3.2). S pomočjo te datoteke lahko enostavno gradimo in poganjamo več vsebnikov hkrati.

Ko zgradimo vse potrebne vsebnike, lahko te tudi poženemo. Na našem

```yaml
version: '3.1'

services:

  web:
    # Dockerfile vsebnika web je v trenutni mapi
    build: .

    # Ob zagonu poženemo spodnji ukaz
    command: python manage.py runserver 0.0.0.0:8000

    # V vsebnikov direktorij /code preslikamo vsebino trenutne mape
    volumes:
      - .:/code

    # Vrata 8000 iz vsebnika preslikamo na lokalna vrata 8000
    ports:
      - "8000:8000"

    # IP naslov vsebnika db se preslika v besedo db
    depends_on:
      - db

  db:
    # vsebnik db uporablja sliko postgres iz Docker Hub-a
    image: postgres
```

Izsek 3.2: Datoteka docker-compose YAML, ki za oba naša vsebnika definira kako naj se zgradita, kako sta povezana in kaj se z njima zgodi ob zagonu.

računalniku oziroma strežniku, s pomočjo ukaza "docker-compose up" zaženemo vso potrebno programsko opremo za razvoj zalednega dela naše aplikacije. Na voljo imamo knjižnico Openface in vse njene funkcionalnosti ter podlago za naš Django zaledni sistem.

Za izgradnjo ogrodja projekta nam knjižnica Django ponuja ukaz "django-admin startproject", ki avtomatsko ustvari osnovne datoteke za začetek razvoja. Naš projekt poimenujemo "biometrics".

Preko nastavitev naš novi projekt, v avtomatsko ustvarjeni datoteki "settings.py", povežemo z našim *db* vsebnikom. Namesto IP naslova baze lahko uporabimo kar besedo *db*, saj za preslikavo poskrbijo vsebniki. Tako imamo postavljeno ogrodje, ki nam omogoča nadaljnji razvoj.

3.3 Implementacija osrednje točke API in podatkovnih modelov

Kot smo v prejšnjem podpoglavju omenili, za razvoj osrednjega API dela našega sistema uporabimo knjižnico Django, ki ponuja ogrodje za gradnjo spletnih aplikacij v jeziku Python. Ker bo celoten zaledni del naše točke API napisan v tem jeziku, bo dostopanje do funkcionalnosti knjižnice Openface enostavno, saj nam ne bo treba povezovati več kot zgolj dveh istojezičnih datotek.

3.3.1 Django aplikacije

Dokumentacija knjižnice Django predlaga, da se različne funkcionalnosti vsakega sistema razdeli v t.i. aplikacije. Zato s pomočjo orodja "manage.py", ki je priloženo vsakemu ustvarjenemu Django projektu, ustvarimo pet aplikacij:

- *course*, ki bo vsebovala podatkovne modele, potrebne za hrambo informacij o različnih predmetih znotraj spletne učilnice in potrebno funkcionalnost za upravljanje z njimi,

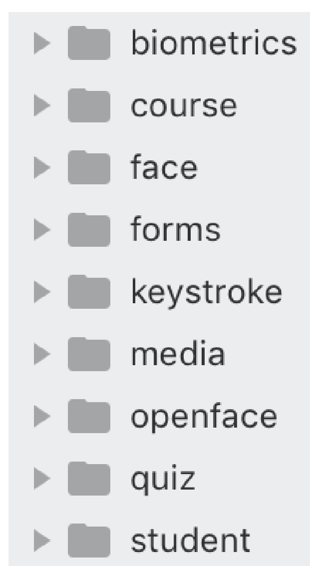
```
1 docker-compose run web python manage.py startapp course
```

Izsek 3.3: Primer manage.py ukaza iz terminala, ki ustvari novo aplikacijo course znotraj docker vmesnika web.

- *quiz*, ki bo vsebovala modele o različnih kvizih, ki se bodo dodajali in izvajali pri posameznih predmetih iz aplikacije *course*,
- *student*, ki bo vsebovala modele in funkcionalnost za upravljanje z informacijami o študentih, ki jih bomo povezali z našim sistemom preko unikatnega ključa iz sistema Moodle,
- *keystroke*, ki bo vsebovala funkcionalnosti in modele s podatki o tipih testov zapisa tipkanja in časi, zajetimi pri opravljanju kviza in
- *face*, ki bo vsebovala vse funkcionalnosti, potrebne za primerjavo obrazov študentov.

Novo aplikacijo ustvarimo s pomočjo orodja "manage.py" in ukaza startapp. Če želimo ukaze poganjati znotraj vsebnika, moramo uporabljati orodje docker-compose in ukaz run. Prvi argument ukazu mora biti ime vsebnika, temu pa sledi dejanski ukaz. Primer uporabe ukaza startapp znotraj našega Docker vsebnika *web* je vidna na izseku 3.3.

Končna datotečna struktura aplikacij znotraj našega sistema je prikazana na sliki 3.1. Mapa "biometrics" je bila ustvarjena ob kreaciji našega Django projekta, ki smo ga tako poimenovali. Vsebuje potrebne konfiguracijske datoteke za celoten zaledni sistem in obenem tudi datoteko "urls.py", kjer lahko povezujemo poglede z naslovi URL. Več informacij o pogledih sledi v podpoglavju 3.3.3. V mapo "media" se shranjujejo slikovne datoteke, ki jih naložijo študentje ob prvi registraciji v sistem. Direktorij "openface" vsebuje vse datoteke iz istoimenske knjižnice. Nastane ob prvem zagonu našega



Slika 3.1: Datotečna struktura aplikacij znotraj zalednega sistema.

Docker vsebnika *web*, čigar vzpostavitvena datoteka Dockerfile vsebuje ukaz, ki klonira Openface-ov repozitorij Git v našo aplikacijo.

3.3.2 Podatkovni modeli

Ko imamo ustvarjeno strukturo aplikacij znotraj projekta (slika 3.1), moramo definirati potrebne podatkovne modele. Django, za ustvarjanje in upravljanje s podatki, vsebuje že priložene funkcije, ki jih lahko uporabimo znotraj datoteke "models.py" vsake aplikacije. Vsakemu Django modelu moramo definirati imena polj in njihove podatkovne tipe, na način, ki ga kaže izsek 3.4.

Najprej se lotimo izdelave modelov aplikacije *course*. Ustvarimo model Course, ki bo hranil podatek o imenu predmeta iz spletne učilnice in povezavo (tuji ključ) na tip testa zapisa tipkanja, ki ga bomo kasneje definirali v aplikaciji *keystroke*.

V aplikaciji *quiz* ustvarimo podatkovni model Quiz, ki za vsako preverjanje znanja hrani ime preverjanja, njegov kratek opis, datum izvajanja in

```
class Course(models.Model):
  owner = models.ForeignKey(User, on_delete=models.CASCADE)
  name = models.CharField(max_length=200)
  keystroke_test_type =
   ↪  models.ForeignKey(KeystrokeTestType,
                                          on_delete=models.CASCADE)
```

Izsek 3.4: Primer Django podatkovnega modela, ki vsebuje dva tuja ključa (owner in keystroke_test_type) in polje name, ki vsebuje niz, dolg največ 200 znakov.

povezavo (tuji ključ) na predmet, ki mu pripada.

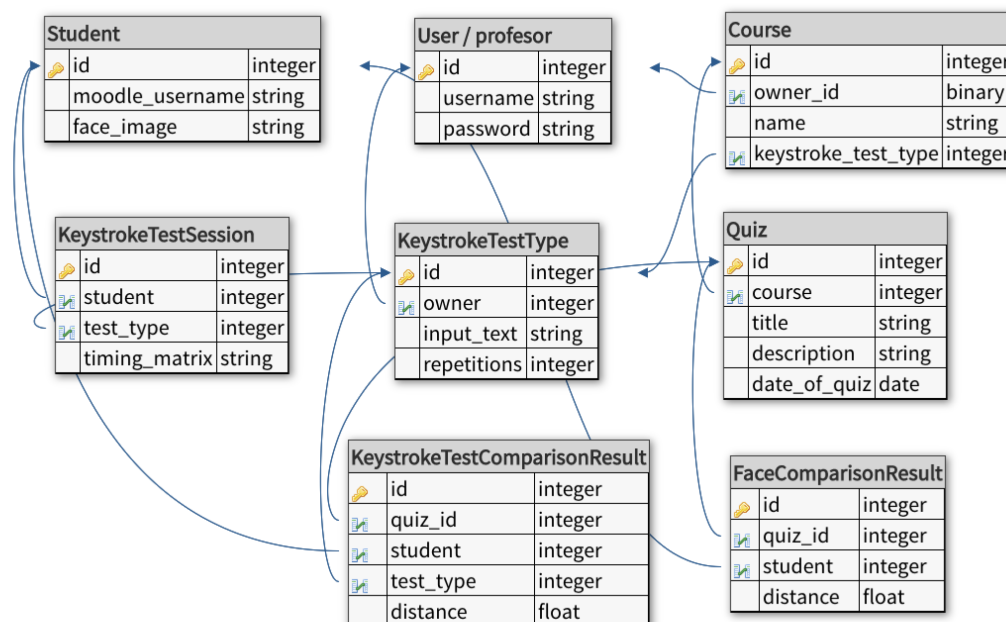
Znotraj aplikacije *student* nato ustvarimo model Student, ki bo hranil študentov unikaten identifikator (ID) iz sistema Moodle in pot na datotečnem sistemu do slike, ki vsebuje njegov obraz. Vsaka instanca modela, za primarni ključ dobi unikatno identifikacijsko številko, ki jo bomo uporabljali znotraj našega zalednega sistema.

Model User, ki simbolizira profesorjev račun, se ustvari avtomatsko ob inicializaciji Django projekta, zato nam ga ni treba dodajati.

Nadaljujemo z definicijo modelov za hranjenje zapisov stila tipkanja in struktur testov. To storimo znotraj prej ustvarjene aplikacije *keystroke*. Ustvarili bomo dva nova podatkovna modela:

- KeystrokeTestType, ki hrani podatke o obliki nekega testa stila tipkanja (dejanski tekst, ki ga mora študent prepisati ob preverjanju znanja in registraciji ter število potrebnih ponovitev pri prepisovanju) in

- KeystrokeTestSession, ki bo vseboval podatke o tem, kdo je test oziroma "session" opravljal (v obliki tujega ključa) in vektorje izmerjenih časov med pritiski tipk, ki jih bomo potrebovali za primerjave. Vsebuje tudi povezavo na tip testa iz modela KeystrokeTestType (kot tuji



Slika 3.2: Končna struktura in povezave modelov sistema.

ključ), ki nam pove, kateri tip testa je oseba izvajala. Tako vemo, če sploh lahko dva različna opravljanja primerjamo med seboj.

Da lahko profesor pregleduje izračunane razdalje med zapisi stila tipkanja bomo te morali tudi shranjevati, za kar naredimo model KeystrokeTestComparisonResult, ki hrani rezultate testa, povezavo na kviz v obliki primarnega ključa in povezavo na osebo, ki je kviz reševala, ravno tako v obliki primarnega ključa.

Tako kot bomo shranjevali razdalje med zapisi stila tipkanja bomo shranjevali tudi izračunane podobnosti obrazov. Znotraj aplikacije *face* zato dodamo nov model FaceComparisonResult, ki bo podobno kot pri aplikaciji *keystroke*, hranil rezultate primerjave slik obraza vsakega študenta. Poti do originalnih slik obraza so shranjene že znotraj modela Student, te pa bomo za vsakega uporabnika naložili na datotečni sistem takoj ob registraciji v sistem.

Ko imamo ustvarjene modele in definirane povezave med njimi (slika 3.2),

```
python manage.py makemigrations
python manage.py migrate
```

Izsek 3.5: Primer uporabe manage.py ukazov iz terminala, ki ustvarita vse potrebne migracijske datoteke in jih poženeta.

moramo zgenerirati in pognati tudi migracijske datoteke. Te sistemu povedo, katere SQL ukaze mora izvesti na podatkovni bazi, da se ta prilagodi novi strukturi podatkovnih modelov. To storimo s pomočjo orodja manage.py in ukazov makemigrations ter migrate, kot je razvidno iz izseka 3.5. Za boljšo preglednost bomo v nadaljevanju izpuščali ukaze iz orodja docker-compose, ki jih moramo dodati pred vsak naš ukaz.

3.3.3 Splošni REST API pogledi

Pogled (ang. view), je v Django projektih razred, ki zna ob spletnem zahtevku preko URL naslova ali drugačnih parametrov pridobiti podatke, ki jih pošlje odjemalec, izvesti procesiranje in v obratni smeri nazaj poslati rezultat v določeni obliki. Naši pogledi na zahtevke odgovorjajo s pošiljanjem podatkov v obliki objekta JSON ali pa v obliki odgovora HTML.

Da bomo pri registraciji uporabnikov v določen predmet lahko preverili, če ima uporabnik zanj že shranjen zapis stila tipkanja in sliko svojega obraza, znotraj aplikacije *course* razvijemo pogled CourseStudentStatusView, ki ob spletnem zahtevku tipa GET nazaj v formatu JSON vrne vse, za vtičnik potrebne informacije. Django nam omogoča enostavno ustvarjanje pogledov s predlogami, ki so knjižnici priložene. Primer pogleda, ki uporablja predlogo View, lahko vidimo na sliki 3.6.

Če želimo dostopati do funkcionalnosti pogleda iz nekega zunanjega sistema, moramo najprej določiti URL naslov, na katerega bomo pošiljali zahtevke. To storimo v datoteki urls.py v mapi *biometrics*, ki smo jo omenili v

```
class IAmADjangoView(View):

  def get(self, request, *args, **kwargs):

      return JsonResponse({
        "foo": "bar"
      })
```

Izsek 3.6: Primer Django pogleda, ki ob GET zahtevku vrne odgovor v podatkovni obliki JSON.

```
urlpatterns = [
    url(r'^url_naslov/(?P<pk>\d+)',
     ↪ IAmADjangoView.as_view())
]
```

Izsek 3.7: Primer povezave URL naslova z Django pogledom v datoteki urls.py.

poglavju 3.3.1. V seznam urlpatterns znotraj te datoteke dodamo regularni izraz, ki predstavlja obliko našega URL naslova in njemu pripadajoč pogled. Ko v datoteki "urls.py" definiramo na katerem naslovu URL je dostopen nek pogled, lahko v regularni izraz dodamo tudi del, ki ga bo Django obravnaval kot parameter pri zahtevku. Primer regularnega izraza je na voljo na izseku 3.7.

Med znakoma "manjše kot" in "večje kot" zapišemo ime spremenljivke, ki bo parameter pri zahtevku prestregla in bo na voljo v pogledu (v primeru na izseku bo ta parameter v pogledu dostopen pod spremenljivko self.kwargs['pk']). Če parametre pri zahtevkih pošljemo kot telo zahtevka

in ne preko parametrov v naslovu, lahko te preberemo preko posebnih spremenljivk, ki jih avtomatsko napolni Django. Do parametrov zahtevka POST dostopamo preko funkcije "request.POST.get", do parametrov zahtevka GET pa preko funkcije "request.GET.get". Obema funkcijama kot argument podamo ime parametra, ki ga želimo pridobiti in privzeto vrednost. Tako lahko enostavno izmenjujemo podatke med čelnim in zalednim delom sistema.

3.3.4 Nadzorna plošča

Kot dodatek osrednji točki API pripravimo tudi funkcionalnosti za upravljanje in pregledovanje podatkov, katere bodo lahko uporabljali izvajalci kvizov. Do njih bodo dostopali preko vizualnega urejevalnika oziroma nadzorne plošče. Našemu osrednjemu API sistemu zato dodamo dodatne poglede, ki bodo s pomočjo HTML predlog na določenih URL naslovih prikazovali podatke iz modelov aplikacije in obrazce za njihovo urejanje. Polnjenje in prikazovanje HTML predlog lahko enostavno naredimo z uporabo Django predloge TemplateView. Kot je razvidno iz primera na izseku 3.8, moramo razredu, ki uporablja to predlogo podati ime naše HTML datoteke.

Če želimo končno HTML strukturo napolniti s podatki, moramo znotraj pogleda napolniti objekt "context". V HTML datoteki lahko do vsake spremenljivke znotraj tega objekta dostopamo tako, da uporabimo par zavitih oklepajev in par zavitih zaklepajev. Med njiju vstavimo ime spremenljivke, po končanem zahtevku pa bo ta sklop nadomeščen z njeno vrednostjo. Primer izpisa spremenljivke je viden na izseku 3.9.

Znotraj vsake HTML datoteke, ki jo uporablja predloga TemplateView, lahko uporabljamo tudi dodatno logiko s pomočjo sintakse, ki jo ponuja Python. Primer uporabe zanke for-in, ki zna iterirati skozi vsa polja določenega seznama vidimo na izseku 3.10.

Za vsako aplikacijo znotraj našega sistema lahko sedaj ustvarimo različne poglede, ki jih dodatno zavarujemo z Djangu priloženim avtentikacijskim modulom. Ta bo omogočil, da ima vsak izvajalec kviza svoj račun in znotraj nadzorne plošče vidi samo svoje predmete, kvize in tipe testov stila tipkanja.

```
class TemplateViewPogled(LoginRequiredMixin, TemplateView):

  template_name = "izpis.html"

  def get_context_data(self, **kwargs):
    context = super(DashCourseList, self)
                    .get_context_data(**kwargs)
    context["spremenljivka"] = "Pozdravljen, svet!"
    return context
```

Izsek 3.8: Primer razreda, ki uporablja predlogo TemplateView in za izpis uporablja HTML datoteko izpis.html, poleg tega pa je zavarovana z avtentikacijskim modulom.

```
<html>
<head>
</head>
<body>
  <div>
  {{ spremenljivka }} <!-- Pozdravljen, svet! --!>
  </div>
</body>
</html>
```

Izsek 3.9: Primer izpisa spremenljivke v HTML datoteki, ki se pošlje skozi razred TemplateViewPogled (definiran na izseku 3.8) in v brskalniku izpiše Pozdravljen, svet!

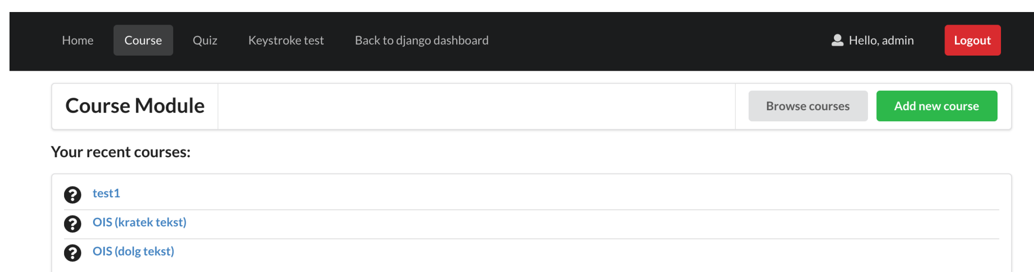
```
<head>
</head>
<body>
  {% for polje in seznam %}
    {{ polje }}
  {% endfor %}
</body>
</html>
```

Izsek 3.10: Primer uporabe zanke for-in za izpis vseh polj znotraj v pogledu definirane spremenljivke context[”seznam”].

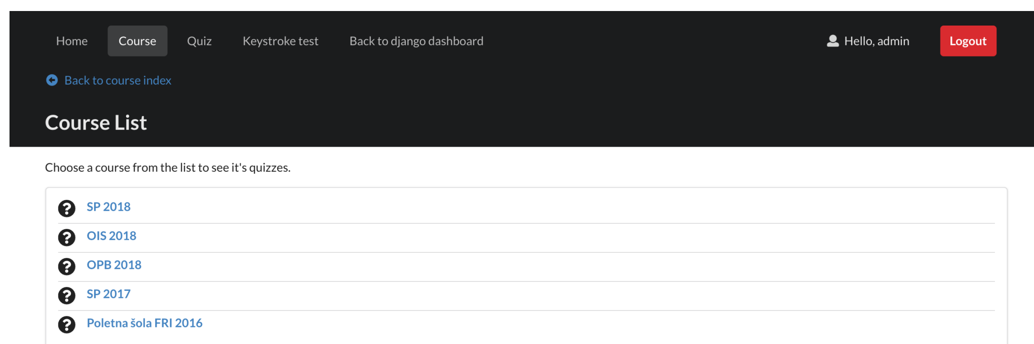
To storimo z dodatnim filtriranjem v pogledu in dodatkom LoginRequired-Mixin, ki ga posredujemo kot argument pri definiciji razreda našega pogleda (definicija pogleda je vidna na izseku 3.8).

Pogledi, potrebni za izgradnjo nadzorne plošče so v našem projektu petih tipov:

- *index*, ki prikaže vse splošne informacije in možne akcije iz nekega modula (v sliki 3.3 lahko vidimo primer takega pogleda za model Course),
- *list*, ki izpiše vse podatke o modelih, ki so direktni potomci trenutnega modela (primer za model Course je viden na sliki 3.4),
- *details*, ki izpiše podrobnosti o trenutnem modelu in ostale potrebne informacije (primer za model Quiz je na sliki 3.5),
- *add*, ki izpiše obrazec s potrebnimi polji za ustvarjanje novih instanc modelov v podatkovni bazi (na sliki 3.6 lahko vidimo obrazec za dodajanje novega tipa testa stila tipkanja) in



Slika 3.3: Pogled tipa index za model Course, ki izpiše tudi možne nadaljne akcije (dodajanje novega predmeta in brskanje po obstoječih).



Slika 3.4: Pogled tipa list za model Course, ki izpiše vse obstoječe predmete, ki jih je trenutno prijavljeni izvajalec kviza ustvaril.

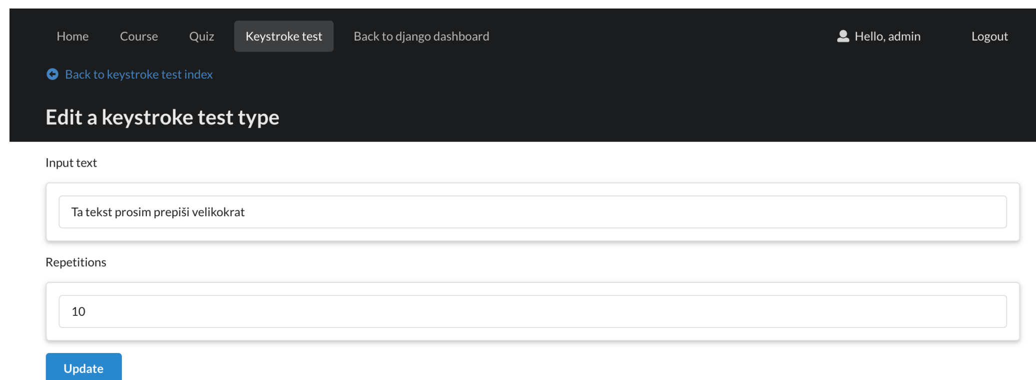
- *edit*, ki izpiše obrazec enake oblike kot tip *add*, z razliko, da so njegova polja že napolnjena (podatke pridobi iz baze ob prisotnosti primarnega ključa za določen model). Preko takega obrazca lahko že ustvarjene objekte v podatkovni bazi spreminjamo.

Za boljšo uporabniško izkušnjo dodamo pogled tipa index za našo celotno aplikacijo na korenski URL (slika 3.8).

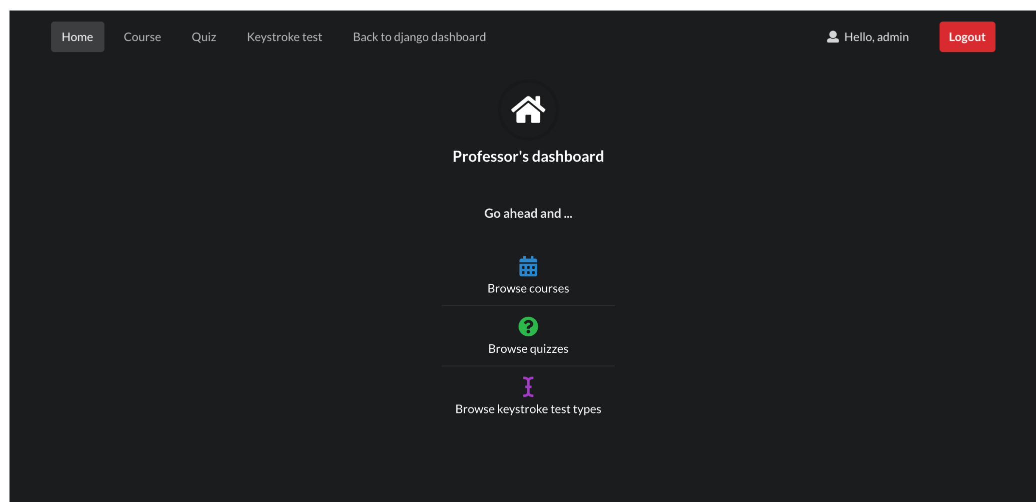
Na tem mestu imamo pripravljeno osnovno ogrodje za našo aplikacijo, kar pomeni, da lahko nadaljujemo z implementacijo posameznih delov sistema, ki zajemajo in računajo z biometričnimi podatki.

Slika 3.5: Pogled tipa detail za model Quiz, ki izpiše vse podatke o kvizu in rezultate primerjav, ki so se izvedle pri študentih, ki so ta kviz reševali.

Slika 3.6: Pogled tipa add za model KeystrokeTestType, ki vsebuje obrazec za ustvarjanje novih instanc modela.



Slika 3.7: Pogled tipa edit za model KeystrokeTestType, ki vsebuje obrazec za ustvarjanje novih instanc modela.



Slika 3.8: Index domači pogled naše nadzorne plošče, ki je dostopen na korenskem URL naslovu.

3.4 Implementacija logike za primerjavo stila tipkanja

3.4.1 Implementacija končnih točk in pogledov za Moodle vtičnik

V Moodle vtičniku bomo potrebovali informacije o besedilu, ki ga pri določenem kvizu študent prepisuje v okence in število ponovitev prepisovanja, da bomo lahko prikazali pravilen obrazec. V ta namen v aplikaciji *quiz* definiramo pogled QuizInfoAPIView, ki ob zahtevku s priloženim primarnim ključem trenutnega kviza nazaj vrne podatke o predmetu, kateremu kviz pripada in vse potrebne podatke za izpis registracijskega ter testnega obrazca.

Potrebujemo tudi pogled, ki nam omogoča shranjevanje podatkov, ki jih bomo zajemali v teh dveh obrazcih. Znotraj aplikacije *student* za potrebe registracije študenta v predmet definiramo nov pogled StudentAPIView, ki ob POST zahtevku iz vtičnika prejme informacije o prijavljenem uporabniku in originalni vektor časov pritiskanja tipk za trenutni kviz. Za potrebe primerjave obraza prejme tudi sliko osebe, a se to zgodi zgolj ob prvi registraciji v naš sistem (neodvisno od predmeta). V primeru, da se študent prvič registrira v naš sistem, v podatkovni bazi ustvarimo novo instanco modela Student in mu pripnemo vse potrebne informacije. Če se je ta študent prej registriral v nek drug predmet, moramo ustvariti samo novo instanco objekta KeystrokeTestSession, ki ga povežemo s trenutnim predmetom in obstoječo instanco modela Student.

3.4.2 Primerjalna logika

Logiko za primerjavo zapisov stila tipkanja razvijemo sami v jeziku Python s pomočjo matematične knjižnice NumPy, ki nam olajša delo s števílkami. To storimo v datoteki "detector.py", ki jo ustvarimo znotraj naše aplikacije *keystroke*. Najprej moramo vedeti, kako se bomo lotili primerjave. Če se obrnemo k članku, kjer sta Fabian Monrose in Aviel D. Rubin opisala pri-

stope za primerjavo stilov tipkanja [19], v tretjem poglavju vidimo, da si moramo najprej zamisliti objekt, ki predstavlja stil tipkanja. Potem lahko iz njega črpamo informacije in z njimi računamo, na koncu pa še klasificiramo uspešnost primerjave dveh stilov. V našem primeru bomo z merjenjem časov med pritiski tipk zapisali stile tipkanja v matrike, izmerili razdalje med njimi in postavili najvišjo dovoljeno mejo razlikovanja. S pomočjo te meje bomo rezultat primerjave stila lahko klasificirali kot sumljiv ali nesumljiv. Ustvarimo razred Detector, ki v konstruktorju sprejme potrebne matrike časov. Vsebuje funkcijo "get_distance", ki vrne razdaljo med dvema stiloma tipkanja.

Za primerjavo razdalje tipkanja uporabimo podoben princip kot Kevin S. Killourhy in Roy A. Maxion iz univerze Carnegie Mellon, ki sta v članku o primerjavi različnih tipov razdalj med stili tipkanja implementirala svoj primerjalnik stila tipkanja [17]. Kot je tudi navedeno V čelnem delu sistema bomo poslušali pritiske tipk študenta, ki bo prepisoval določen tekst v obrazec. Primerjali bomo naslednje časovne intervale:

- časovni interval med pritiskom in spustom trenutne tipke,
- časovni interval med pritiskom prejšnje in pritiskom trenutne tipke in
- časovni interval med spustom prejšnje in pritiskom trenutne tipke.

Za primerjavo bomo potrebovali dve matriki časovnih intervalov. Prva bo originalna matrika vektorjev časovnih intervalov med pritiski tipk, ki jih bomo zajemali ob registraciji uporabnikov v predmete. Smatramo jo kot pravi zapis stila tipkanja tega uporabnika. Vsaka vrstica znotraj matrike vsebuje zajete časovne intervale ene ponovitve prepisovanja teksta.

Kot primer lahko za tekst izberemo besedo "lol" in 2 za število ponovitev. Čas med pritiskom in spustom trenutne tipke označimo z oznako D (down), čas med pritiskom prejšnje in pritiskom trenutne tipke z oznako DD (down-down) in čas med spustom prejšnje in pritiskom trenutne tipke z oznako UD (up-down). V indeksih oznak bomo uporabljali črke na tipkah, med katerimi merimo razdalje. Vsaka vrstica matrike bo pri prepisovanju besede "lol" vsebovala naslednje čase:

$$[D_l, D_o, DD_{lo}, UD_{lo}, D_l, DD_{ol}, UD_{ol}].$$

Lahko bi uporabljali samo eno ponovitev prepisovanja daljšega teksta, a se izkaže, da je točnost primerjave ob več ponovitvah prepisovanja krajšega teksta boljša, kot pa enkratna ponovitev daljšega teksta (več o tem v poglavju 4). Vse vrstice pri primerjavi povprečimo po stolpcih in tako dobimo zapis študentovega stila tipkanja pri trenutnem poskusu.

Dva stila tipkanja moramo nato znati tudi primerjati in oceniti podobnost. V naši nadzorni plošči si zato želimo za opazovanje odstopanja čim bolj opisne mere. Zamislimo si mero, ki nam v odstotkih pove kako dobro se dva stila tipkanja ujemata. Za vsak par števil v vektorjih časov, ki nastanejo kot povprečeni stolpci matrike iz prejšnjega odstavka, lahko izračunamo ujemanje in za končen rezultat vzamemo njihovo povprečje po formuli:

$$match_{\%} = \frac{\sum_{i=1}^{n} \frac{min(q_i, p_i)}{max(q_i, p_i)}}{n},$$

kjer je q originalni vektor povprečnih časov in p trenutni vektor povprečnih časov.

Razmerje med minimumom in maksimumom si lahko predstavljamo kot mero, ki nam pove za koliko odstopa trenuten čas od originalnega.

Funkcionalnosti za primerjavo stila tipkanja shranimo v datoteko "detector.py", ki jo uvozimo v datoteko s pogledi in uporabljamo v pogledu KeystrokeTestDistanceAPIView. V tem pogledu preko POST zahtevka prejemamo potrebne podatke o časih, tipu testa in zajetih časih. Med objekti modela KeystrokeTestSession v podatkovni bazi poiščemo primerni originalni test študenta, ki se smatra kot pravilni (izsek 3.11). Pri tem iskanju se morata ujemati tip (oblika) testa in pa identifikator trenutnega uporabnika, ki rešuje kviz.

Podatke o originalnih in trenutnih zajetih časih študenta lahko sedaj posredujemo uvoženemu razredu Detector in pokličemo funkcijo "get_distance" ter ta podatek shranimo v bazi preko modela KeystrokeTestComparisonResult. Te shranjene podatke nato prikažemo pri rezultatih kviza na nadzorni

```
1 KeystrokeTestSession.objects.filter(student=user_id,
  ↪ test_type=test_type_id)
```

Izsek 3.11: Ukaz, ki s pomočjo identifikatorjev trenutnega uporabnika in tipa testa stila tipkanja poišče originalni zapis, zajet ob registraciji.

plošči za profesorje.

3.5 Implementacija logike za primerjavo slik obraza

3.5.1 Openface

Za primerjavo slik obrazov bomo uporabili knjižnico Openface, ki deluje s pomočjo globokih nevronskih mrež [1]. Knjižnica Openface zna, s pomočjo modela, treniranega na več sto tisočih slikah obrazov, zaznavati obrazne lastnosti, jih pretvoriti v vektor značilk in te med seboj razlikovati. Poleg tega zna tudi rezati vhodne slike tako, da iz njih izloči samo del, kjer je prisoten obraz in tega tudi poravnati. Tako so vsi obrazi, ki jih primerja, enako postavljeni.

3.5.2 Primerjalna logika

Pri implementaciji primerjalne logike se bomo držali originalnih navodil iz dokumentacije in primerov iz knjižnice Openface.

Za primerjavo obrazov bomo implementirali vse potrebne funkcije znotraj datoteke "detector.py" v aplikaciji *face*, v kateri bomo ustvarili razred Detector. Ta bo prejel poti do originalne in trenutne slike obraza, podobnost med njima pa bo vrnil ob klicu funkcije "get_similarity". Najprej poskrbimo, da

```
1   lokacija_obraza =
    ↪  openface.AlignDlib(model_za_predikcijo_obraza).getLargestFaceBoundingBox(rgb_slika)
```

Izsek 3.12: Ukaz, ki na sliki v formatu RGB poišče pozicijo največjega obraza.

je knjižnica Openface pravilno skonfigurirana znotraj datoteke, kar storimo s pravilno povezavo priloženega modela za detekcijo obraza. Kot argumenta pri ustvarjanju instance objekta dodamo poti do originalne in trenutne slike. Da lahko dve sliki primerjamo, ju moramo najprej pretvoriti v format RGB, kar storimo s pomočjo knjižnice cv2. Ti dve sliki moramo nato obrezati tako, da na njih ostaneta samo obraza. Knjižnica Openface nam že ponuja funkcionalnost iskanja največjega obraza na sliki (izsek 3.12).

Če obraza na sliki ne najdemo, takoj sprožimo napako in s tem povemo, da obraza ne moremo primerjati, kar posledično lahko zahteva tudi ponovno nalaganje drugačne slike. Slika obraza more biti razločna, za kar pa je dovolj že navadna spletna kamera. Če sistemu pošljemo sliko, na kateri je na listu papirja natisnjen obraz, ga ta v večini primerov sploh ne bo zaznal. Če nam uspe narediti zelo dobro kopijo slike in na fotografiji, zajeti s spletno kamero, zakriti dejstvo, da je na sliki papir, sistem sliko sprejme in jo začne primerjati (o točnosti več v 4. poglavju). V tem primeru se še vedno lahko zanesemo na primerjavo stila tipkanja.

Na začetku funkcije za primerjavo obrazov izvedemo rezanje in poravnavo obraza na sliki, da bo lahko primerjava obraza enostavnejša in točnejša (izsek 3.13).

Vse kar še potrebujemo je, da iz slik dobimo vektorja značilk, med katerima bomo lahko izračunali razdaljo. Da ju dobimo, pošljemo porezani sliki s poravnanima obrazoma skozi model Openface-ove nevronske mreže (izsek 3.14).

Ko imamo vektorja značilk v obliki polja za oba obraza, lahko izračunamo razdaljo med njima. Kot narekuje primer iz dokumentacije, bomo uporabili

```
1  poravnani_obraz = openface.AlignDlib(model_za_predikcijo_obraza).align(velikost_slike,
     ↪   rgb_slika, lokacija_obraza,
     ↪   landmarkIndices=openface.AlignDlib.OUTER_EYES_AND_NOSE)
```

Izsek 3.13: Ukaz, ki sliko formata RGB poreže tako, da na njej ostane samo obraz, tega pa obenem tudi poravna glede na pozicijo oči in nosu.

```
1  znacilke = openface.TorchNeuralNet(model_nevronske_mreze,
     ↪   velikosti_slik).forward(poravnani_obraz)
```

Izsek 3.14: Ukaz, ki iz slike obraza zgradi vektor značilk.

evklidsko razdaljo:

$$d_{evklidska} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2},$$

Tu je q vektor značilk obraza iz prve slike in p vektor značilk obraza iz druge. Naši vektorji značilk imajo posebne lastnosti. V članku, ki so ga o knjižnjici Openface napisali njeni avtorji [2] zasledimo, da vse točke vektorja ležijo na enotski hipersferi. V primerih iz knjižnice piše, da na koncu kot razdaljo med dvema taka vektorjema dobimo število med 0 in 4. Avtorji so s pomočjo testiranja na več tisočih slik obrazov določili tudi optimalno mejo za klasifikacijo, $0,99$. Kakovost ujemanja v odstotkih za lažjo interpretacijo ob pregledovanju rezultatov dobimo po formuli:

$$match_{\%} = \frac{4 - razdalja}{4}.$$

Če je razdalja 0, bo ujemanje stoodstotno. Sodeč po dokumentaciji, lahko vsako razdaljo, večjo od $0,99$ oziroma vsako ujemanje, manjše od $75,25\%$ uvrstimo med sumljive.

Tako kot za prejšnjo funkcionalnost lahko primerjavo obrazov dodamo na pogled, ki bo dostopen čelnemu vmesniku na določenem URL naslovu. V ta namen razvijemo pogled FaceDistanceAPIView, ki bo prejel študentovo uporabniško ime in trenutno zajeto sliko obraza, poiskal originalno sliko, ju primerjal in znal izračunati podobnost med njima ter rezultate shraniti v podatkovno bazo.

3.6 Implementacija vtičnika za Moodle

Kot smo omenili v uvodnem poglavju, bomo za beleženje zapisa stila tipkanja in slike obraza razvili vtičnik za spletne učilnice, ki tečejo na programski opremi Moodle.

3.6.1 Ideja

Vtičnik bo ponujal nov tip vprašanja, ki ga bomo lahko dodali na začetek vsakega kviza, v to vprašanje pa bo vdelal spletni obrazec, ki bo vseboval dva elementa:

- vhodno okence za prepisovanje gesla, kamor bo uporabnik moral n-krat prepisati vnaprej določen tekst in
- slikovno polje, kjer bo viden vhod iz spletne kamere, tam pa bo uporabnik slikal svoj obraz in ga s tem tudi poslal v primerjavo.

Ob konfiguraciji vprašanja bo profesor lahko določil dva tipa obrazca:

- registracijski obrazec (tipa "registration"), ki bo zajel prvotni zapis stila tipkanja in osnovno, pravo sliko obraza in
- primerjalni obrazec (tipa "test"), ki bo zajel trenutni zapis stila tipkanja in trenutno sliko obraza tistega, ki rešuje kviz, da bo lahko nato te podatke primerjal z originalnimi in zabeležil odstopanja ter jih posredoval profesorju v nadzorno ploščo.

```php
<?php
protected function definition_inner($mform) {
    $name = 'quiz_id';
    $label = "Quiz ID from the dashboard";
    $mform->addElement('text', $name, $label, array('size' => 3));
    $mform->setType($name, PARAM_INT);
}
?>
```

Izsek 3.15: Funkcija definition_inner znotraj datoteke "edit_keystrokerecorder_form.php", kjer v nastavitve vprašanja dodamo dodatno polje za konfiguracijo identifikatorja kviza.

Hkrati bo profesor moral v nastavitve vprašanja vpisati tudi identifikacijsko številko kviza, ki jo najdemo na detajlnem pogledu kviza v nadzorni plošči. Preko nje se podatki pravilno pridobivajo in pošiljajo med čelnim in zalednim delom.

3.6.2 Zaledni del vtičnika s pomočjo jezika PHP

Vtičnik za nov tip vprašanja naredimo tako, da ustvarimo novo mapo v že obstoječi mapi question/types znotraj programske opreme Moodle. Ustvarimo vse potrebne datoteke, ki so potrebne za delovanje vtičnika in so navedene tudi v dokumentaciji na Moodle-ovi spletni strani [21]. Sedaj lahko začnemo s prilagajanjem vtičnika našim potrebam.

Za svoj tip vprašanja vsak vtičnik vsebuje datoteko, znotraj katere lahko definiramo dodatna polja za konfiguracijo. Njeno ime je oblike "edit_IME VTIČNIKA_form.php". V našem primeru potrebujemo dodatno vnosno polje za shranjevanje identifikatorja kviza iz zalednega sistema. Dodamo ga v funkciji definition_inner (izsek 3.15).

Za to polje moramo definirati tudi novo tabelo v podatkovni bazi spletne učilnice, kar storimo z definicijo XML datoteke install.xml (del, ki opisuje

```
1  <TABLE NAME="qtype_keystrokerecorder" COMMENT="biometric qtype table">
2    <FIELDS>
3      <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="true"/>
4      <FIELD NAME="question_id" TYPE="int" LENGTH="10" NOTNULL="false" SEQUENCE="false" COMMENT="QuestionID"/>
5      <FIELD NAME="quiz_id" TYPE="int" LENGTH="10" NOTNULL="false" SEQUENCE="false" COMMENT="QuizID from
        ↪    dashboard."/>
6    </FIELDS>
7    <KEYS>
8      <KEY NAME="primary" TYPE="primary" FIELDS="id"/>
9    </KEYS>
10 </TABLE>
```

Izsek 3.16: Datoteka XML, ki ob inštalaciji vtičnika v bazi ustvari novo tabelo s stolpci id, question_id in quiz_id.

strukturo tabele vidimo na izseku 3.16) v mapi db našega vtičnika. Naredili bomo tabelo, ki bo vsebovala dva atributa:

- *question_id*: primarni ključ vprašanja, na katerega bomo vezali dodatne atribute in
- *quiz_id*: identifikator kviza, na katerega se bo obrazec iz vprašanja povezal preko AJAX klica. Ta identifikator bomo dobili iz polja, ki smo ga definirali v datoteki "edit_keystrokerecorder_form.php" (slika 3.15).

Dodaten atribut, ki ga shranjujemo v to tabelo, moramo tudi povezati z našim obrazcem za urejanje vprašanja. Vse kar moramo storiti je to, da ob zahtevku za shranjevanje vprašanja prestrežemo vrednost polja, ki smo ga definirali na izseku 3.15 in izvedemo SQL ukaz, ki to vrednost shrani. To storimo v datoteki "questiontype.php". Zaradi preglednosti, iz izseka kode na slikah 3.17 in 3.18 odstranimo logiko, ki upravlja z izjemami in napakami na podatkovni bazi.

Če želimo urejati že obstoječe vprašanje, moramo polja za nastavitve ob izpisu strani napolniti s podatki iz podatkovne baze. V sosednji funkciji "get_question_options" definiramo iz kje naj obrazec vzame že shranjen identifikator kviza.

```
1  <?php
2  public function save_question_options($question /* podatki o vprašanju */) {
3      global $DB;
4      $table_name = 'qtype_keystrokerecorder';
5
6      $options = (object)array(
7          'question_id' => $question->id, // identifikator Moodle vprašanja
8          'quiz_id' => $question->quiz_id // naše novo vnosno polje
9      );
10
11     if ($DB->get_field($table_name, 'id', array('question_id' => $question->id))) {
12         // če vprašanje že obstaja, mu popravimo identifikator kviza
13         $DB->update_record($table_name, $options);
14     } else {
15         // sicer vstavimo novo vrstico
16         $DB->insert_record($table_name, $options);
17     }
18
19     parent::save_question_options($question); // shranimo ostale nastavitve, ki niso
        ↪   del našega vtičnika
20     return true;
21 }
22 ?>
```

Izsek 3.17: Funkcija save_question_options znotraj datoteke "questiontype.php", kjer prestrežemo vrednost vnosnega polja za identifikator kviza in to shranimo v podatkovno bazo.

```php
<?php
public function get_question_options($question /* podatki o vprašanju */) {
    global $DB;

    $question->options = $DB->get_record('qtype_keystrokerecorder',
    ↪   array('question_id' => $question->id)); // iz baze dobimo podatek o trenutnem
    ↪   identifikatorju kviza

    parent::get_question_options($question); // napolnimo še ostale nastavitve, ki niso
    ↪   del našega vtičnika

    return true;
}
?>
```

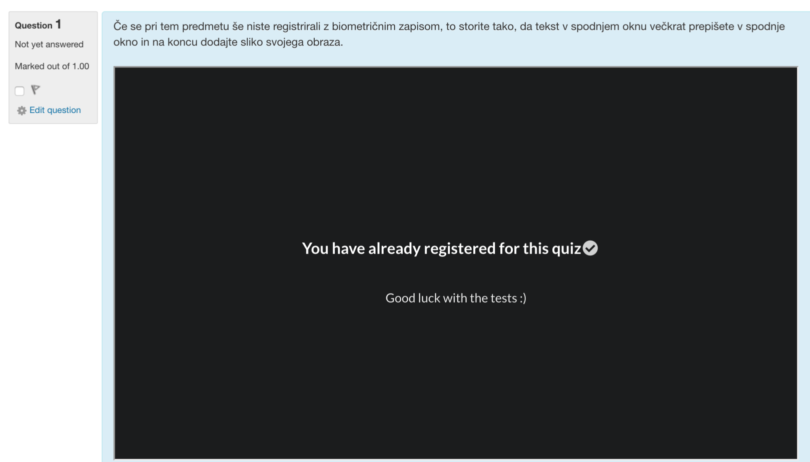
Izsek 3.18: Funkcija get_question_options, kjer iz podatkovne baze dobimo že nastavljene vrednosti in z njimi ob urejanju nastavitev vprašanja prednapolnimo vnosna polja.

Za izbor ustreznega tipa obrazca, ki ga želimo prikazati, preberemo vrednost iz naslova vprašanja. Če je v naslovu beseda "registration", bomo prikazali registracijski obrazec, testni pa tedaj, ko bo tam beseda "test". Za shranjevanje naslova vprašanja poskrbi že spletna učilnica. Sedaj, ko poznamo oba potrebna atributa za prikaz obrazca, lahko v datoteki renderer.php, ki skrbi za izpis vprašanja, določimo izpis obrazca.

3.6.3 Čelni del s pomočjo jezikov HTML, CSS in JavaScript

Kot smo omenili v podpoglavju 3.6.2, moramo ob izpisovanju znotraj kviza v vsako naše vprašanje vdelati primeren obrazec.

Zato potrebujemo čelno HTML strukturo obeh obrazcev, CSS stile in JavaScript funkcije za vsak tip obrazca posebej. V datoteki renderer.php določimo strukturo obrazcev in na posamezne elemente obesimo CSS razrede iz knjižnice Semantic UI, ki ponuja vnaprej narejene stile za HTML



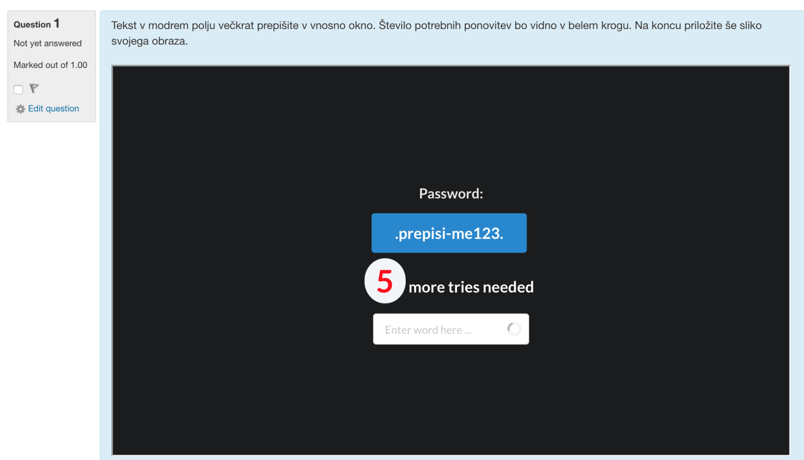
Slika 3.9: Obrazec v stanju, ko je uporabnik pri predmetu že registriran.

elemente. Poleg tega potrebujemo še nekaj svojih dodatnih CSS stilov za dodatno preoblikovanje obrazca in pa JavaScript datoteko, kjer implementiramo vso logiko, ki bo znala meriti čas pritiska tipk in izmenjevati vse potrebne podatke preko AJAX zahtevkov z našim zalednim sistemom. Izgledi teh obrazcev v različnih stanjih so vidni na slikah 3.9, 3.10, 3.11 in 3.12.

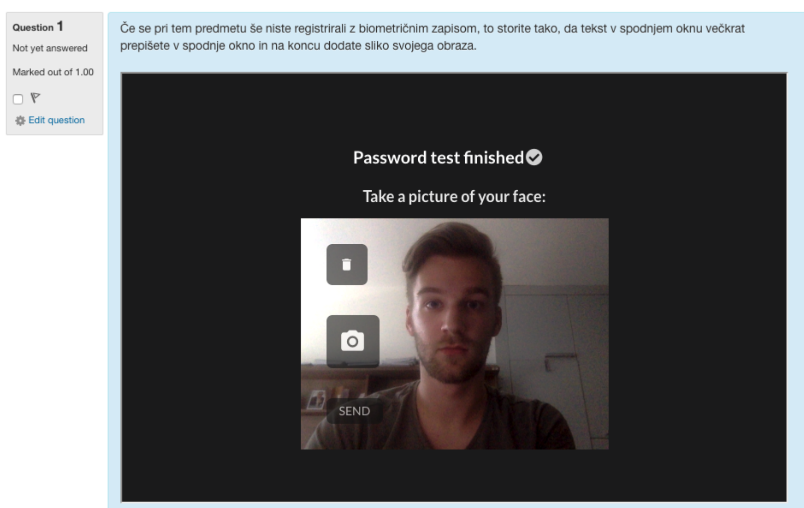
V implementaciji zapisovanja časa trajanja pritiska tipk moramo tudi paziti na morebitne izjeme, ki bi pokvarile naše podatke. Potrebno bo ugotoviti, če sta bili morebiti dve zaporedni tipki pritisnjeni v premajhnem časovnem razmaku, kar bi pomenilo, da jih naša skripta ni utegnila pravilno zapisati. Lahko se tudi zgodi, da uporabnik po nesreči pritisne na napačno tipko. V takih primerih ga bomo prosili, da v vnosno okno znova prepiše celoten tekst.

V te obrazce moramo preko parametrov GET zahtevka ob testu poslati identifikator kviza ter identifikator Moodle računa, ki je trenutno prijavljen v spletno učilnico. Ob prvotni registraciji v katerikoli predmet, se ta identifikator poveže z novo instanco modela Student na našem zalednem sistemu.

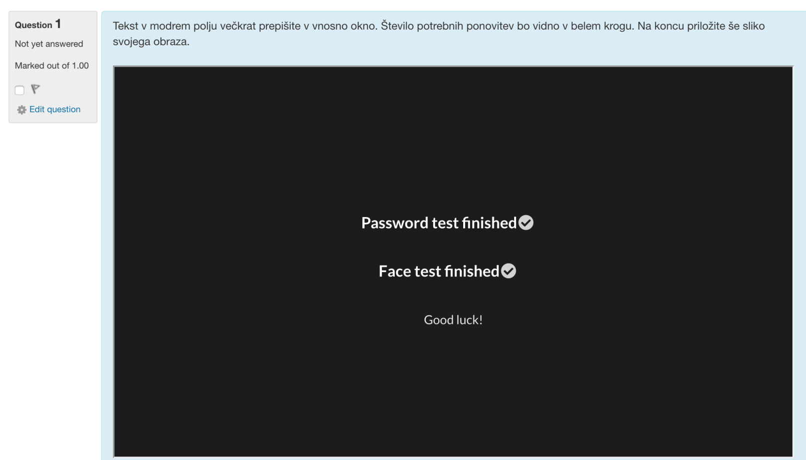
Če je trenutni uporabnik že prijavljen v našem sistemu pri katerem drugem predmetu, ob prikazu obrazca tipa "registration"od njega ne zahtevamo več slike obraza, saj jo je tam že posredoval. Prav tako pri obrazcu tipa



Slika 3.10: Obrazec za prepisovanje teksta v vnosno polje (uporabljen tako pri registraciji kot pri testu).



Slika 3.11: Obrazec za nalaganje slike obraza, ki se pojavi ob prvi registraciji v sistem in pri vsakem testu.



Slika 3.12: Obrazec v stanju, ko uporabnik zaključi z registracijo oziroma testom.

"registration"od študenta ne zahtevamo več zapisa stila tipkanja, če se je ta že registriral pri trenutnem predmetu. V ta namen našemu obrazcu dodamo JavaScript kodo, ki se izvede ob prikazu obrazca. Ta naredi AJAX zahtevek na zaledni sistem, ki pridobi vse potrebne informacije o tipu testa stila tipkanja in podatkih, ki jih mora študent še posredovati našemu sistemu. Podatke o tipu testa dobi s klicem na pogled QuizInfoAPIView (iz poglavja 3.4.1, informacije o potrebnih podatkih pa s klicem na pogled CourseStudentStatusView (iz poglavja 3.3.3). Ta zahtevek naredimo s pomočjo knjižnice axios.

Sedaj znamo nastaviti vse potrebne elemente obrazca za izvedbo kviza in poskrbeti, da se že registrirani uporabniki ne prijavljajo ponovno v predmet. Preostane nam, da napišemo funkcije, ki beležijo čase med zaporednimi pritisnjenimi tipkami, te sestavijo v vektorje in pošljejo na našo centralno točko API.

Te naredimo s pomočjo jezika JavaScript, ki nam ponuja funkcije, ki na poljubnih HTML elementih spremljajo spremembe. Tako lahko prestrežemo vsako uporabnikovo interakcijo z našim obrazcem (primer definicije funkcije, ki posluša pritiske in spuste tipk vidimo na izseku 3.20). Slediti bomo morali

```
axios.get('http://localhost:8000/quiz/' + identifikatorKviza)
    .then((podatki_o_kvizu) => {

        // na tem mestu nastavimo tekst in število ponovitev

        axios.get('http://localhost:8000/course/' + identifikatorPredmeta +
         ↪   '/student_status/' + trenutniUporabnik)
            .then((podatki_o_studentu) => {

                const studentHasRecord = podatki_o_studentu.data.has_record
                const studentHasImage = podatki_o_studentu.data.has_picture

                if (studentHasRecord) {
                    // na tem mestu skrijemo vse elemente
                } else if (studentHasImage) {
                    // na tem mestu skrijemo vhodni element za sliko obraza
                }
            }
        })
    })
```

Izsek 3.19: AJAX zahtevek na naš zaledni sistem ob prikazu registracijskega obrazca, ki v spremenljivki "podatki_o_kvizu" vrne odgovor naše centralne točke API v obliki objekta JSON. Poleg informacij o kvizu preveri tudi, če je morda trenutni uporabnik že registriran v trenutni predmet in posledično skrije vse vhodne elemente.

```
1  const listenToKeys = () => {
2      const wordInput = document.getElementById('wordInput') // referenca na vnosno okno
3      wordInput.onkeydown = wordInput.onkeyup = measureTimes // ob vsakem pritisku in
    ↪   spustu tipk znotraj vnosnega okna naj se izvede funkcija measureTimes
4  }
5
6  const measureTimes = (e) => {
7      // funkcija, ki ob vsakem pritisku ali spustu dobi informacije o trenutni tipki
8  }
```

Izsek 3.20: JavaScript funkcionalnost za določitev funkcije, ki se izvede ob pritisku in spustu poljubne tipke.

vsaki vpisani črki in preveriti, če je enaka kot trenutna črka iz besede, ki jo študent prepisuje. V hipu, ko se ta zmoti, mora prepisovanje besede začeti znova. Za vse tri tipe časov, ki jih moramo posneti za oba obrazca, implementiramo merilce. Čas med pritiskom in spustom ene tipke zajamemo tako, da si ob pritisku in spustu zapomnimo trenutne čase in med njimi izračunamo razliko ter jo dodamo v naš vektor (za potrebe razlage poenostavljeno logiko vsebuje izsek 3.21).

Na podoben način razliko izračunamo med pritiski in spusti (oziroma zaporednimi pritiski) dveh zaporednih tipk. Ravno tako si lahko ob pritisku in spustu zapomnimo trenutne čase in med njimi izračunamo razlike (poenostavljena logika za računanje preostalih dveh časov je vidna na izsekih 3.22 in 3.23).

Za vsako ponovitev prepisovanja tako sestavimo nov vektor, te pa združimo v matriko, ki jo s pomočjo zahtevka POST pošljemo na točko API in shranimo v našo podatkovno bazo. Na tem mestu imamo delujoča obrazca za registracijo in testiranje na začetku preverjanja znanja, ki znata zajemati vse potrebne podatke in jih s pomočjo REST protokola shraniti v naš zaledni sistem. S tem je implementacija našega sistema za registracijo in primerjavo biometričnih lastnosti študentov zaključena in jo lahko preizkusimo.

```
1   // v objektu e, ki se posreduje naši funkciji najdemo podatek o tipu dogodka (pritisk
    ↪   oziroma spust tipke)
2
3   // če pritisnemo na tipko, poiščemo njeno ASCII kodo in si zapomnimo trenutni čas
4   if (e.type === 'keydown') {
5       keyTimes[ascii_koda].lastDown = Date.now()
6   }
7
8   // če tipko spustimo, poiščemo čas pritiska, izračunamo razliko med njim in trenutnim
    ↪   časom ter jo shranimo v vektor, ki hrani dolžine držanja tipk
9
10  if (e.type === 'keyup') {
11      keyTimes[ascii_koda].lastUp = Date.now()
12
13      const duration = keyTimes[ascii_koda].lastUp - keyTimes[ascii_koda].lastDown
14      hDurations.push({key: ascii_koda, duration})
15  }
```

Izsek 3.21: JavaScript funkcija, ki beleži dolžine pritiskanja posameznih tipk.

```
1  if (previousDownKey) { // pogoj za računanje je, da je bila prej neka tipka že
    ↪   pritisnjena
2      casZadnjegaPritiska = previousDownKey.timestamp
3
4      // vektorju razlik dodamo nov vnos
5      ddDurations.push(
6        {
7          key1: previousDownKey.key, // ascii koda tipke, ki jo je uporabnik nazadnje
            ↪   pritisnil
8          key2: currentDownKey.key, // ascii koda tipke, ki jo je uporabnik sedaj
            ↪   pritisnil
9          duration: Date.now() - casZadnjegaPritiska
10       }
11     )
12 }
13
14 previousDownKey = currentDownKey
```

Izsek 3.22: JavaScript funkcija, ki beleži časovno razliko med pritiskom prejšnje in trenutne tipke.

```
1  if (currentUpKey) { // pogoj za računanje je, da je bila prej neka tipka že spuščena
2      casZadnjegaSpusta = currentUpKey.timestamp
3
4      // vektorju razlik dodamo nov vnos
5      udDurations.push(
6        {
7          key1: currentUpKey.key, // ascii koda tipke, ki jo je uporabnik nazadnje
            ↪   spustil
8          key2: currentDownKey.key, // ascii koda tipke, ki jo je uporabnik sedaj
            ↪   pritisnil
9          duration: Date.now() - casZadnjegaSpusta
10       }
11     )
12 }
```

Izsek 3.23: JavaScript funkcija, ki beleži časovno razliko med spustom prejšnje in pritiskom trenutne tipke.

Poglavje 4

Testiranje sistema

V tem poglavju bomo na prostovoljcih preizkusili naši dve tehniki biometrične avtentikacije. Svojo implementacijo primerjave stila tipkanja bomo najprej testirali do te mere, da določimo potrebno mejo za klasifikacijo. Po tem popravku bomo celoten sistem še enkrat testirali in zapisali ugotovitve.

Preden na splošno preverimo kvaliteto delovanja našega sistema, moramo najprej nastaviti njegove parametre. Ko bomo na prostovoljcih testirali delovanje primerjanja zapisa stila tipkanja, bomo morali definirati tekst in število ponovitev prepisovanja v vnosno okno. Najprej ugotavljamo, če se bolj splača uporabiti dolgo besedilo, ki ga prepišemo manjkrat ali kratko besedilo, ki ga prepišemo večkrat. Naše testiranje bomo izvedli na računalniku Apple MacBook Pro z naslednjimi specifikacijami:

- Procesor: 2,7 GHz Intel Core i5,
- RAM: 8GB 1867 Mhz DDR3,
- Grafična kartica: Intel Iris Graphics 6100 s 1536MB grafičnega pomnilnika,
- Ekran: 13 palčni Retina ekran z ločljivostjo 2560x1600 pikslov.

Na ekranu bomo vedno imeli odprte zgolj obrazce, ki so končni izdelek iz prejšnjega poglavja o implementaciji.

Rezultati ujemanja stila za dolgi tekst				
Lastnik / Uporabnik	Jaka	Iztok	Špela	Nuša
Jaka	79.69%	61.67%	60.75%	64.89%
Iztok	60.21%	80.04%	60.27%	75.72%
Špela	62.16%	63.60%	76.26%	62.18%
Nuša	56.51%	69.17%	63.10%	70.01%

Tabela 4.1: Rezultati ujemanja stilov tipkanja za eno ponovitev daljšega teksta.

4.1 Testiranje implementacije primerjave zapisa tipkanja

Naredili bomo kratek enostaven test, kjer se bo vsak od prostovoljcev najprej registriral za oba tipa testa, z različno dolgimi teksti in potem poskusil s primerjavo stila tipkanja. Primerjal se bo s samim seboj in z vsemi drugimi sodelujočimi v testu.

Za kratko besedilo si za potrebe testiranja izberemo frazo ":ok_hand:", saj je ta dokaj neobičajna in nepoznana ljudem, na katerih naš sistem testiramo in tudi, ker je njena sedemkratna ponovitev ravno toliko dolga kot naš dolg tekst. Dolgo besedilo, ki ga uporabimo, je "prijava le z uporabniskim imenom in geslom ni dovolj zanesljiva". Sedemkratna ponovitev kratkega teksta je tako kot enkratna ponovitev daljšega dolga 63 znakov.

4.1.1 Testiranje na prostovoljcih za ugotovitev potrebnih parametrov

Med izvedbo testov na vsakem prostovoljcu si rezultate skrbno zapisujemo v tabelo. Za oba tipa testa so le-ti na voljo v tabelah 4.1 in 4.2.

Ko imamo na voljo rezultate obeh tipov testov, lahko ugotovimo kako dober je naš primerjalnik s pomočjo krivulj ROC in mere AUC, ki predstavlja ploščino pod njimi. Krivulje ROC predstavljajo povezane točke na enotskem

Rezultati ujemanja stila za kratek tekst				
Lastnik / Uporabnik	Jaka	Iztok	Špela	Nuša
Jaka	92.44%	61.00%	55.03%	63.81%
Iztok	51.55%	88.51%	67.58%	71.80%
Špela	55.27%	78.65%	80.05%	78.55%
Nuša	52.71%	75.47%	64.61%	81.68%

Tabela 4.2: Rezultati ujemanja stilov tipkanja za sedem ponovitev kratkega teksta.

grafu v dvodimenzionalnem koordinatnem sistemu. Te točke dobijo svoje koordinate glede na delež resničnih pozitivnih primerov (ang. true positive rate oziroma TPR) in delež lažnih pozitivnih primerov (ang. false positive rate ali FPR) pri neki določeni meji za določanje razreda pri binarnih klasifikatorjih (slovenska imena za deleže smo si sposodili iz diplomskega dela Mihe Bička [4]). V našem primeru želimo sestaviti klasifikator, ki bo na podlagi rezultata ujemanja stila tipkanja v odstotkih določil ali gre za ponarejanje identitete oziroma ali kviz rešuje prava oseba. Delež resničnih pozitivnih primerov nam pove, koliko rezultatov, ki smo jih uvrstili v nek razred, je dejansko tudi v tem razredu. Delež lažnih pozitivnih primerov pa nam pove, koliko rezultatov, ki smo jih uvrstili v razred, ne bi smelo biti v tem razredu. Mejo bomo za naš test premikali od najnižjega ujemanja v odstotkih do najvišjega (uporabili bomo sortirane rezultate od najmanjšega do največjega) in poskusili ugotoviti, kje je ta meja najboljša za klasifikacijo. Poleg tega bomo pri vsakem tipu testa pogledali deleža resničnih in lažnih pozitivnih primerov ter izrisali krivuljo ROC in preverili ploščino pod njo. Ploščina pod krivuljo ROC oziroma AUC nam pove kako dober je v splošnem nek klasifikator. Če nek klasifikator, kljub temu, da pri neki meji ni tako natančen kot pri svoji najbolj optimalni meji, še vedno bolje klasificira stvari v prave razrede kot drugi klasifikator, bo ta imel delež resničnih pozitivnih primerov v neki točki višji. Če se to pojavi večkrat, bo po vsej verjetnosti zato imel tudi večji AUC.

Seveda ne smemo zanemariti dejstva, da je bil naš vzorec prostovoljcev majhen in bi morda ob večjem obsegu podatkov dobili drugačen rezultat. Že iz podatkov vidimo, da je meja zelo tanka, saj pri primerjavi Špelinega testnega stila z njenim lastnim registracijskim dobimo ujemanje $80,05\%$, pri primerjavi Iztokovega testnega stila s Špelinim registracijskim pa $78,65\%$. Po našem kratkem testu s 16 kombinacijami poskusov avtentikacije se po pridobljenih podatkih nekako izkaže, da je bolj natančen test s kratkim besedilom, ki ima AUC enak 1, kar pomeni, da ob optimalno nastavljeni meji naš klasifikator vedno pravilno napove, ali je oseba res tista, ki bi morala biti. Pri dolgem besedilu pa je AUC enak 0.89583, saj se naš klasifikator tudi ob najbolje nastavljeni meji vsaj enkrat zmoti. V upoštev pri nastavitvah bodo prišle še nekatere druge stvari. Skozi potek implementacije smo ugotovili, da za primerjavo dveh stilov tipkanja potrebujemo vektorje časov enake oblike. Če testiramo ljudi, ki res hitro tipkajo, se lahko zgodi, da dve tipki pritisnejo zaporedoma zelo hitro oziroma praktično hkrati. JavaScript ponavadi tega ne more tako hitro zabeležiti, kar pomeni, da mora uporabnik besedo prepisati še enkrat od začetka, da bi dobili vektor časov primerne oblike. Če za avtentikacijo uporabimo daljši tekst, se lahko proti koncu prepisovanja tega teksta zgodi, da nekdo dve tipki pritisne hkrati in mora zato ponoviti pisanje celotnega besedila, kar je izjemno utrujajoče in frustrirajoče za študente, ki so že tako živčni. Seveda se ti lahko tudi zmotijo, kar tudi pomeni, da morajo ponovno začeti s prepisovanjem. Vse to so dodatni faktorji, zaradi katerih se v nadaljnjem testiranju raje poslužimo metode krajšega teksta z večjim številom ponovitev.

4.1.2 Izboljšave in predelave

Najbolj optimalna meja za določanje razreda pri binarnem klasifikatorju je tam, kjer je delež resničnih pozitivnih primerov največji oziroma tam, kjer največ primerov pravilno uvrstimo v njihove razrede. Pri našem sistemu zato mejo ujemanja postavimo na 78.65%, ker pri naši krivulji ROC ta opiše točko, ki ima delež resničnih pozitivnih primerov enak 1 in delež lažnih pozitivnih

primerov enak 0.

4.2 Testiranje prototipa sistema

V tem podpoglavju ne bomo smeli ponovno testirati ljudi, ki smo jih testirali za namene prirejanja parametrov našega sistema, saj tako ne bi dobili smiselnih rezultatov. Če bi sistem testirali na parametrih, ki so najboljši za ljudi, na katerih smo jih nastavili, bi dobili zelo dobre rezultate. To pa ne pomeni, da sistem dobro deluje tudi v splošnem.

Za primerjavo slik obraza s pomočjo knjižnice Openface so njeni avtorji optimalno klasifikacijsko mejo po testih na več tisoč obrazih postavili na 0.99. Če bi sami hoteli določiti mejo, verjetno ne bi dobili tako natančne številke, kot so jo dobili avtorji. Na voljo nimamo toliko prostovoljcev, iskati mejo na slikah obrazov iz prosto dostopnih podatkovnih vzorcev iz interneta pa ne bi bilo smiselno, ker je bila meja 0.99 določena ravno na tak način. Zato bomo že vnaprej določeno mejo tudi sami uporabili kot mejo za klasifikacijo študentov v našem sistemu.

V poglavju o implementaciji smo omenili, da sistem v idealnih pogojih sprejme tudi sliko obraza, ki jo natisnemo na list papirja. V tem primeru točnost ni enaka tisti, ki jo pričakujemo. Po rezultatih testiranja sodeč, bi v vsakem primeru pričakovali točnost nekje nad 91%. Izvedemo test, ki primerja registracijsko sliko študenta z enako sliko, natisnjeno na listu papirja. Pri tej primerjavi je ujemanje 88.77%, kar je nižje od povprečnega pravega ujemanja iz rezultatov testa na prostovoljcih. V tem primeru se še vedno lahko zanesemo na primerjavo obrazov. Če se kviz izvaja v predavalnici, kjer profesor še vedno nadzoruje študente, da ti ne prepisujejo, je skoraj vedno moč opaziti, da nekdo pred ekranom drži list papirja. Seveda pa to predstavlja veliko težavo pri reševanju kvizov od doma, kjer tega nadzora ni. Točnost bi morda lahko izboljšala tudi sprememba dovoljene meje razlikovanja, a te na tako majhnem vzorcu rezultatov ne moremo tako trdno določiti.

Za novo testiranje uporabimo kombinacije (tokrat žal ne vseh možnih)

Rezultati ujemanja stila tipkanja (meja: 78.65%)						
L / U	Matija	Didka	Sandi	Matevž	Sašo	David
Matija	85.13% P	68.83% S	76.57% S	67.13% S	68.37% S	68.80% S
Didka	65.23% S	83.66% P	66.19% S	67.22% S	68.16% S	72.31% S
Sandi	70.20% S	71.20% S	83.03% P	68.49% S		64.11% S
Matevž	69.83% S	76.72% S	72.45% S	87.76% P		75.17% S
Sašo	57.91% S	72.49% S	68.28% S		87.11% P	70.27% S
David	59.79% S	80.08% P				82.90% P

Tabela 4.3: Rezultati ujemanja stilov tipkanja za 7 ponovitev kratkega teksta na končnem prototipu sistema.

Rezultati ujemanja slike obraza (meja: 75.25%)						
L / U	Matija	Didka	Sandi	Matevž	Sašo	David
Matija	97.83% P	56.06% S	71.97% S	64.33% S	67.81% S	74.49% S
Didka	49.47% S	97.59% P	54.03% S	29.99% S	38.35% S	57.80% S
Sandi	78.39% P	65.98% S	98.99% P	76.38% P		56.98% S
Matevž	70.65% S	37.55% S	68.41% S	91.30% P		39.91% S
Sašo	62.50% S	40.26% S	56.73% S		97.06% P	42.38% S
David	64.68% S	70.60% S				92.34% P

Tabela 4.4: Rezultati ujemanja obrazov iz fotografij na končnem prototipu sistema.

prijav šestih različnih ljudi, ki jih nismo uporabili za nastavljanje parametrov. Rezultati primerjav in klasifikacije s pomočjo izračunanih parametrov so vidni v tabelah 4.3 in 4.4. Če smo študenta klasificirali kot pravega lastnika računa (odstotek ujemanja nad 78.65%), mu pripišemo oznako (P), če pa smo ga zaznali kot sumljivega (odstotek ujemanja pod ali enak 78.65%), pa mu dodamo oznako (S). Za test smo uporabili enak kratek tekst in število njegovih ponovitev kot v testu za nastavitev parametrov. V zgornjem levem polju tabele z rezultati uporabimo oznako L, ki označuje lastnika računa in U, ki označuje trenutnega uporabnika, ki se poskuša prijaviti.

Takoj opazimo, da je naš klasifikator, ki primerja stile tipkanja, pravilno

ocenil identitete študentov skoraj v vsakem primeru. Prevare ni zaznal le v primeru, ko je Didka reševala kviz pod pretvezo, da je David. To je verjetno posledica tudi relativno majhnega števila podatkov in dejstva, da med testi ni minilo toliko časa, kot ga ponavadi mine med dvema spletnima preverjanjema (skoraj 2 meseca). Po vsej verjetnosti bi se stil tipkanja neke osebe vsaj minimalno spremenil čez čas, kar bi privedlo do malo tesnejših rezultatov, ki bi bili za napoved manj ugodni. Sistem bi lahko na tem mestu tudi nadgradili tako, da bi si ob vsakem uspešno prestanem testu zapomnil trenutni stil tipkanja in ga dodal k prvotnemu ter ju povprečil. Več o nadgradnjah v poglavju 5. Kljub temu opazimo, da koncept primerjave stila tipkanja uspešno deluje, saj se je pri 30 poskusih avtentikacije naš sistem zmotil samo enkrat.

Zgodba pa se seveda tukaj še ne zaključi, saj v sistemu uporabljamo še eno tehniko biometrične verifikacije, primerjavo obrazov. Ta del našega sistema je rezultate napačno napovedal pri dveh primerih. Matija in Matevž sta uspešno prestala test primerjave obraza, čeprav je bil prijavljen uporabnik Sandi. Vidimo tudi, da bi na naših podatkih morda bila bolj optimalna kakšna višja meja, a tega z gotovostjo ne moremo trditi, saj je bil vzorec avtentikacijskih poskusov relativno majhen v primerjavi s celotno populacijo študentov, ki bi ta sistem lahko uporabljali. Navkljub napakam primerjave obraza, bi pri našem testu, sumljive poskuse avtentikacije prestregla primerjava stila tipkanja in obratno.

Za končni klasifikator uporabimo tako rezultat primerjave tipkanja, kot rezultat primerjave obraza. Test študent uspešno prestane le takrat, ko sta oba odstotka ujemanja nad mejo, ki smo jo določili. V tem primeru bi naš sistem pri vseh testih prostovoljcev, ki jih ni reševal lastnik računa, zaznal sumljiva dejanja.

4.3 Analiza SWOT

Analiza SWOT je tehnika, ki nam pomaga poiskati in bolje razumeti prednosti, slabosti, priložnosti in nevarnosti nekega sistema. Tehnika je najbolj priljubljena pri strateškem načrtovanju v podjetjih, mi pa jo bomo uporabili za evalvacijo našega sistema za biometrično primerjavo identitet pri spletnih preverjanjih znanja.

Matriko SWOT strukturiramo na isti način kot David W. Pickton in Sheila Wright v svojem članku, ki opisuje to tehniko analize [24]. Prednosti in slabosti se nahajajo v levem stolpcu matrike SWOT, kjer je prostor za notranje dejavnike. Ti so taki, da lahko nanje neposredno vplivamo ter jih prilagodimo potrebam. Zunanji dejavniki so takšni, da nanje ne moremo neposredno vplivati, a se nanje lahko prilagodimo s spreminjanjem notranjih lastnosti. Priložnosti in nevarnosti so torej v desnem stolpcu matrike, saj sodijo med zunanje dejavnike.

4.3.1 Gradnja matrike SWOT

Iz prej dobljenih rezultatov in opazk izluščimo omenjene dejavnike ter z njimi napolnimo tabelo (slika 4.5).

4.3.2 Interpretacija matrike

Iz matrike lahko razberemo, da ima naš sistem velik potencial za uporabo v prihodnosti, saj vpeljuje učinkovit pristop za preverjanje identitete, bodisi samostojno, bodisi kot dodatek. Poleg uporabe v študijske namene, bi bilo mogoče prilagajanje za različne tipe spletnih učilnic, saj je zaledni sistem zgrajen popolnoma neodvisno od vtičnika za spletno učilnico Moodle. To lahko komu, ki si morda želeli dostopati do vseh informacij na eni skupni točki, spletni učilnici, predstavlja bolj slabost kot prednost, vendar tudi na tem področju je možno dograditi dodatne funkcionalnosti, ki bi vgradile nadzorno ploščo neposredno v spletno učilnico. Najverjetnejši razlog za neuporabo sistema bi bil morebitni nastanek podobnega sistema, ki bi ga razvilo

	Notranji dejavniki	**Zunanji dejavniki**
+	*Prednosti*: • Hitrost preverjanja identitete • Dobra preglednost rezultatov • Izmenjava biometričnih podatkov med različnimi predmeti • Enostavno dodajanje dodatnih biometričnih tehnik primerjave	*Priložnosti*: • Možnost uporabe na fakulteti • Možnost uporabe na drugih fakultetah • Možnost predelave za neakademske namene
-	*Slabosti*: • Ne izboljša varnosti reševanja od doma • Zaenkrat na voljo zgolj kot vtičnik za Moodle • Zaledni del ni vgrajen v Moodle	*Nevarnosti*: • Nove zakonske omejitve glede pridobivanja osebnih podatkov (npr. GDPR) • Nedelujoča strojna oprema • Nezaupanje izvajalcev kvizov • Investicije renomiranih podjetij v vtičnik, ki opravlja iste funkcije

Tabela 4.5: Matrika SWOT za naš prototip sistema.

kakšno od bolj renomiranih podjetij, ki mu ljudje v splošnem bolj zaupajo. Po vsej verjetnosti bi se izvajalci kvizov raje odločili za sistem, ki ga je razvil Microsoft kot pa sistem, ki ga je izdelal študent, pa četudi z njim ni nič narobe. S tem ko smo zgradili ločena zaledni in čelni del sistema, smo si odprli možnost predelav in nadgradenj za različne (tudi neakademske) sisteme, kjer bi lahko za preverjanje identitete uporabljali naše pristope. Možnosti za nadgradnjo je še ogromno, več o teh pa v zaključnem poglavju.

Poglavje 5

Zaključek

5.1 Možne izboljšave in nadgradnje

Pri registraciji v naš sistem smo študente prosili, da preko obrazca tipa "registration" naložijo sliko svojega obraza. Namesto takega pristopa bi lahko slike študentov vzeli iz portala portala ID.uni-lj.si. Paziti bi morali, da slika študenta tam res obstaja, v nasprotnem primeru bi študenta prosili za sliko na tak način, kot smo ga že implementirali v sistemu. Sistem bi lahko ob primerjavah shranil vsako sliko obraza na datotečni sistem in tako ponudil profesorju možnost podrobnejšega vpogleda v podatke, uporabljene pri primerjavi. Lahko bi tudi zaznali, da se slika pojavi že drugič, kar bi skoraj gotovo pomenilo, da nekdo ne prilaga trenutne slike obraza, temveč že prej zajeto fotografijo.

V okviru diplomskega dela smo implementirali sistem primerjave stila tipkanja, ki s pomočjo enostavnih mer profesorju prikazuje odstopanja originalnih in trenutnih biometričnih zapisov. Obstajajo še druge, bolj kompleksne tehnike za računanje razdalj med vektorji in nekatere, za potrebe primerjave stila tipkanja, delujejo bolje kot druge. Prej omenjena raziskava Killourhy-ja in Maxion-a iz univerze Carnegie Mellon [17] o natančnosti razdalj je pokazala, da sta najučinkovitejši dve, Manhattanska in razdalja Mahalanobisa (v kombinaciji z algoritmom iskanja najbližjega soseda). Ti dve razdalji bi lahko

implementirali v naš sistem in s tem izboljšali njegovo natančnost. Lahko bi tudi profesorju ponudili meni, kjer bi si želeno razdaljo izbral. Poleg razdalj bi lahko uvedli tudi nekatere dodatne časovne meritve. V članku na temo avtentikacije s pomočjo stila tipkanja [5] sta Patrick Bours in Soumik Mondal v svoj sistem dodala še čas, ki poteče med spustoma dveh zaporednih tipk.

Pri tipih testa (KeystrokeTestType) za testiranje stila tipkanja smo se omejili na eno besedilo in eno točno določeno število ponovitev njegovega prepisovanja v vnosno okno. Kot nadgradnjo bi lahko v prihodnje dodali tudi možnost povezave več tipov na en predmet, ob vsakem izvedenem kvizu pa bi se lahko profesor odločil za naključnega. S tem bi povečali varnost in zanesljivost sistema, saj bi študentom preprečili možnost učenja stila tipkanja za drugo osebo, za zgolj en tip testa, kar je relativno enostavno izvedljivo. Seveda bi uvedba več tipov testa s seboj prinesla tudi bolj mučno in daljšo registracijo. Ob uvedbi več tipov testa, bi moral vsak študent ob registraciji v predmet, za vse teste registrirati svoje stile tipkanja, za kar bi potreboval nekaj več časa. Vseeno bi dodatna zanesljivost odtehtala slabšo uporabniško izkušnjo.

Nadgradnja bi lahko bila tudi shranjevanje vseh zapisov tipkanja, ki uspešno prestanejo preizkus istovetnosti za uporabnika in bi tudi te upoštevali pri računanju novih razdalj pri prihodnjih preverjanjih znanja. Lahko bi skozi kvize popravljali oziroma povprečili pravi zapis stila tipkanja uporabnika in tako še dodatno izboljšali naše originalne podatke. Namesto osnovne, prve matrike bi sestavili matriko, ki bi vsebovala povprečne čase med pritiski iz vseh preteklih testov, ki so bili uspešno prestani. Ob uspešno prestanem testu, bi vzeli originalen in trenutni vektor, ju sešteli in delili z 2. Pri vnovičnem testiranju bosta obe matriki še vedno iste oblike, zato bomo s pomočjo enake funkcije iz aplikacije "keystroke" še vedno znali izračunati razdaljo med dvema stiloma tipkanja.

Ne smemo pozabiti, da sta v našem sistemu implementirana biometrična pristopa zgolj dva izmed mnogih, ki jih lahko uporabljamo za identifikacijo ljudi. Trenutno najbolj razširjena in zanesljiva biometrična metoda je primer-

java prstnih odtisov, ki je kot primarni način avtentikacije dandanes prisotna že skoraj na vsakem pametnem mobilnem telefonu. Na žalost pa bralniki prstnih odtisov niso tako pogosti na osebnih in prenosnih računalnikih. Poleg avtentikacije s prstnim odtisom obstajajo še številne druge metode, kot so primerjava oblike dlani, ušes, podpisa, glasu in zenice.

Veliko možnosti za izboljšavo sistema je tudi na področju poenostavitve uporabniške izkušnje. S pomočjo animacij, ki se izvajajo med procesiranjem, nalaganjem in izmenjavo podatkov, bi lahko študente lažje usmerjali skozi testni obrazec.

5.2 Možnost uporabe sistema v prihodnosti

Kot smo omenili v uvodnem poglavju, bi naš sistem omogočil premestitev reševanja spletnih kvizov iz učilnic v predavalnico. Tako bi namesto več terminov, vse študente umestili v enega samega in zmanjšali čas, porabljen za administrativni del spletnih preverjanj znanj.

5.3 Sklepne ugotovitve

Razvili smo sistem za prijavo s pomočjo biometričnih značilnosti študentov pri spletnih preverjanjih znanja. Celotna aplikacija je na voljo v dveh ločenih repozitorijih na spletni strani Github.com. Prvi repozitorij je na voljo pod imenom quiz-biometrics-api [14] in vsebuje vso zaledno logiko in točko API našega sistema. Drugi repozitorij z imenom quiz-biometrics-plugin [15] pa vsebuje čelni del našega sistema oziroma vtičnik za Moodle. Vsakemu repozitoriju so priložena tudi navodila za namestitev v lokalnem okolju. Sistem na prostovoljcih deluje, kar pomeni, da po vsej verjetnosti deluje tudi na večjem številu ljudi. Četudi obstaja možnost, da bi se pri večjem vzorcu ljudi sistem nekajkrat zmotil, to še ne pomeni, da ne bi mogel prispevati k lažjemu preverjanju identitet študentov na preverjanjih znanja. Sam sistem vidim kot pripomoček, ki bi lahko pomagal profesorju pri identifikaciji

študentov, ne bi bil pa obvezujoč. Če sistem označi poskus prijave na preverjanje znanja kot sumljiv, lahko profesor osebo še vedno fizično preveri, kar pa v splošnem vzame manj časa, kot če bi moral preveriti vsakega študenta posebej. Obenem je sistem zasnovan tako, da omogoča še ogromno nadgradenj, ki bi pripomogle k robustnosti in zanesljivosti. Z vsemi izboljšavami bi bil sistem še bolj zanesljiv in bolj praktičen za uporabo, čeprav se je že pri testiranju prostovoljcev pokazal kot precej točen.

Literatura

[1] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[2] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, 2016.

[3] Axios: Promise based http client for the browser and node.js. Dosegljivo: `https://github.com/axios/axios`. [Dostopano: 23. 5. 2018].

[4] Miha Biček. Grafični gradnik za merjenje kvalitete klasifikatorja s pomočjo krivulj. Diplomsko delo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2009.

[5] Patrick Bours and Soumik Mondal. Continuous authentication with keystroke dynamics. *Norwegian Information Security Laboratory NISlab*, pages 41–58, 2015.

[6] Coursera honor code. Dosegljivo: `https://learner.coursera.help/hc/en-us/articles/209818863-Coursera-Honor-Code`. [Dostopano: 23. 5. 2018].

[7] Introducing signature track. Dosegljivo: `https://blog.coursera.org/signaturetrack/`. [Dostopano: 23. 5. 2018].

[8] Css introduction. Dosegljivo: https://www.w3schools.com/css/css_intro.asp. [Dostopano: 23. 5. 2018].

[9] The web framework for perfectionists with deadlines — django. Dosegljivo: https://www.djangoproject.com/. [Dostopano: 27.8.2018].

[10] Docker - build, ship, and run any app, anywhere. Dosegljivo: https://www.docker.com/. [Dostopano: 27.8.2018].

[11] What is a container — docker. Dosegljivo: https://www.docker.com/resources/what-container. [Dostopano: 27.8.2018].

[12] Docker hub. Dosegljivo: https://hub.docker.com/. [Dostopano: 27.8.2018].

[13] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. Doktorsko delo, University of California, 2000.

[14] jstavanja/quiz-biometrics-api: Bachelor's thesis backend api code. Dosegljivo: https://github.com/jstavanja/quiz-biometrics-api. [Dostopano: 6.9.2018].

[15] jstavanja/quiz-biometrics-plugin: Moodle plugin for question type, that communicates with the quick-biometrics-api. Dosegljivo: https://github.com/jstavanja/quiz-biometrics-plugin. [Dostopano: 6.9.2018].

[16] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[17] K. S. Killourhy and R. A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. Technical report, June 2009.

[18] Andrew Maas, Chris Heather, Chuong Do, Relly Brandman, Daphne Koller, and Andrew Ng. Moocs and technology to advance le-

arning and learning research. *Ubiquity,(May 2014). http://doi.org/10.1145/2591684*, 2014.

[19] Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4):351–359, 2000.

[20] Features - moodle. Dosegljivo: https://moodle.com/features/. [Dostopano: 23. 5. 2018].

[21] Question type plugin how to — moodledocs. Dosegljivo: https://docs.moodle.org/dev/Question_type_plugin_how_to. [Dostopano: 28.8.2018].

[22] About numpy. Dosegljivo: https://docs.scipy.org/doc/numpy/about.html. [Dostopano: 23. 5. 2018].

[23] Php: Preface - manual. Dosegljivo: http://be2.php.net/manual/en/preface.php. [Dostopano: 23. 5. 2018].

[24] David W Pickton and Sheila Wright. What's swot in strategic analysis? *Strategic change*, 7(2):101–109, 1998.

[25] Postgresql: The world's most advanced open source database. Dosegljivo: https://www.postgresql.org/. [Dostopano: 27.8.2018].

[26] About python™. Dosegljivo: https://www.python.org/about/. [Dostopano: 23. 5. 2018].

[27] Semantic ui - design beautiful websites quickly. Dosegljivo: https://semantic-ui.com. [Dostopano: 23. 5. 2018].

[28] W3Schools.com. Introduction to html. Dosegljivo: https://www.w3schools.com/html/html_intro.asp. [Dostopano: 23. 5. 2018].