Web sustav za pohranu i vizualizaciju biometrijskih odrednica vlastoručnih potpisa s dodirnih zaslona

Bijelac, Domagoj

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Web sustav za pohranu i vizualizaciju biometrijskih odrednica vlastoručnih potpisa s dodirnih zaslona

Rijeka, rujan 2023.

Domagoj Bijelac
0069088384

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Web sustav za pohranu i vizualizaciju biometrijskih odrednica vlastoručnih potpisa s dodirnih zaslona

Mentor: izv.prof.dr.sc. Sandi Ljubić

Rijeka, rujan 2023.

Domagoj Bijelac
0069088384

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Ime Prezime

Sadržaj

Popis slika **vii**

Popis isječaka programskog koda **ix**

1 Uvod **1**

2 Razvoj web sustava za pohranu i vizualizaciju biometrijskih odrednica vlastoručnih potpisa s dodirnih zaslona **2**

2.1 Tehnologije korištene u razvoju 2

2.1.1 Python i korištene biblioteke 2

2.1.2 MongoDB 5

2.1.3 Jinja2 7

2.2 Protok podataka 10

2.3 Problemi i rješenja 12

3 Opis slučajeva korištenja **17**

3.1 Prikaz grafova 17

3.2 Dodavanje korisnika 28

3.3 Pregled svih korisnika i filtriranje 33

3.4 Usporedba korisnika 40

3.4.1 Analiza usporedbe potpisa dvaju korisnika 44

4 Zaključak **49**

Bibliografija **50**

Sažetak **51**

Popis slika

2.1 Izgled web stranice prije i poslije dodavanja sadržaja u osnovni predložak. 9

2.2 Primjer visoke razine protoka podataka u aplikaciji. 11

2.3 Prototip grafa potpisa korisnika bez temelja za web aplikaciju. . . . 12

2.4 Struktura složenog podatkovnog zapisa. 13

2.5 Uvećani prikaz potpisa bez zaglađivanja. 14

2.6 Lokalni utjecaj B-splajn interpolacije, samo jedan segment mijenja svoj oblik. 16

3.1 Vizualizacija potpisa korisnika. 19

3.2 Grafovi komponenata magnetskog polja. 21

3.3 Grafovi brzine dodirnih gesti s obzirom na komponente x i y. 22

3.4 Graf pritiska. 23

3.5 Multigrafovi brzine dodirnih gesti i vrijednosti magnetskog polja. . . 25

3.6 3D prikaz vektora magnetskog polja. 27

3.7 Prikaz stranice za dodavanje novih ili podataka postojećim korisnicima. 29

3.8 Iskočni prozori prilikom uspješnog i neuspješnog dodavanja korisnika u bazu. 32

3.9 Pregled svih korisnika i njihovih inačica potpisa u obliku kartica. . . 34

3.10 Detaljan prikaz sastava jedne kartice. 35

3.11 Filtriranje po imenu korisnika. 38
3.12 Prikaz jedne kartice prije i tijekom filtriranja inačica potpisa. 39
3.13 Prikaz stranice za unos podataka za usporedbu korisnika. 41
3.14 Prikaz dinamičkog mijenjaja broja inačice potpisa u ovisnosti o imenu korisnika. 43
3.15 Prikaz isječka stranice usporedbe potpisa korisnika. 45
3.16 Usporedba brzine izvođenja dodirnih gesti kod potpisa korisnika *a9* i *m7* 46
3.17 Usporedba grafova magnetskog polja očitanog pri potpisu korisnika *a9* i *m7*. 47
3.18 Usporedba pritiska tijekom izvođenja potpisa. 48

Popis isječaka programskog koda

2.1 Implementacija funkcije za zaglađivanje. 15

3.1 Implementacija funkcije za vizualizaciju potpisa. 18

3.2 Stvaranje vizualizacije vlastoručnog potpisa unutar HTML kôda. . . 19

3.3 Implementacija funkcije za prikaz pritiska i provjera nužnosti zaglađivanja. 23

3.4 Implementacija funkcije prikaz multigrafa magnetskog polja. 24

3.5 Implementacija funkcije za 3D prikaz komponenata magnetskog polja. 26

3.6 Implementacija skripte za osluškivanje podnošenja obrasca i iščekivanje odgovora. 28

3.7 Implementacija funkcije dodavanje datoteke s digitaliziranim potpisom. 30

3.8 Implementacija iskočnog prozora za obavijest o uspješnosti dodavanja korisnika. 31

3.9 Pojednostavljena struktura HTML kôda za prikaz svih korisnika. . . 35

3.10 Isječak kôda za pridruživanje korisničkog imena broju potpisa. . . . 36

3.11 Implementacija funkcije za filtiranje po imenu korisnika. 37

3.12 Skripta za osluškivanje promjene odabira korisnika. 40

3.13 Funkcija za promjenu broja potpisa u ovisnosti o odabranom korisničkom imenu. 42

Poglavlje 1

Uvod

Vlastoručni potpis jedna je od osnovnih odrednica naših identiteta. Potpis nije samo grafička mrlja na papiru, već rezultat kompleksnih i suptilnih pokreta naših ruku, zglobova i prstiju. Kao takav sa sobom nosi brojne biometrijske odrednice kojih nismo niti svjesni. Od promjena magnetskog polja zbog načina micanja ruke i efekta magnetskih materijala u okolini pa do točnog pritiska koji vršimo u određenim dijelovima potpisa, moguće je kvantificirati uz senzore i digitalne zaslone. Stoga, iako je uz puno vježbe moguće lažirati nečiji potpis, u grafološkom smislu, ostvariti isto na temelju tih biometrijskih odrednica gotovo je nemoguće.

Cilj ovog rada je vizualizirati biometrijske odrednice vlastoručnih potpisa s dodirnih zaslona. Osim vizualizacije, potrebna je i mogućnost dodavanja novih datoteka u bazu podataka te uspoređivanje inačice potpisa različitih korisnika kao i onih istog korisnika. Kako bi pronalazak podataka bio lakši, bitno je imati i filtriranje na temelju identifikatora korisnika i njihovih inačica potpisa.

U drugom poglavlju daje se pregled nekih od korištenih tehnologija, koje su služile u razvoju web aplikacije. Osim toga, analizirat će se i protok podataka u aplikaciji te neki od glavnih problema i njihova rješenja. Kroz opis slučajeva korištenja, pomoću brojnih vizualnih primjera i objašnjenih isječaka programskog kôda, vidljiva je realizacija svih ciljeva završnog rada i njihova međusobna interakcija.

Poglavlje 2

Razvoj web sustava za pohranu i vizualizaciju biometrijskih odrednica vlastoručnih potpisa s dodirnih zaslona

2.1 Tehnologije korištene u razvoju

U ovome potpoglavlju bit će opisane glavne tehnologije korištene u procesu razvoja web aplikacije. Svaka tehnologija imat će kratki uvod o svojim funkcionalnostima, koje su joj prednosti ili nedostaci te prije svega zašto je baš ona odabrana. Cilj je na kraju nadolazećih potpoglavlja bolje razumjeti korištene tehnologije, kako bi njihov doprinos bio jasniji u poglavlju 3.

2.1.1 Python i korištene biblioteke

Kao jedan od najpopularnijih programskih jezika visoke razine na svijetu, za znati što je *Python*, nije potreban posebno detaljan uvod. Velika većina programera barem jednom susrela se s ovim izrazito početnički prijateljskim i jednostavnim jezikom. Često asociran s podatkovnom znanošću, a u novije vrijeme i strojnim učenjem, *Pyt-*

hon je proširio svoj utjecaj u mnoge grane računarstva. Jedna od takvih je i razvoj web aplikacija. Iako naizgled netipičan kao prvi odabir za takav zadatak, zajedno s web okvirom *Flask* omogućuje stvaranje robusnih aplikacija. Kroz nadolazeće paragrafe istražit će se na koji način *Python* interaktira s *Flaskom*, koje su ostale glavne biblioteke korištene u radu, a s time i zaključiti zašto je baš *Python* odabran u implementaciji ovog zadatka.

Python programerima pruža bogatu standardnu biblioteku i širok ekosustav paketa trećih strana, što znatno olakšava rad i izvršavanje različitih zadataka. U izradi web aplikacija, *Python* kôd piše se za definiranje ruta, obradu zahtjeva i odgovora, interakciju s bazom podataka i izvođenje raznih zadataka specifičnih za aplikaciju uz pomoć nekog web okvira.

Jedan od takvih je *Flask*; minimalistički web okvir napisan u *Pythonu*. Nudi sve bitne značajke za izradu web aplikacija slijedeći pritom načelo jednostavnosti *Pythona*. *Flask* omogućuje rad sa svim tipičnim mehanizmima usmjeravanja zahtjeva, izradu predložaka i podršku za rukovanje i odgovaranje HTTP (engl. *Hypertext Transfer Protoko*l) zahtjevima. Osim toga, dopušta integraciju s bibliotekama poput *PyMongo*, za pristup bazi podataka, i *Plotly*, za stvaranje grafova. Dolazi i s pogonom za izradu HTML (engl. *HyperText Markup Language*) predložaka - *Jinja2* (engl. *templating engine*), čije će značajke posebno doći do izričaja u kasnijim poglavljima.

Osnovni princip rada i postavke okruženja *Flaska* mogu se opisati kroz nekoliko koraka [1]:

1. Uvoz *Flask* modula i kreiranje instance klase, obično imena `app`:

   ```
   from flask import Flask
   app = Flask(__name__)
   ```

2. Definiranje ruta i rukovođenje zahtjeva - u *Flasku* rute definiraju URL-ove na koje aplikacija odgovara i funkcije koje obrađuju te zahtjeve. Rute se definiraju pomoću `@app.route` dekoratora. Sljedeći isječak kôda predstavlja rutu koja odgovara na korijenski URL (engl. *Uniform Resource Locator*) (`"/"`) i korisniku vraća poruku: "Hello, World!":

```
@app.route("/")
def hello():
    return "<p>Hello, World!</p>"
```

3. Pokretanje aplikacije - na kraju *Python* datoteke dovoljno je dodati idući isječak kôda:

```
if __name__ == '__main__':
app.run()
```

Navedeni primjeri ukazuju na to koliko se brzo može namjestiti radno okruženje za razvoj web aplikacija u *Pythonu* koristeći *Flask*, a posebno kada se radi o jednostavnijim aplikacijama ili samostalnim projektima za učenje. Idući ključni razlog za odabir *Flaska* je već spomenuta neprimjetna integracija s raznim *Python* bibliotekama, od kojih je od najvećeg značaja u ovom radu bila *Plotly* za generiranje grafova.

Plotly pruža interaktivne i oku ugodne mogućnosti vizualizacije podataka. Podržava širok raspon vrsta grafova, uključujući: linijske, stupčaste, tortne grafove, raspršene dijagrame i još mnoge druge [2]. Interaktivne značajke *Plotlyja* omogućuju korisnicima zumiranje, pomicanje, postavljanje kursora iznad podatkovnih točaka za detalje i promjenu vidljivosti prikazanih serija u slučaju multigrafova. Osim širokog broja dostupnih grafova i interaktivnog načina korištenja, prednosti su mu i velika razina prilagođavanja i uređivanja tih grafova po vlastitom ukusu. Nastavljajući tradiciju jednostavnosti za stvaranje i prikaz samih vizualizacija, potrebno je svega nekoliko linija kôda vidljivih u primjeru ispod. Funkciji `px.line` predaju se x i y koordinate i naslov po želji te se funkcijom `.show` prikazuje graf.

```
import plotly.express as px
coords_x = [1, 2, 3, 4, 5]
coords_y = [10, 14, 18, 24, 30]

fig = px.line(x=coords_x, y=coords_y, title="Moj linijski graf")
fig.show()
```

Iako *Plotly* ima svoje nedostatke, u svrhu ovog rada oni nisu igrali veliku ulogu. Upravo zbog velike razine slobode i mogućnosti uređivanja grafova te vizualno kompleksnih prikaza u nekim slučajevima, može doći do sporijeg renderiranja grafova. No, kako podatkovni skup s podacima korisnika ne sadrži ogromne količine podataka, većih problema s performansama nije bilo.

Vrijedi napomenuti i glavnu negativnu stranu odabira *Pythona* kao glavnog programskog jezika. Naime, nije prikladan traži li se najbrže i optimalno rješenje. Za potrebe ovog rada njegova brzina ne igra ključnu ulogu, već njegova robusnost i integracija brojnih dobro održavanih biblioteka. Na taj način olakšan je cijeli razvojni proces i osiguran kohezivni tijek rada.

Detaljnija analiza *Plotlya*, *Flaska*, kao i njihove međusobne interakcije, uz konkretne primjere iz programskog kôda, bit će obrađena u poglavlju 3.

2.1.2 MongoDB

MongoDB je vrlo popularna na dokumentima zasnovana *NoSQL* baza podataka [3]. Podaci su organizirani unutar dokumenata (engl. *documents*) koji se nalaze unutar kolekcija (engl. *collections*). Za upite u bazu ne koristi se SQL (engl. *Structured Query Language*) programski jezik, već *MongoDB-ov* vlastiti jezik za upite MQL (engl. *MongoDB Query Language*). Odlikuje ga odlično rukovanje ne strukturiranih ili polu-strukturiranih, velikih količina podataka. U paragrafima ispod podrobnije će biti opisana neka od njegovih glavnih obilježja i kroz usporedbe sa klasičnim SQL pristupom otkriti zašto je baš MongoDB pravi izbor za ovu aplikaciju.

U svakoj aplikaciji gdje se spremaju neki podaci pa tako i u ovoj, jedno od najvažnijih pitanja koje se treba zapitati je na koji način će se oni spremati. Naime, ako je uvjet da aplikacija ima: visoku dosljednost podataka, kompleksna spajanja podataka iz više različitih tablica te strukturirani model podataka, tada su relacijske baze jasan odabir. U slučaju kada se očekuju veće količine nestrukturiranih podataka, puno čitanja, a malo promjena samih podatka, potrebna je fleksibilnost i agilnost NoSQL baza kao što je *MongoDB*.

Uzimajući u obzir neke od iznad navedenih obilježja, može se zaključiti kako

zbog vrste i količine podataka s kojom radi, *MongoDB* ne koristi klasičnu relacijsku bazu podataka. Primljeni podaci nisu modelirani i nemaju sheme niti primarne i strane ključeve. Radi se o ne-relacijskoj bazi podataka koja pomoću "horziontalnog" skaliranja (dodavanje više uređaja za raspodjelu opterećenja baze) pruža efikasan rad s velikom količinom podataka. Pri radu sa samo jednim uređajem nudi i "vertikalno" skaliranje [4]. Za predstavljanje i pohranu podataka, *MongoDB* koristi dokumente i kolekcije. Dokument je podatkovna struktura slična JSON-u (engl. *JavaScript Object Notation*), koja predstavlja jedan zapis. Može se zamisliti kao ekvivalent jednom retku u tradicionalnim bazama podataka. Kolekcija je pak grupa povezanih dokumenata. Povuče li se još jednom paralela na relacijske baze podataka radilo bi se o tablici, ali za razliku od tablica, kolekcije ne provode fiksnu shemu. Svaki dokument unutar zbirke može imati svoju vlastitu kompleksnu i ugniježđenu strukturu. Kako je tip izvora podataka u ovom radu upravo jedna takva kompleksna JSON datoteka, analizirana u potpoglavlju 2.3, vidi se jedan od razloga zašto je odabran upravo *MongoDB* sa svojim dokument modelom.

Sljedeći od ključnih razloga odabira *MongoDB-a* je njegova elegantna i brza integracija sa *Pythonom*. Putem *PyMongo* knjižnice omogućeno je sve od izvođenja jednostavnih CRUD (engl. *create, read, update, delete*) operacija pa do izvršavanja naprednih upita. Knjižnica sadrži opsežan skup razreda i metoda za rad s *MongoDB-om*. Neki od osnovnih slučajeva upotrebe su:

1. Veze - *PyMongo* omogućava uspostavljanje veze s *MongoDB* poslužiteljem pomoću klase `MongoClient`. Potrebno je specificirati adresu poslužitelja, port i po potrebi druge opcije povezivanja.
2. Manipuliranje podacima funkcijama:
   - insert_one() i insert_many() - ubacuje jedan ili više dokumenata u kolekciju
   - find_one() i find() - pronalazi jedan ili više dokumenata koji zadovoljavaju određene kriterije
   - update_one() i update_many() - ažurira jedan ili više odgovarajući do-

kumenata u kolekciji

- delete_one() i delete_many() - briše jedan ili više dokument iz kolekcije

3. Indeksiranje - *PyMongo* podržava stvaranje i upravljanje indeksima u *MongoDB-u*. Indeksi služe za poboljšavanje izvedbe upita na način da dopuštaju brže dohvaćanje podataka na temelju proslijeđenih podataka.

Jednostavnost spajanja na bazu vidljiva je u idućem primjeru:

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017")
mydb = client["signatures"]
mycol = mydb["userSignature"]
```

U primjeru iznad uvozi se biblioteka `pymongo`, a potom stvara varijabla `client`, koristeći razred `MongoClient`. Ona predstavlja početnu točku za interakciju s *MongoDB* bazom podataka. Klijent je spojen na *MongoDB* poslužitelj pokrenut na lokalnom računalu (engl. *localhost*), na zadanom portu - 27017. Već u idućem retku `client` se koristi za dohvaćanje baze podataka imena `signatures`, s *MongoDB* poslužitelja u varijablu `mydb`. Iz dobivene baze pristupa se željenoj kolekciji. U ovom slučaju to je `userSignature` spremljen u varijablu `mycol`.

PyMongo sveobuhvatna je i široko korištena *Python* biblioteka za rad s *MongoDB-om*. Omogućuje brzu i jednostavnu interakciju s *MongoDB* bazama podataka, olakšavajući izgradnju skalabilnih i fleksibilnih aplikacija.

2.1.3 Jinja2

Jinja2 je popularan i brz pogon za izradu predložaka dizajniran za *Python* web aplikacije [5]. Pruža mnoštvo funckionalnosti, koje olakšavaju proces generiranja dinamičkog sadržaja u web aplikacijama. Neki od tih korišteni za potrebe ove aplikacije su:

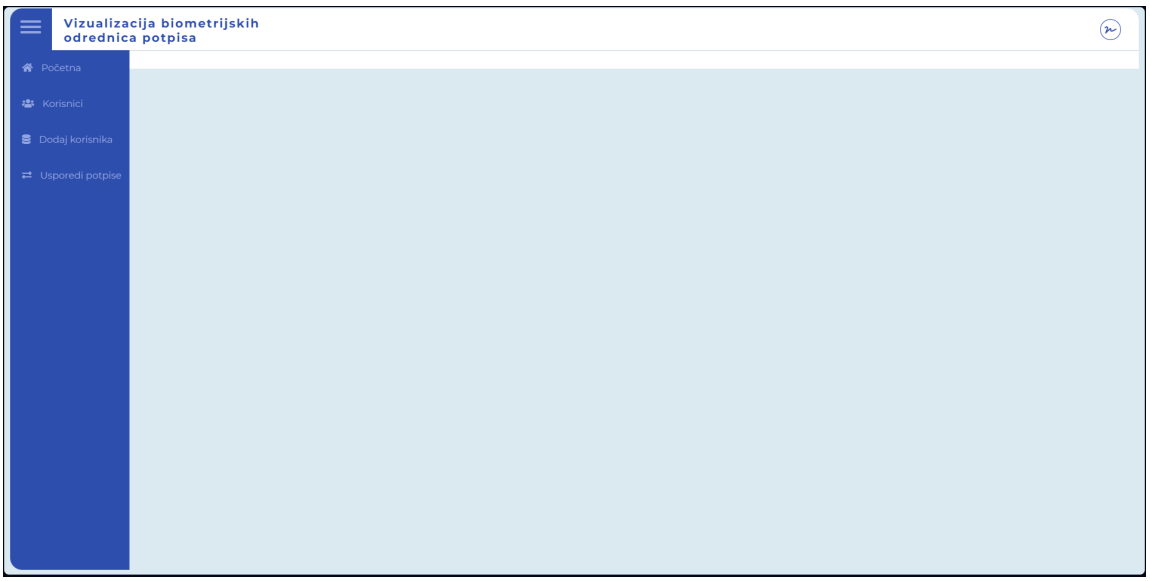
- Svojstvo primanja dinamičkih kontekstualnih podataka - mogućnost primanja i

manipuliranja raznih tipova podataka od običnih brojeva pa sve do kompleksnih objekata.

- Nasljeđivanje predložaka - pruža način kreiranja osnovnog predloška sa zajedničkim elementima, koji se ne mijenjaju. Neki od takvih su: navigacijska traka, hamburger izbornik, podnožje i sl. Primjer prije i poslije dodavanja sadržaja u naslijeđeni predložak nalazi se na slici 2.1.
- Kontrolne strukture - konstrukti kao što su petlje i grananja. Služe za iteriranje po rječnicima ili uvjetno uključivanje/isključivanje na temelju određenih kriterija.

Svaka od funkcionalnosti ima i svoju posebnu sintaksu, a sam programski kôd vrlo je sličan onome kod *Pythona*. Tako se kontekstualni podaci prikazuju unutar duplih vitičastih zagrada - `{{ime_varijable}}`. Kontrolne strukture prikazuju se unutar jednostrukih vitičastih zagrada - `{% for stavka in rječnik %}` i između znaka za postotak, a kraj `for` petlje označava izjava - `{% endfor %}`. Osnovni predložak definira se kao - `{% extends 'base.html' %}`, dok onaj podređeni, unutar kojeg se nalazi sadržaj web stranice, započinje s - `{% block content %}` i završava s - `{% endblock %}`.

Zaključno, *Jinja2* igra ključnu ulogu u razvoju ove aplikacije, pružajući učinkovit i dinamičan način za stvaranje HTML predložaka. Potiče dobru praksu održavanja kôda, njegove ponovne upotrebe i daje vrlo važnu oznaku dinamičnosti. Njezine značajke i intuitivna sintaksa neki su od brojnih razloga zašto baš ovaj pogon za izradu predložaka dolazi integriran s web okvirom *Flask*.



(a) Izgled početne stranice - predložak bez sadržaja

(b) Izgled početne stranice nakon dodavanja sadržaja.

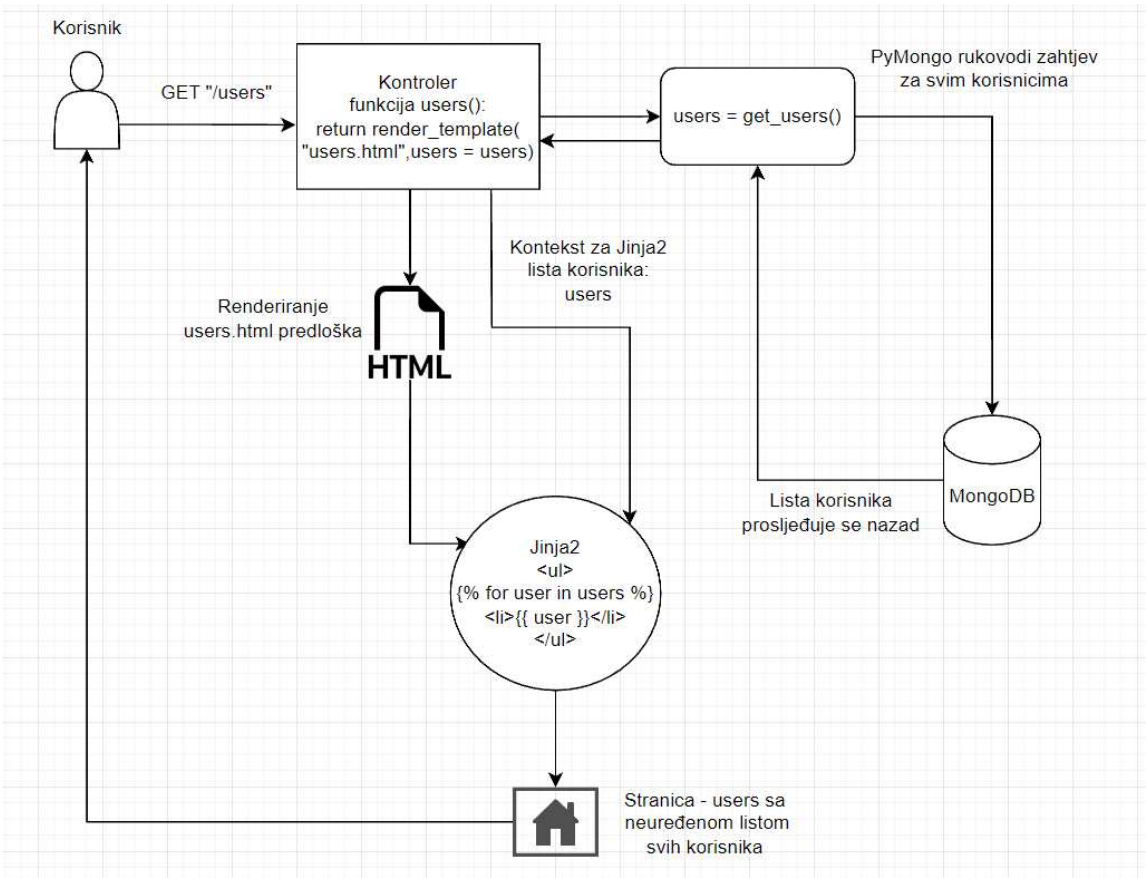
Slika 2.1 Izgled web stranice prije i poslije dodavanja sadržaja u osnovni predložak.

2.2 Protok podataka

Protok podataka u web aplikaciji najvećim dijelom ovisi o implementaciji o kojoj se radi. U većini slučajeva kada se pristupa nekoj od stranica kao što su: *Početna*, *Korisnici* ili *Usporedba korisnika*, protok podataka prati onaj prikazan na slici 2.2. Jedna od iznimaka je dodavanja korisnika objašnjena u poglavlju 3.2. Primjer sa slike kao takav ne koristi se u aplikaciji, već predstavlja prikaz protoka podataka s visoke razine. Pretpostavlja se da je krajnji cilj doći na stranicu - *Korisnici* na kojoj se nalazi neuređena lista imena svih korisnika. Sve počinje u trenutku kada korisnik klikne na željenu stranicu. Dolazi do GET zahtjeva na URL `/users`. Datoteka `app.py` u sebi sadrži dektorator `@app.route("/users")`, ispod kojeg je definirana funkcija `users()`. Ona komunicira s bazom podataka i prenosi te informacije za daljnji prikaz korisniku.

Varijabla `users` puni se sadržajem dobivenim iz baze podataka. Sadržaj je lista imena korisnika koji se nalaze u bazi. Zatim nastupa *Jinja2* koja ovisi o dva izvorna materijala [6], a to su: predložak `users.html` i dinamički kontekstualni podaci - varijabla `users`. Oboje dobiva iz funkcije *Flaska* - `render_template()`. Pomoću ta dva izvorna materijala stvara se predložak na temelju kojeg se renderira i konačni dokument. Za pristupanje listi `users` i varijablama općenito, koriste se dvostruke vitičaste zagrade. Neuređena lista stvara se iskorištavanjem *Jinja2* izjave (engl. *statement*) `{% for user in users %}` preko koje se iterira po listi korisnika.

```
@app.route("/users")
def users():
    users = get_users()
    return render_template("users.html", users = users)
```



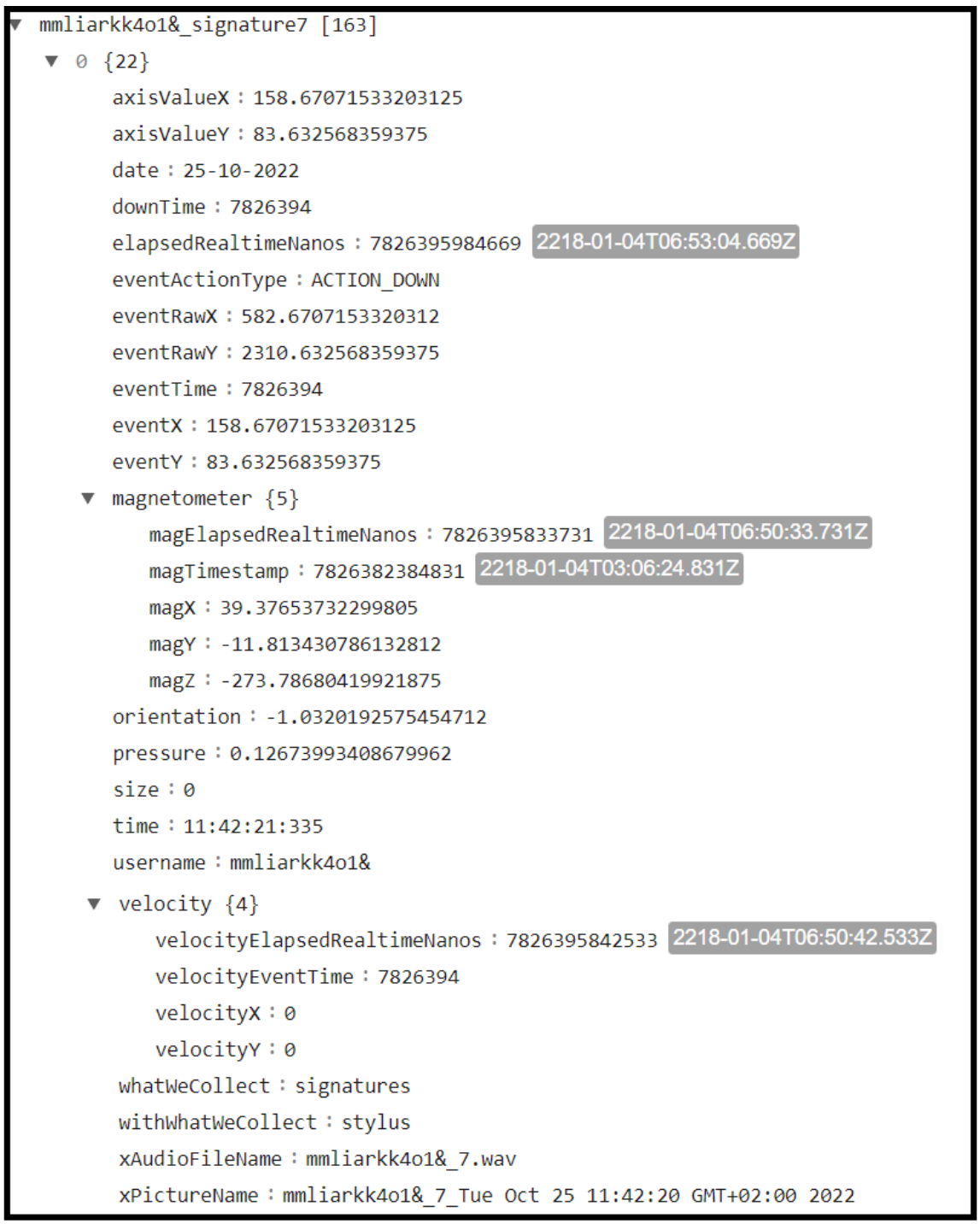
Slika 2.2 Primjer visoke razine protoka podataka u aplikaciji.

2.3 Problemi i rješenja

U procesu razvoja često puta pojavili su se razni izazovi. U nadolazećim primjerima bit će riječi o potonjima i na koji način su u konačnici uspješno prevladani. Ovi problemi i njihova rješenja pružaju uvid u pristup razvoju web aplikacije i prilagodbi, kako bi se osiguralo optimalno iskustvo rada.

Prvi od takvih pojavio se vrlo rano u razvoju u fazi stvaranja prototipa potpisa sa slike 2.3. Riječ je o dohvaćanju podataka iz baze. Kako bi razumjeli zašto je do toga došlo, mora se raščlaniti struktura podatkovnog zapisa. On se sastoji od jedne "glavne" liste objekata prikazane na slici 2.4. Lista u sebi sadrži 163 objekta od kojih svaki u sebi ima podatke o potpisu. Neki od njih također su objekti kao što je `magnetometer` ili `velocity` pa je tu riječ o ugniježđenim objektima. Srž problema čini unikatno ime liste, ovisno o imenu korisnika i broju potpisa. Konkretno, na slici 2.4, to je `mmliarkk401&_signature7`. Kako unaprijed nisu poznata sva imena korisnika i brojeva njihovih potpisa čini se nemoguće znati za kojeg korisnika se čitaju podaci. Iako je, u kasnijem istraživanju za rad, otkriveno da je moguće doći

Slika 2.3 Prototip grafa potpisa korisnika bez temelja za web aplikaciju.



Slika 2.4 Struktura složenog podatkovnog zapisa.

do unikatnih imena korisnika i pristupiti njihovim podacima, u počecima razvoja to nije bilo očito. Rješenje koje je implementirano bila je minimalna modifikacija samog zapisa, na način da se umjesto unikatnog imena svaka lista objekata zove `signature`. Imena korisnika i broj inačice potpisa dobiveni su iz parametra `xAudioFileName`. Jedino što je potrebno napraviti, je ukloniti nastavak ".wav" i ono što ostaje je ime korisnika i broj njegova potpisa u formatu `imeKorisnika + "_" + brojInačice`.

Drugi problem kod kojeg je bila potrebna dublja analiza je realističniji prikaz potpisa. S obzirom da se podatkovni zapis sastoji od samo 163 objekta, na puno mjesta potpis će izgledati isprekidano i nerealistično. Zumirani prikaz takvog potpisa nalazi se na slici 2.5. Prvotna ideja bila je interpolacijom zagladiti potpis. Specifično, pokušaj kubičnog splajna rezultirao je neuspjehom i pogreškom.

Slika 2.5 Uvećani prikaz potpisa bez zaglađivanja.

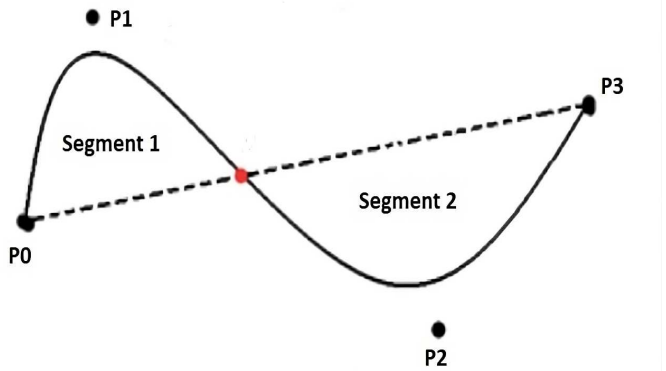
Pogreška se javlja jer kubične Bezierove krivulje zahtijevaju da x-koordinate budu jedinstvene [7]. Kod dostupnih potpisa to nije bio slučaj jer su zbog naravi potpisa na raznim mjestima bila moguća preklapanja, a time i ponavljanje istih x-koordinata. Iz navedenog razloga u konačnom rješenju korištena je B-splajn interpolacija [8]. Glavna razlika je u tome što kontrolne točke B-splajna daju lokalnu kontrolu nad oblikom krivulje, a ne globalnu kontrolu kao Bezierova krivulja. To znači da je utjecaj svake kontrolne točke ograničen na određeni interval definiran čvorom. Ovo

svojstvo omogućuje B-splajnu rukovanje dupliciranim x-koordinatama tako što ima različite kontrolne točke pridružene svakom duplikatu. Utjecaj kontrolnih točki na oblik krivulje kod B-splajn interpolacije prikazan je na nizu slika 2.6.

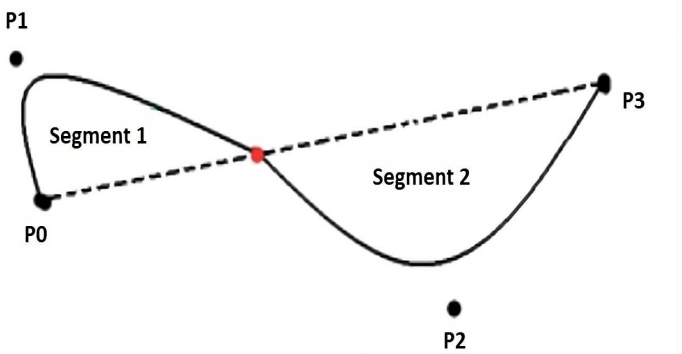
Zaglađivanje se tako odvija putem funkcije `smooth()`, prikazane na isječku programskog kôda 2.1, koja prima x i y koordinate potpisa. Prva linija definira čvorove jednako razmaknute u vrijednostima između 0 i 1 koristeći metode `numpy` knjižnice. Druga linija dodaje dodatne granične čvorove za glađe prijelaze na krajevima. Varijabla `tck` je tip podatka *tuple* koji sadrži vrijednosti čvorova, splajn koeficijente x i y te stupanj B-splajn krivulje. Jedino što još preostaje prije interpolacije, je izračun dodatnog parametera - `ut` potrebnog za evaluaciju krivulje [9].

```
1  import numpy as np
2  import scipy.interpolate as intrp
3  def smooth(x, y):
4      t = np.append([0, 0, 0], np.linspace(0, 1, len(x)-2,
       endpoint=True))
5      t = np.append(t, [1, 1, 1])
6
7      tck = [t, [x, y], 3]
8      ut = np.linspace(0, 1, (max(5*len(x), 10)), endpoint=True)
9
10     return intrp.splev(ut, tck)
```

Ispis 2.1 Implementacija funkcije za zaglađivanje.



(a) Izgled krivulje prije pomicanja kontrolne točke P1.



(b) Izgled nakon pomicanja P1 u lijevo: *Segment 2* ostaje isti i mijenja se samo *Segment 1*.

Slika 2.6 Lokalni utjecaj B-splajn interpolacije, samo jedan segment mijenja svoj oblik.

Poglavlje 3

Opis slučajeva korištenja

Najvažniji element cijele aplikacije upravo je vizualizacija biometrijskih odrednica potpisa kao i samog potpisa s dodirnog zaslona. Iz tog razloga izuzetno bitno bilo je odabrati odgovarajuće tehnologije i biblioteke koje mogu raditi sa složenim podatkovnim zapisom i napraviti pomoću njega sve potrebne vizualizacije. Uz puno slikovnih primjera i isječaka kôda kroz nadolazeća poglavlja prikazat će se pun potencijal biblioteka opisanih u poglavlju 2.1.1.

3.1 Prikaz grafova

Vizualizacija potpisa omogućena je koristeći modul *Plotly Express*, najčešće uvezen kao - `px`. On nudi više od 30 funkcija [10] za stvaranje različitih vrsta figura kroz svega nekoliko linija kôda.

Za vizualizaciju potpisa korisnika koristi se funkcija `px.line`, a sve počinje deklariranjem varijable `coords` koja predstavlja koordinate potpisa. Prvi put vidi se korištenje funkcije `smooth` iz poglavlja 2.3. Rezultat je puno više x i y točaka, što daje efekt zaglađenije i nesiprekidane linije. Zatim slijedi deklariranje same figure varijablom - `sign_graph`. Unutar `px.line`, svaka podatkovna točka, predstavljena je kao vrh (čija je lokacija dana stupcima x i y) polilinije u 2D prostoru [11].

Sljedeći neobavezan korak za prikaz grafa jest primjenjivanje metode `update_layout` koja mijenja stvari kao što su naslov, vrsta i veličina fonta, boja linije itd. Također,

daje bolju imerziju u potpis isključivši prikaz osi ordinate i apscise. S obzirom da ne utječe na tehničku stranu prikaza grafa, u ostalim primjerima zbog kraćeg prikaza kôda će biti izostavljena, makar se nalazi u izvornom kôdu. Isto vrijedi i za metodu `update_traces` koja mijenja izgled linije grafa (primjerice njezinu debljinu ili boju).

Posljednja stvar koju je potrebno napraviti je pretvoriti rječnik sadržan u `sign_graph`, koji je specifičan za *Plotly*, u JSON-formatirani znakovni niz pomoću prilagođenog `PlotlyJSONEncodera`, kako bi se osigurala ispravna serijalizacija podataka za prikaz u web aplikaciji te u razmjeni podataka. Tek onda ga možemo vratiti nazad u rutu koja ga prosljeđuje dalje za renderiranje u predlošku. Implementacija navedenoga prikazana je u ispisu 3.1

```
def get_signature_graph(data,username):
      coords = smooth(data["x"], data["y"])
      sign_graph = px.line(x = coords[0],y = coords[1])

      sign_graph.update_layout(
              yaxis_visible = False,
              xaxis_visible = False,
              title = f"Vizualizirani potpis korisnika - {
   username}",
              font = dict(
                family = "Arial",
                size = 16,
                color = "black"
          )
      )
        sign_graph.update_traces(patch ={"line":{"width":2.5}})
        sign_graph = json.dumps(sign_graph, cls=plotly.utils.
   PlotlyJSONEncoder)
      return sign_graph
```

Ispis 3.1 Implementacija funkcije za vizualizaciju potpisa.

Nakon što je proslijeđen predlošku, parsira se JSON-formatirani znakovni niz. U ispisu 3.2, vidljiv je način pristupanja varijabli `signature_graph` uz *Jinja2* sintaksu karakeriziranu dvostrukim vitičastim zagradama. Filter `safe` govori da je sigurno renderirati preneseni tekst direktno, odnosno da nije potrebno raditi enkodiranje za

Slika 3.1 Vizualizacija potpisa korisnika.

znakove koji mogu utjecati na HTML kôd. Funkciji `.newPlot` iz biblioteke *Plotly* predaje se ime - `div` elementa unutar kojeg će graf biti renderiran i podatke koje treba renderirati. U konačnici dobiven je graf potpisa korisnika na slici 3.1.

```
<script>
  var signature_graph = JSON.parse('{{ signature_graph | safe
   }}');
  Plotly.newPlot('signature-graph', signature_graph);
</script>
```

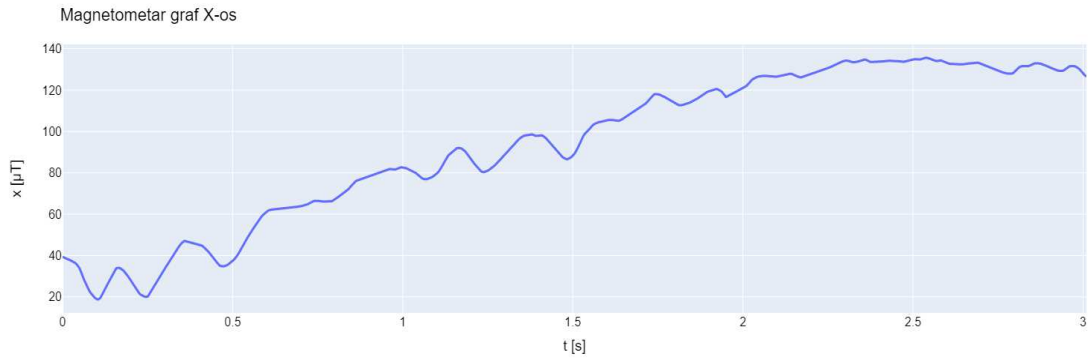
Ispis 3.2 Stvaranje vizualizacije vlastoručnog potpisa unutar HTML kôda.

Idući na redu pojedinačni su grafovi x, y i z magnetskog polja te x i y osi brzine. Sam kôd između njih vrlo je sličan i jedino što se mijenja jesu stvari kao: naslov, imena varijabli i korišteni podaci. Stoga prikazom kôda za samo jednu od osi, može se dobiti generalna ideja i kako su ostali implementirani:

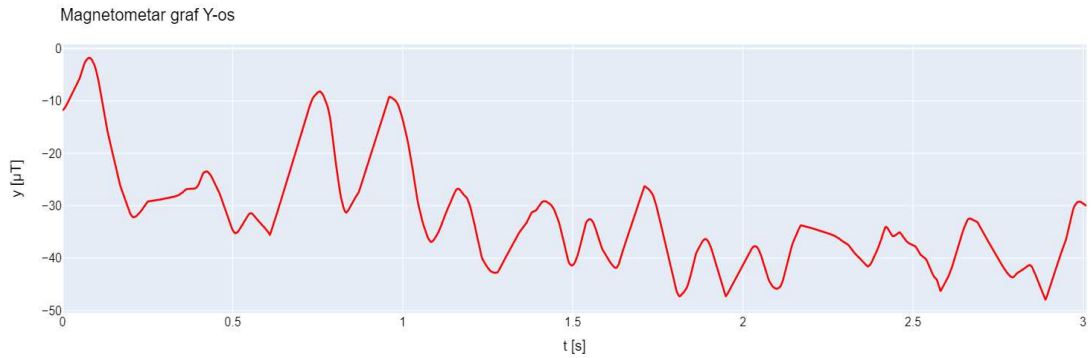
```
def get_magx_graph(data):
     coordinatesx = smooth(data["magnmtime"],data["xmagnm"])
     magx_graph = px.line(x = coordinatesx[0], y = coordinatesx[1])
     magx_graph = json.dumps(magx_graph, cls=plotly.utils.PlotlyJSONEncoder)

     return magx_graph
```

Kao što je ranije napomenuto u ovom i idućim primjerima, izbačene su metode `update_layout` i `update_traces` pa je sada još vidljivija snaga *Pythona* i knjižnice *Plotly*. Glavna razlika u usporedbi s funkcijom `get_signature_graph` je u podacima koje prosljeđuje funkciji `smooth`. Naime, ordinatu predstavlja vrijeme, konkretno u ovom slučaju - `magnmtime`. Izgled implementacija grafova komponenti magnetskog polja vidljiv je na nizu slika 3.2, a grafovi brzine dodirnih gesti na nizu slika 3.3.

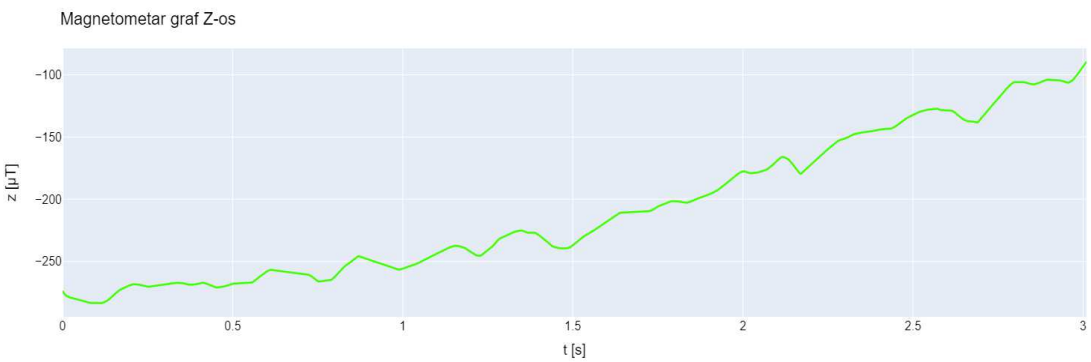
Još jedna od vizualiziranih biometrijskih odrednica je pritisak. Specifičnost kod njega je da korisnici čiji je potpis rađen prstom imaju konstantan pritisak od 1.0 (nedefinirana jedinica pritiska po vremenu). Zbog toga, prije zaglađivanja dodana je provjera je li ono potrebno funkcijom `check_if_elements_equal`. Funkcija vraća logički tip laž (engl. *False*), ukoliko je samo jedan broj različit od prvog tj. ako se ne radi o listi konstanti. U suprotnom, ako iteriramo do kraja liste bez da smo naišli na različit broj vraća se logički tip istina (engl. *True*) i zaglađivanje se preskače. Ostatak kôda vrlo je sličan primjerima iznad. Za x-os mogla se koristiti varijabla `data["velocitytime"]` ili `data["magnmtime"]`, oboje predstavlja isto vrijeme. Izvedba grafa pritiska vidljiva je na slici 3.4, a njegova implementacija na ispisu 3.3



(a) Magnetsko polje, x-komponenta.

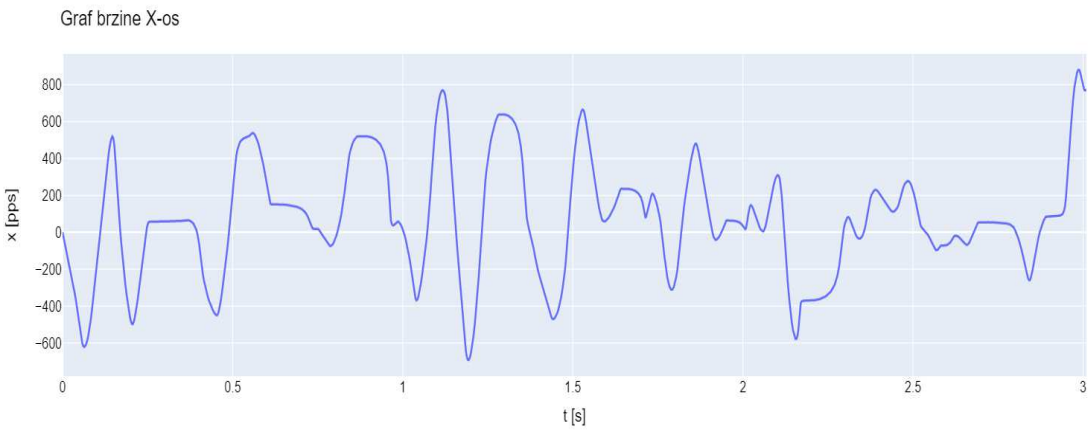


(b) Magnetsko polje, y-komponenta.

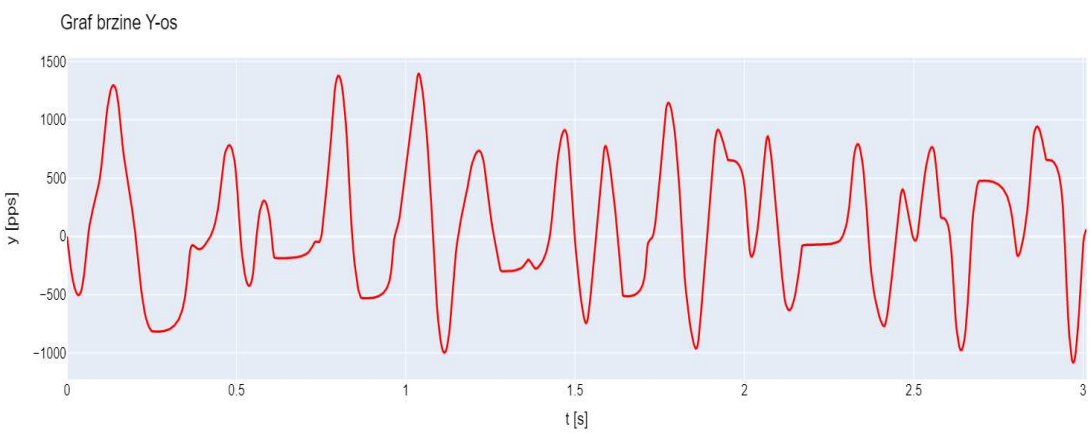


(c) Magnetsko polje, z-komponenta.

Slika 3.2 Grafovi komponenata magnetskog polja.



(a) Graf izvođenja dodirne geste, x-komponenta.

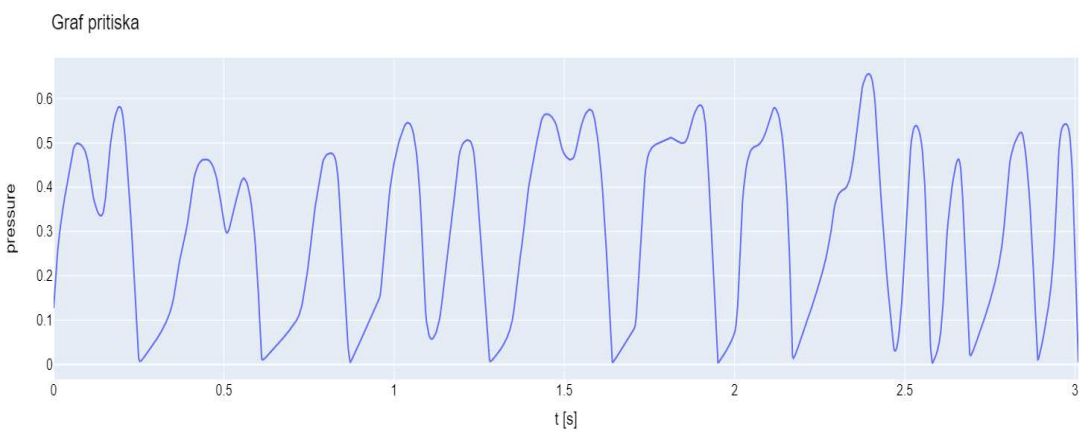


(b) Graf izvođenja dodirne geste, y-komponenta.

Slika 3.3 Grafovi brzine dodirnih gesti s obzirom na komponente x i y.

```
def check_if_elements_equal(data, control):
      for num in data:
            if num != control:
                  return False
      return True


def get_pressure_graph(data):
    if not check_if_elements_equal(data["pressure"], control = data[0]):
        coordinates = smooth(data["velocitytime"], data["pressure"])
    else:
        coordinates = [data["velocitytime"], data["pressure"]]

    pressure_graph = px.line(x = coordinates[0], y = coordinates[1])
    pressure_graph = json.dumps(pressure_graph,
    cls=plotly.utils.PlotlyJSONEncoder)

    return pressure_graph
```

Ispis 3.3 Implementacija funkcije za prikaz pritiska i provjera nužnosti zaglađivanja.

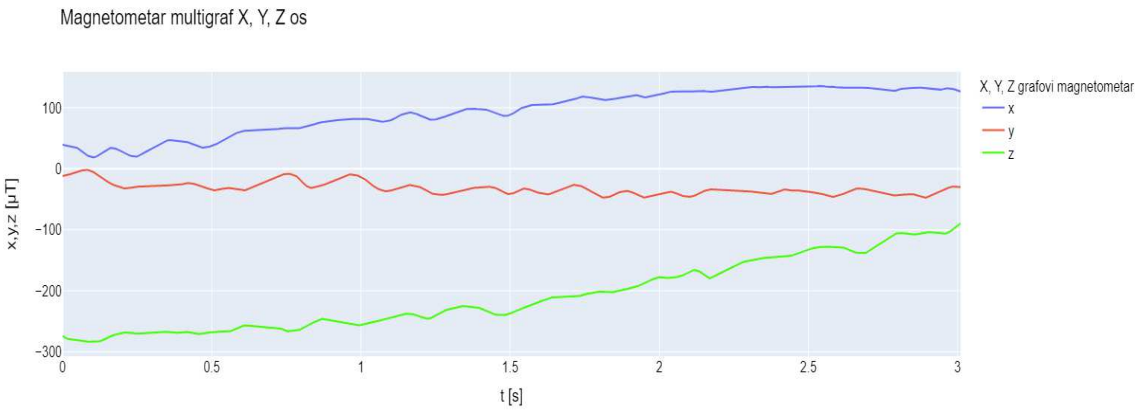


Slika 3.4 Graf pritiska.

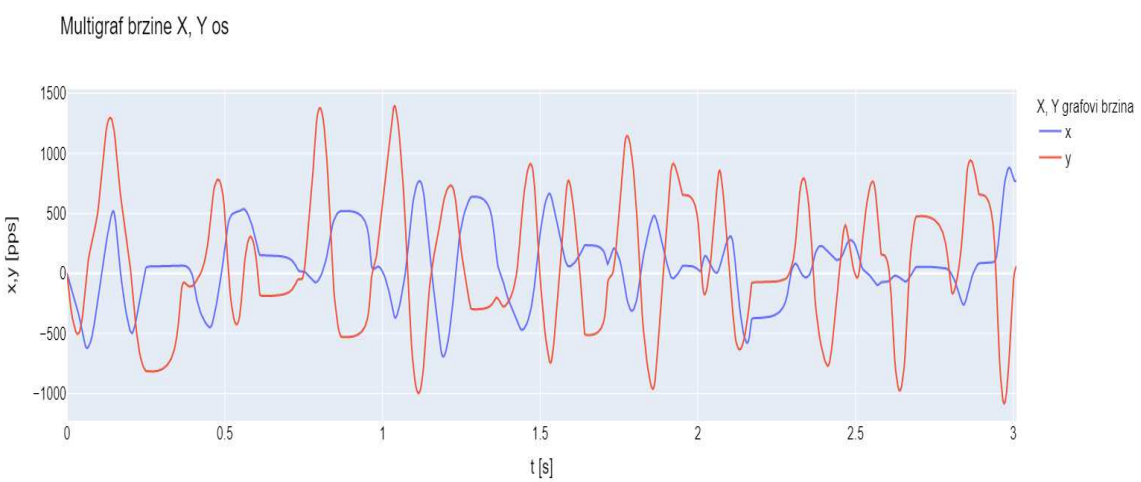
Iduća vrsta grafova su multigrafovi. Pomoću njih svi prethodni grafovi komponenata magnetskog polja i brzine dodirnih gesti mogu se staviti na jedan kompaktan, kompozitni graf. Time je dobivena bolja usporedba podataka i jasniji prikaz. Koordinate se dobiju vrlo slično kao prije i zaglađuju se baš kao u prethodnim primjerima. Novost kod multigrafova je korištenje modula `plotly.graph_objs` unesenog kao `go`. Iako je u svim dosadašnjim primjerima korišten modul `plotly.express`, svaki put kada se zovu njegove funkcije zapravo dobivamo instance *Plotly Graph* objekta [12]. Dakle, svaka figura proizvedena s bibliotekom *Plotly*, zapravo koristi objekte modula `plotly.graph_objs`. Kako *Plotly Express* ne dopušta kreiranje multigrafova, potrebno je spustiti se razinu niže i koristiti *Graph Objects*. Konstruiraju se tri odvojena `Scatter` objekta uz - `mode` definiran kao linijski graf i svakome pridijeljeno svoje ime. U konačnici stvorena je finalna figura, koja kao podatke prima tri prethodno stvorena objekta. Na taj način iz temelja je sagrađen vlastiti tip figure, koji se sastoji od tri različite vizualizacije. Rezultat programskog kôda prikazanog u ispisu 3.4 jesu multigrafovi magnetskog polja i brzine dodirnih gesti prikazani na nizu slika 3.5.

```
def get_all_magnm_graphs(data):
        coordsx = smooth(data["magnmtime"],data["xmagnm"])
        coordsy = smooth(data["magnmtime"],data["ymagnm"])
        coordsz = smooth(data["magnmtime"],data["zmagnm"])

        magx_graph = go.Scatter(x = coordsx[0], y = coordsx[1],
      mode = "lines", name = "x")
        magy_graph = go.Scatter(x = coordsy[0], y = coordsy[1],
      mode = "lines", name = "y")
        magz_graph = go.Scatter(x = coordsz[0], y = coordsz[1],
      mode = "lines", name = "z")

        mag_graph = go.Figure(data=[magx_graph, magy_graph,
   magz_graph])
        all_mag_graph = json.dumps(mag_graph, cls=plotly.utils.
   PlotlyJSONEncoder)

        return mag_graph
```

Ispis 3.4 Implementacija funkcije prikaz multigrafa magnetskog polja.



(a) Multigraf magnetskog polja.



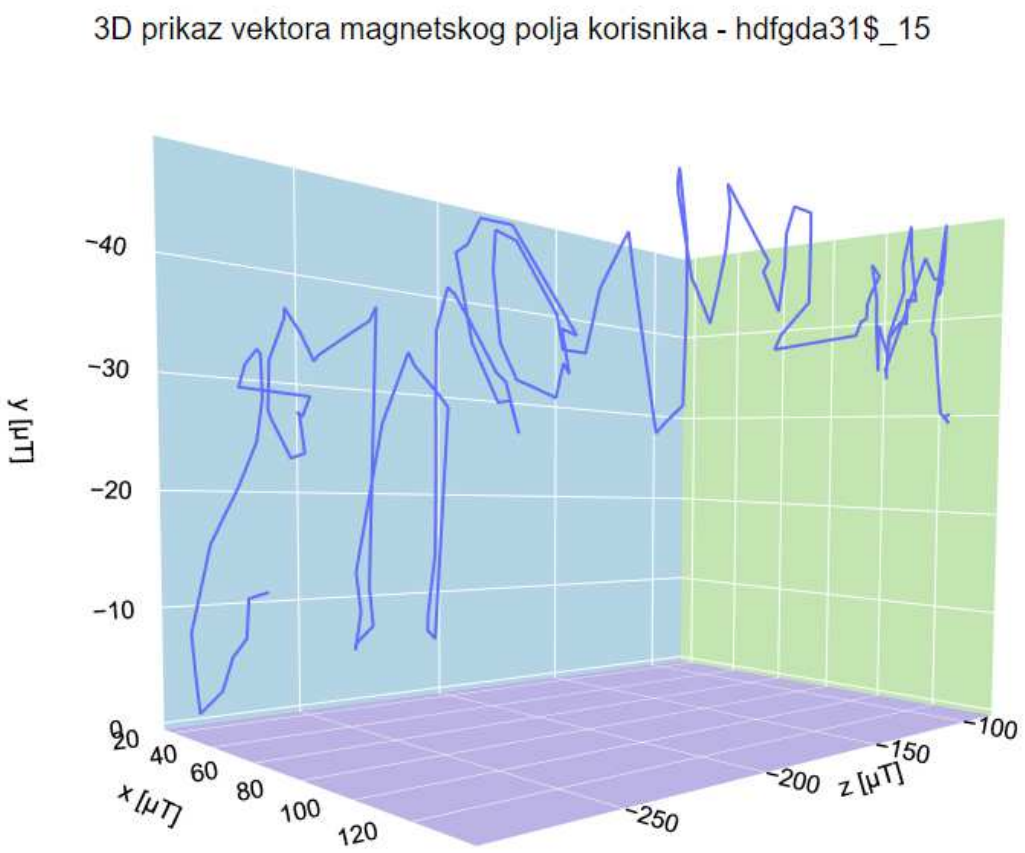
(b) Multigraf brzine izvođenja dodirnih gesti.

Slika 3.5 Multigrafovi brzine dodirnih gesti i vrijednosti magnetskog polja.

Zadnji i vizualno najzanimljiviji tip grafa je vizualizacija 3D trajektorija vektora magnetskog polja. Kako magnetometar daje podatke o sve tri komponente magnetskog polja - x, y i z, moguće ih je vizualizirati u 3D prikazu. To nudi još jedna od brojnih funkcija modula `plotly.express` - `line_3d`. U ispisu 3.5 funkciji se podaci prosljeđuju bez zaglađivanja, a pod parametar `title` prosljeđuje se ime korisnika. Još jedna od novosti u usporedbi s dosadašnjim figurama dolazi iz 3D naravi grafa, a to je varijabla `camera`. Pomoću nje određena je početna poziciju kamere koja gleda na figuru prilikom njezina renderiranja. Metodom `update_layout` ažurira se pozicija kamere na ranije definiran položaj u 3D prostoru. U konačnici rezultirani graf prikazan je na slici 3.6. Ono što se sa slike može primijetiti je velika sličnost ranije prikazanom potpisu korisnika na slici 3.1 u nižoj rezoluciji.

```
def get_3d_signature_graph(data, username):
    sig_graph_3d = px.line_3d(
    x = data["xmagnm"], y = data["ymagnm"], z = data["zmagnm"],
    title = f"3D graf podataka magnetomera korisnika - {username}"
    )

    camera = dict(
      up=dict(x = 1, y = 0, z = 0),
      center=dict(x = 0, y = 0, z = 0),
      eye=dict(x = 1.5, y = 1.5,  z= 0)
    )

    sig_graph_3d.update_layout(scene_camera = camera)
    sig_graph_3d = json.dumps(sig_graph_3d, cls=plotly.utils.PlotlyJSONEncoder)

    return sig_graph_3d
```

Ispis 3.5 Implementacija funkcije za 3D prikaz komponenata magnetskog polja.



Slika 3.6 3D prikaz vektora magnetskog polja.

3.2 Dodavanje korisnika

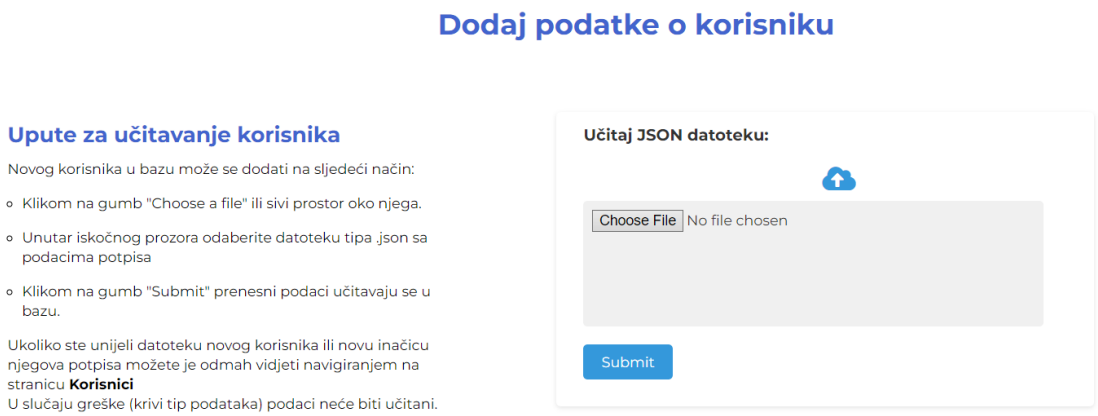
U ovome poglavlju detaljno će biti objašnjen proces dodavanja korisnika, odnosno datoteka s njihovim digitaliziranim potpisima. Za razumijevanje načina protoka informacija kod dodavanja novih korisnika potrebna je detaljnija analiza. Navigiranjem na stranicu - "Dodaj korisnika" prikazanu na slici 3.7 , predstavljeni smo s kratkim uputama za korištenje s lijeve strane i obrascem za unos datoteke s desne strane.

Unutar predloška `add_data.html` nalazi se *JavaScript* skripta koja osluškuje podnošenje obrasca na web stranici. Obrazac se podnosi klikom na gumb *Submit*. Prikupljeni podaci šalju se kao POST zahtjev poslužitelju. U programskom kôdu prikazanom u ispisu 3.6 vidi se na koji način je to implementirano.

```
1    <script>
2    const form = document.querySelector('form');
3    form.addEventListener('submit', async (e) => {
4        e.preventDefault();
5        const formData = new FormData(form);
6
7        const response = await fetch('/add_data', {
8            method: 'POST',
9            body: formData
10       });
11       const data = await response.json();
12                   .....
13   </script>
```

Ispis 3.6 Implementacija skripte za osluškivanje podnošenja obrasca i iščekivanje odgovora.

Prva linija kôda odabire prvi HTML element tipa - `<form>` koristeći metodu `querySelector` i dodjeljuje ga konstanti - `form`. Zatim je na konstantu `form` dodan slušatelj događaja (engl. *EventListener*), metodom `addEventListener`, slušajući za `submit` događaj definiran u HTML kôdu obrasca. Kada je obrazac podnesen, izršava se funkcija unutar slušatelja događaja. Linijom `e.preventDefault();` spriječeno je zadano ponašanje podnošenja obrasca. Neće doći do ponovnog učitavanja stranice i na taj način podaci će biti poslani asinkrono putem *JavaScripta*. Iduća linija stvara



Slika 3.7 Prikaz stranice za dodavanje novih ili podataka postojećim korisnicima.

novi `FormData` objekt i popunjava ga podacima obrasca. Nadolazeći redak funkcijom `fetch` šalje POST zahtjev poslužitelju na URL `/add_data`, prosljeđujući podatke obrasca kao tijelo zahtjeva i specificira HTTP metodu kao POST. Ključna riječ `await` osigurava da čekamo s daljnjim izršavanjem kôda, dok ne dobijemo odgovor.

Kôd za definiciju rute također je nešto drugačiji od ostalih. Unutar dekotratora `@app.route("/add_data", methods=['GET', 'POST'])` specificirano je da ruta može raditi i s GET i POST metodom. Grananjem se radi provjera je li zahtjev došao POST metodom i ako da dobiveni podaci se šalju u `add_user()` funkciju. Odgovor JSON formata se vraća pomoću funkcije *Flaska* - `jsonify`. U suprotnom znači da se radi o GET zahtjevu i jednostavno se renderira stranica za dodavanje korisnika:

```
@app.route("/add_data", methods=['GET', 'POST'])
def add_data():
        if request.method == "POST":
            resp = add_user(request.files['file'])
            return jsonify(resp)
        else:
            return render_template("add_data.html")
```

Funkcija `add_user()` prikazana u ispisu 3.7 okružena je `try, except` blokovima. Prilikom dodavanja korisnika pokušavaju se uhvatiti potencijalne greške, kako se one ne bi propagirale dalje u bazu ili da ne dođe do vremenskog ograničenja pristupnika. Glavna očekivana greška je `UnicodeDecodeError`, koja nastaje prilikom učitavanja podataka u funkciji `json.load()`. Do nje dolazi ukoliko je korisnik podnesao datoteku koja nije tipa JSON. U tom slučaju vraća se odgovor sa statusom "neuspješno" i daje do znanja korisniku zbog čega je došlo do greške. Drugi slučaj je generalni i uhvatit će bilo koju iznimku do koje je moglo doći te u poruci korisniku napisati o čemu se radi. Ukoliko je sve u redu, metodom `insert_one()` unose se proslijeđeni podaci u bazu.

```
    def add_user(file):
        try:
            data = json.load(file)
            mycol.insert_one(data)
            return {'status': 'success', 'message':
            'Korisnik uspjesno dodan u bazu!'}

        except UnicodeDecodeError:
            return {'status': 'failed', 'message':
            'Unesen krivi tip podataka, molimo unesite datoteku
    tipa .json'}

        except Exception as e:
            return {'status': 'failed', 'message':
            'Doslo je do pogreske pokusajte ponovno! - ' + str
    (e)}
```

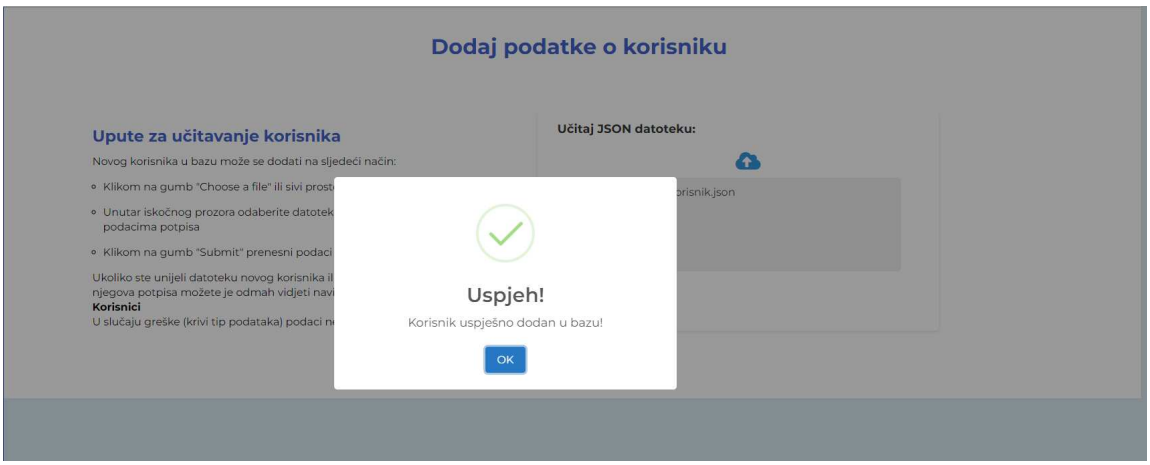
Ispis 3.7 Implementacija funkcije dodavanje datoteke s digitaliziranim potpisom.

Dobiveni odgovor u obliku JSON-a se parsira `.json()` metodom i podaci spremaju u konstantu `data`. Na temelju odgovora, ovisno o tome je li dodavanje bilo uspješno, korisniku će biti prikazana prilagođena iskočna poruka, kreirana *SweetAlert* knjižnicom. Iskočni prozor dobiven je pozivom metode `Swfal.fire` prikazane na ispisu 3.8 . Metoda prima rječnik u kojemu je definirana ikona ovisna o uspjehu ili neuspjehu, naslov i tekst unutar kojeg se nalazi poruka generirana u funkciji `add_user()`.

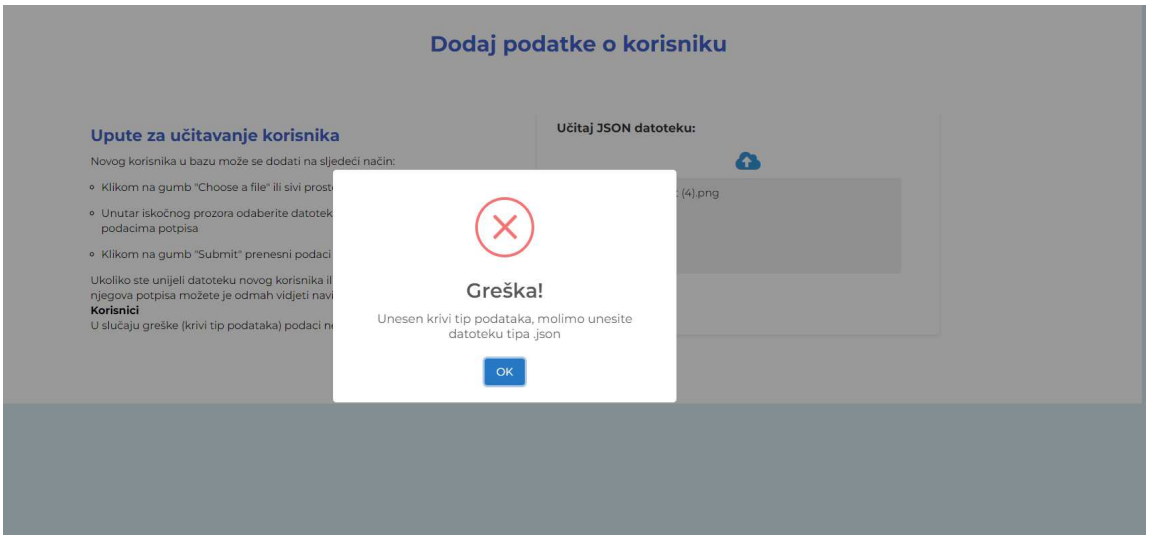
Primjer uspješnog i neuspješnog dodavanja korisnika nalazi se na nizu slika 3.8.

```
<script>
                .....
    if (data.status === 'success') {
        Swal.fire({
            icon: 'success',
            title: 'Uspjeh!',
            text: data.message
        });
    }else{
        Swal.fire({
            icon: 'error',
            title: 'Greska!',
            text: data.message
        });
    }
});
</script>
```

Ispis 3.8 Implementacija iskočnog prozora za obavijest o uspješnosti dodavanja korisnika.



(a) Iskočni prozor prilikom uspješnog dodavanja korisnika.



(b) Iskočni prozor prilikom neuspješnog dodavanja korisnika.

Slika 3.8 Iskočni prozori prilikom uspješnog i neuspješnog dodavanja korisnika u bazu.

3.3 Pregled svih korisnika i filtriranje

Pregled i filtriranje svih korisnika daje odličan uvid međusobnom nadopunjavanju *Flaska* i *Jinje2*. Zbog dodavanja potpuno novih korisnika ili novih inačica potpisa postojećim korisnicima, nadolazeći podaci nisu statični pa je potrebno osigurati način za rad s dinamičkim podacima. Cijeli proces kreće slanjem varijabli, popunjenih podacima o korisnicima iz baze u *Jinja2* predložak. U slučaju prikaza svih korisnika to su sljedeća dva rječnika:

- `short_user_dict` je skraćeni rječnik svih korisnika. Sadrži samo tri nasumično odabrana potpisa. Razlog tomu je slučaj u kojem jedan korisnik ima puno inačica potpisa, stoga su njihovim limitiranjem izbjegnuti potencijalni problemi u dizajnu. Rječnik je sastavljen na način da su ključevi imena korisnika, a vrijednosti skraćena lista njihovih inačica. Na primjeru ispod možemo zamisliti da *kor2* ima više od tri neproslijeđene inačice potpisa.

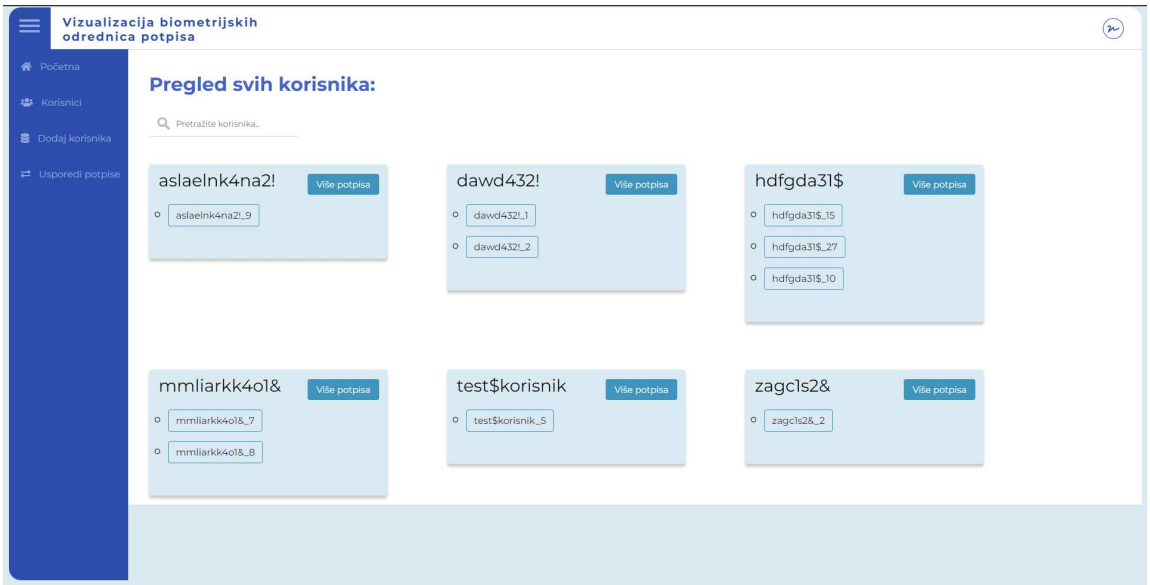
```
{
kor1: [kor1_potp7],
kor2: [kor2_potp3, kor2_potp6, kor2_potp9],
kor3: [kor3_potp1, kor3_potp2]
            ....
}
```

- `nums_user_dict` je drugi tzv. brojčani rječnik. Sadrži imena svih korisnika kao ključeve, a kao vrijednosti listu brojeva svih inačica potpisa tog korisnika. Primijetimo, *kor2* za kojega sada na raspolaganju stoje sve inačice potpisa.

```
{
kor1: [7],
kor2: [3, 6, 9, 8, 15, 4],
kor3: [1, 2]
            ....
}
```

Korisnici i informacije o njihovim potpisima prikazani su u obliku kartica. To je moguće iteriranjem kroz prethodno objašnjenje rječnike koristeći *Jinja2* izjave specifično - `for` petlje. Prolaskom kroz `short_user_dict` generiraju se kartice i početni prikaz stranice - *Korisnici* prikazan na slici 3.9. Kartice se nalaze unutar rešetki definiranih pomoću CSS-ovog (engl. *Cascading Style Sheets*) `grid-template-columns` svojstva. Svaka kartica sastoji se od dva dijela:

- Zaglavlje koje se dodatno dijeli na dvije strane:
  1. Lijevu - sadrži samo ime korisnika.
  2. Desnu - sastoji se od gumba za prikaz više potpisa.
- Tijelo unutar kojeg su do tri inačice potpisa korisnika spremne za brzi pristup.

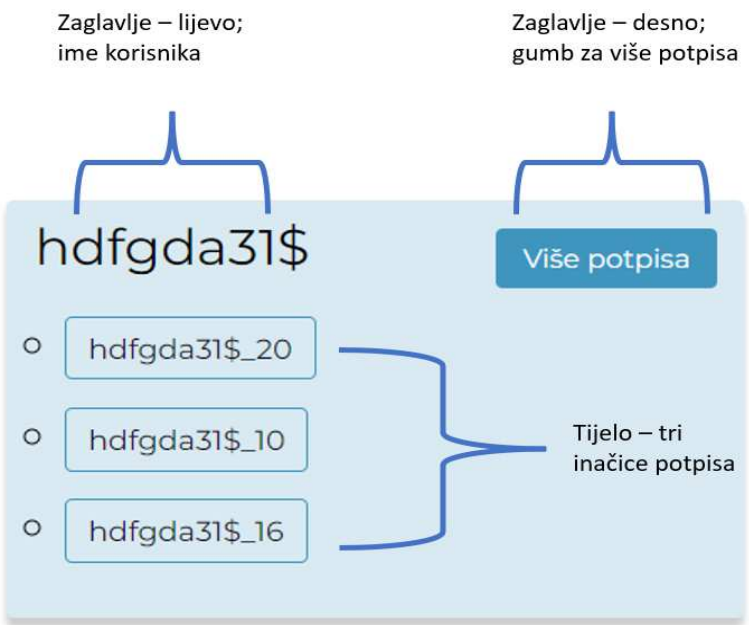


Slika 3.9 Pregled svih korisnika i njihovih inačica potpisa u obliku kartica.

Drugim riječima unutar HTML kôda implementirana je struktura sa ispisa 3.9 , detaljno razložena na slici 3.10:

```
<div class = "card-grid" id = "user-list">
    <div class = "card card-main">
        <div class = "card-header">
            <div class = "card-header-left">
            <span><div class="user"></div></span>
            </div>
            <div class = "card-header-right">...</div>
        </div>
        <div class = "card-body">
            <ul>...</ul>
        </div>
    </div>
</div>
```

Ispis 3.9 Pojednostavljena struktura HTML kôda za prikaz svih korisnika.



Slika 3.10 Detaljan prikaz sastava jedne kartice.

Dodavanjem izraza `{% for key in short_user_dict.keys() %}`, poslije razreda `card-grid` generira se onoliko `card card-main` klasa, a s time i kartica, koliko ključeva u sebi ima rječnik `short_user_dict`. Taj rječnik korišten je na još jednom mjestu, unutar razreda `card-body`. Služi za stvaranje neuređene liste tri inačice potpisa jednog korisnika. Elementima rječnika pristupa se varijablom `key` u izrazu `{% for usr_num in short_user_dict[key] %}`. Imena su zapisana unutar gumba, čijim se klikom otvara stranica za prikaz svih grafova tog korisnika i te inačice potpisa.

Osim u lijevoj strani zaglavlja i tijelu kartice, *Jinja2* izjave korištene su još i u desnoj strani zaglavlja za prikaz svih ostalih brojeva inačica potpisa u padajućem izborniku - *Više potpisa*. Ovoga puta radi se o brojčanom rječniku - `nums_user_dict`. Slično kao i u tijelu, pristupa se njegovim elementima za specifičnog korisnika ovisno o varijabli `key`, izrazom `{% for usr_num in nums_user_dict[key] %}`. Koristi se i novi izraz `set` za definiranje varijable - `imeKor`, koja je zapravo puno ime korisnika sa brojem inačice potpisa. To je učinjeno kako bi unutar padajuće liste za svaki broj bilo pridruženo i odgovarajuće korisničko ime, koje se potom šalje za prikaz svih grafova.

Način implementacije iznad objašnjenog prikazan je u idućem skraćenom isječku kôda prikazanom na ispisu 3.10.

```
<div id = "{{ key }}" class="dropdown-content">
    <input>
    {% for usr_num in nums_user_dict[key] %}
        {% set imeKor = key + "_" + usr_num|string %}
        <a href="{{ url_for('user', username=imeKor) }}">
                  {{usr_num }}</a>
    {% endfor %}
</div>
```

Ispis 3.10 Isječak kôda za pridruživanje korisničkog imena broju potpisa.

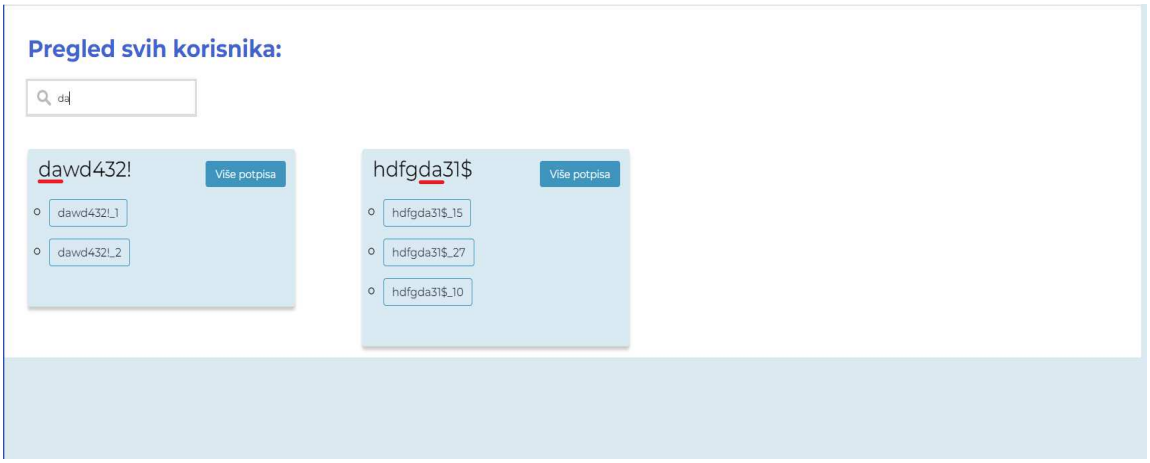
Za implementaciju filtriranja korisnika i inačica njihovih potpisa zaslužne su dvije *JavaScript* funkcije: `filterUsers()` i `filterNumber()`. Po načinu djelovanja u svojoj srži vrlo su slične uz male razlike pri dohvaćanju unosa, ali kako je algoritam za filter isti, bit će riječi samo o onoj za filtriranje korisnika prikazanoj na ispisu 3.11. Sve počinje od HTML elementa - `input`, koji u sebi ima definiran događaj imena `onkeyup`. On pokreće skriptu za filtriranje čim korisnik otpusti tipku na tipkovnici. Unos korisnika dohvaća se metodom `getElementById` zbog standardizacije i kasnije usporedbe, odmah stavlja u mala slova metodom `toLowerCase()`. Potom se dohvaćaju informacije iz spremnika `user-list` i dodjeljuju istoimenoj varijabli. Iz spremnika se vadi lista elemenata definiranih oznakom `span`, unutar kojih se nalaze imena korisnika, što je prethodno prikazano u pojednostavljenoj HTML strukturi na ispisu 3.9.

```
function filterUsers() {
  var input = document.getElementById("search-input");
  var filter = input.value.toLowerCase();
  var user_list = document.getElementById("user-list");

  var users = user_list.getElementsByTagName("span");
  var user_box = user_list.getElementsByClassName("card card-
   main ")

  for (var i = 0; i < users.length; i++) {
    var user = users[i].getElementsByClassName("user")[0];
    if (user.innerHTML.toLowerCase().indexOf(filter) > -1) {
      user_box[i].style.display = "";
    } else {
      user_box[i].style.display = "none";
    }
  }
}
```

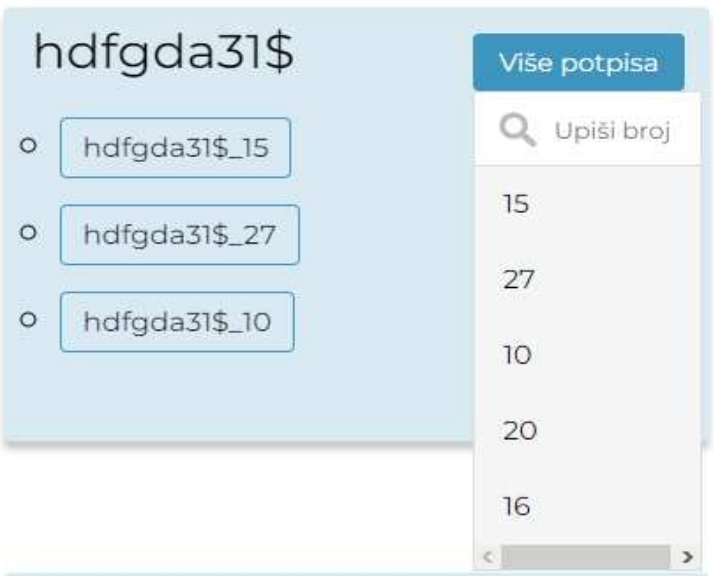
Ispis 3.11 Implementacija funkcije za filtiranje po imenu korisnika.

Jedino preostalo je dobiti glavni spremnik kartice spremljen u varijabli `user_box`. S njime unutar `for` petlje dinamički se skrivaju ili prikazuju potrebne kartice ovisno o unosu korisnika. U petlji dolazi do grananja, gdje se metodom `indexOf()` traži

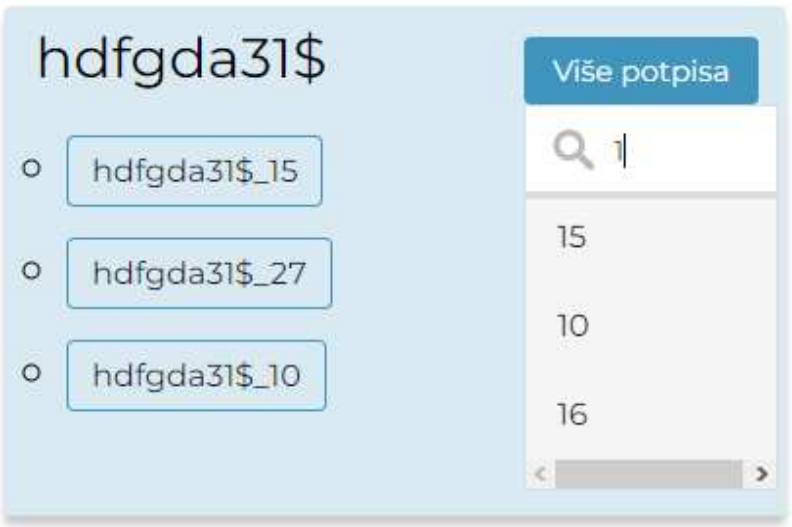
postoji li podudaranje s unosom i imenom korisnika. Ako podudaranja nema, metoda vraća -1 i svojstvo `display` spremnika `card card-main` postaje `none`. Na taj način sakrivene su sve one kartice kod kojih se znakovni niz proslijeđen od strane korisnika nigdje ne preklapa s imenom postojećih korisnika. Primjer toga u praksi prikazan je na slici 3.11, a primjer filtriranja inačice potpisa na nizu slika 3.12.



Slika 3.11 Filtriranje po imenu korisnika.



(a) Početni izgled kartice i svi potpisi bez unosa.



(b) Filtriranje inačice potpisa.

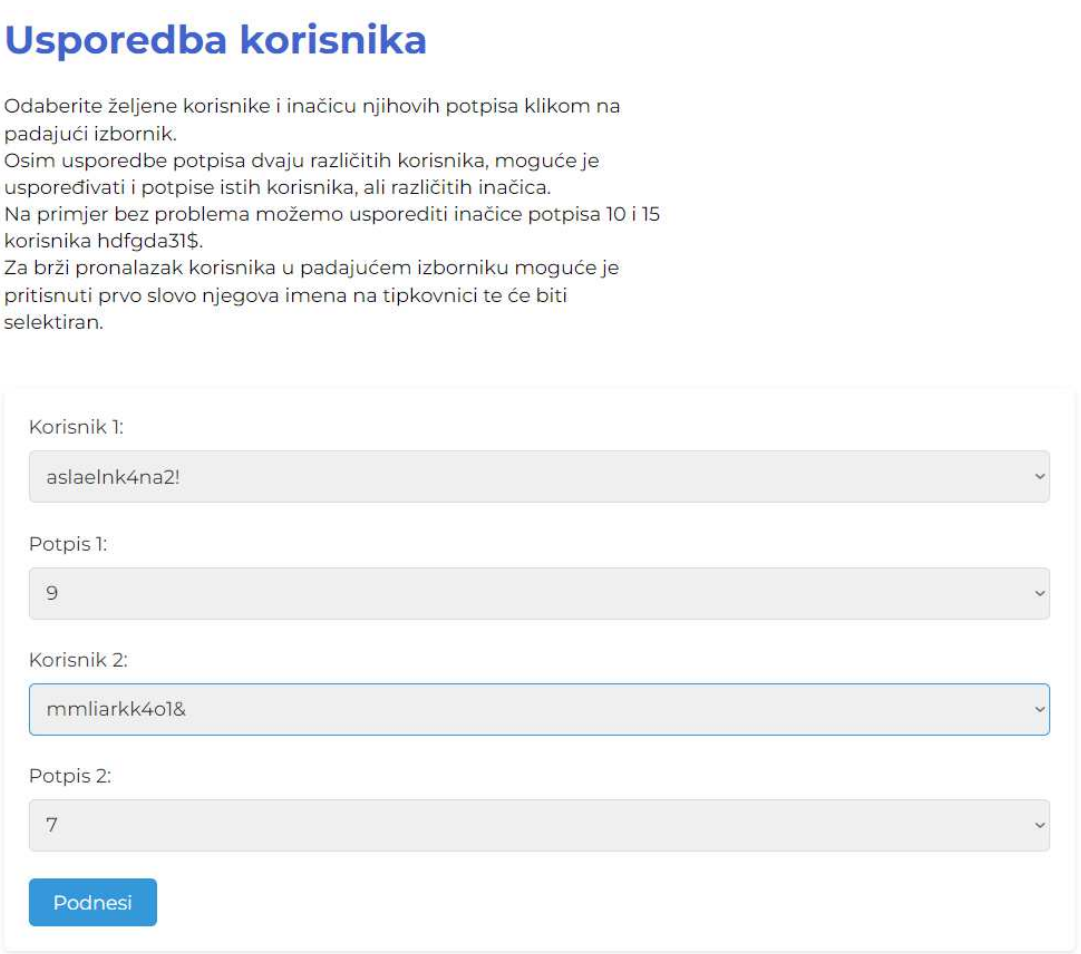
Slika 3.12 Prikaz jedne kartice prije i tijekom filtriranja inačica potpisa.

3.4 Usporedba korisnika

Nakon što je opisano kako dodavati korisnike, pregledavati njihove grafove i filtrirati ih po imenu i inačici potpisa, u idućim potpoglavljima bit će govora o usporedbi njihovih potpisa. Navigiranjem na stranicu *Usporedi potpise* - slika 3.13 glavna stvar je obrazac unutar kojeg se odabire ime korisnika, a sukladno tome i broj inačica njegova potpisa. Ono što se odmah može zamijetiti je kako drugi ulaz, odnosno broj potpisa ovisi o prvom ulazu, tj. imenu korisnika. Stoga je na dinamički način potrebno mijenjati izbor inačica potpisa. Ulazni podatak u formu iste je strukture i imena kao brojčani rječnik `nums_user_dict` iz potpoglavlja 3.3. Prije svega, parsiraju se ulazni podaci i pridjeljuju konstanti `user_numbers`, zatim se metodom `getElementById()` dohvaća sadržaj HTML elementa - `select` za prvog i drugog korisnika. To uključuje ime i broj potpisa korisnika. Potom se na oba elementa dodaje slušatelj događaja, koji prilikom promjene odabira zove funkciju `update_number_select_options()`. Posljednja stvar ostala za napraviti je inicijalizirati opcije za odabir broja inačice potpisa na temelju početno odabranog korisničkog imena. U primjeru isječka kôda u ispisu 3.12 , radi bolje preglednosti, izbačeno je: dohvaćanje podataka, dodavanje slušatelja i inicijalizacija za odabir drugog korisnika za usporedbu. Postupak je isti te se jedina razlika nalazi u imenu konstanti.

```
<script>
  const user_numbers = JSON.parse('{{ nums_user_dict|tojson|
    safe }}');

  const username1_select=document.getElementById("username1");
  const number1_select=document.getElementById("number1");

  username1_select.addEventListener("change",() =>{
    update_number_select_options(username1_select,
    number1_select);
  });
  update_number_select_options(username1_select, number1_select
    );
</script>
```

Ispis 3.12 Skripta za osluškivanje promjene odabira korisnika.



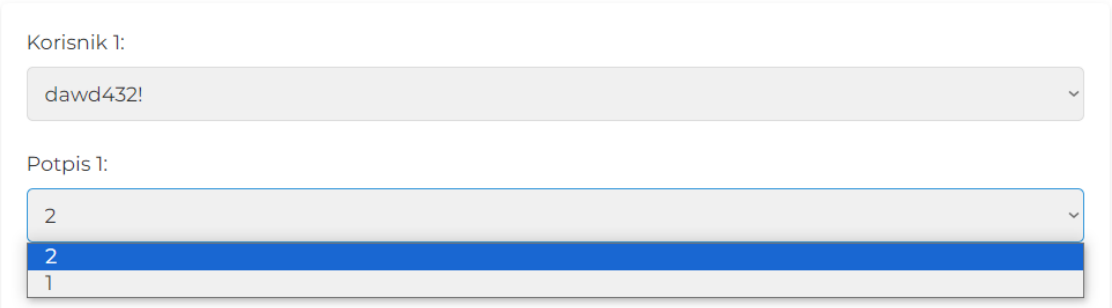
Slika 3.13 Prikaz stranice za unos podataka za usporedbu korisnika.

Kao što se može vidjeti iz ispisa 3.12, funkcija koja omogućuje dinamičku narav prikaza brojeva potpisa je `update_number_select_options` i kao argumente prima sadržaj obrasca. Prikazana u ispisu 3.13 za ime korisnika prima `username_select`, a za broj potpisa `number_select`. Ime korisnika dobiva se iz svojstva `.value`, a lista brojeva inačica potpisa `numbers` kreira se u ovisnosti o imenu korisnika. `While` petlja vrti se dok god čvor dobiven svojstvom `firstChild` nije prazan. Ukoliko postoje, svi prethodni brojevi potpisa brišu se metodom `removeChild()`, kako bi bio osiguran prikaz samo onih asociranih za tog korisnika. Kada su sve opcije izbrisane, iteracijom kroz listu brojeva potpisa kreirane su nove opcije metodom `.createElement()` i dodane kao jedan od izbora pomoću `appendChild()` metode.

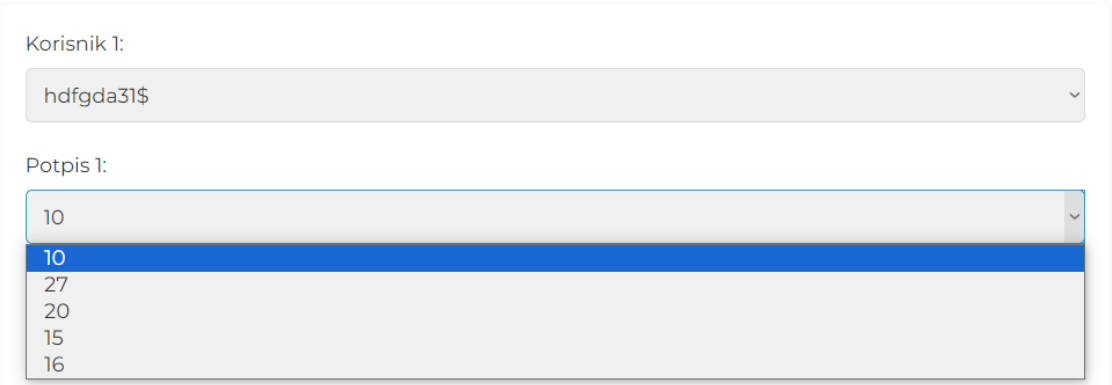
Primjer rada funkcije najbolje se vidi na nizu slika 3.14

```
<script>
function update_number_select_options(username_select,
   number_select) {
    const selected_username = username_select.value;
    const numbers = user_numbers[selected_username];

    while (number_select.firstChild) {
      number_select.removeChild(number_select.firstChild);
    }

    numbers.forEach((number) => {
      const option = document.createElement("option");
      option.value = number;
      option.textContent = number;
      number_select.appendChild(option);
    });
  }
</script>
```

Ispis 3.13 Funkcija za promjenu broja potpisa u ovisnosti o odabranom korisničkom imenu.



(a) Brojevi potpisa za korisnika *hdfgda31$*.



(b) Brojevi potpisa za korisnika *dawd432!*.

Slika 3.14 Prikaz dinamičkog mijenjaja broja inačice potpisa u ovisnosti o imenu korisnika.

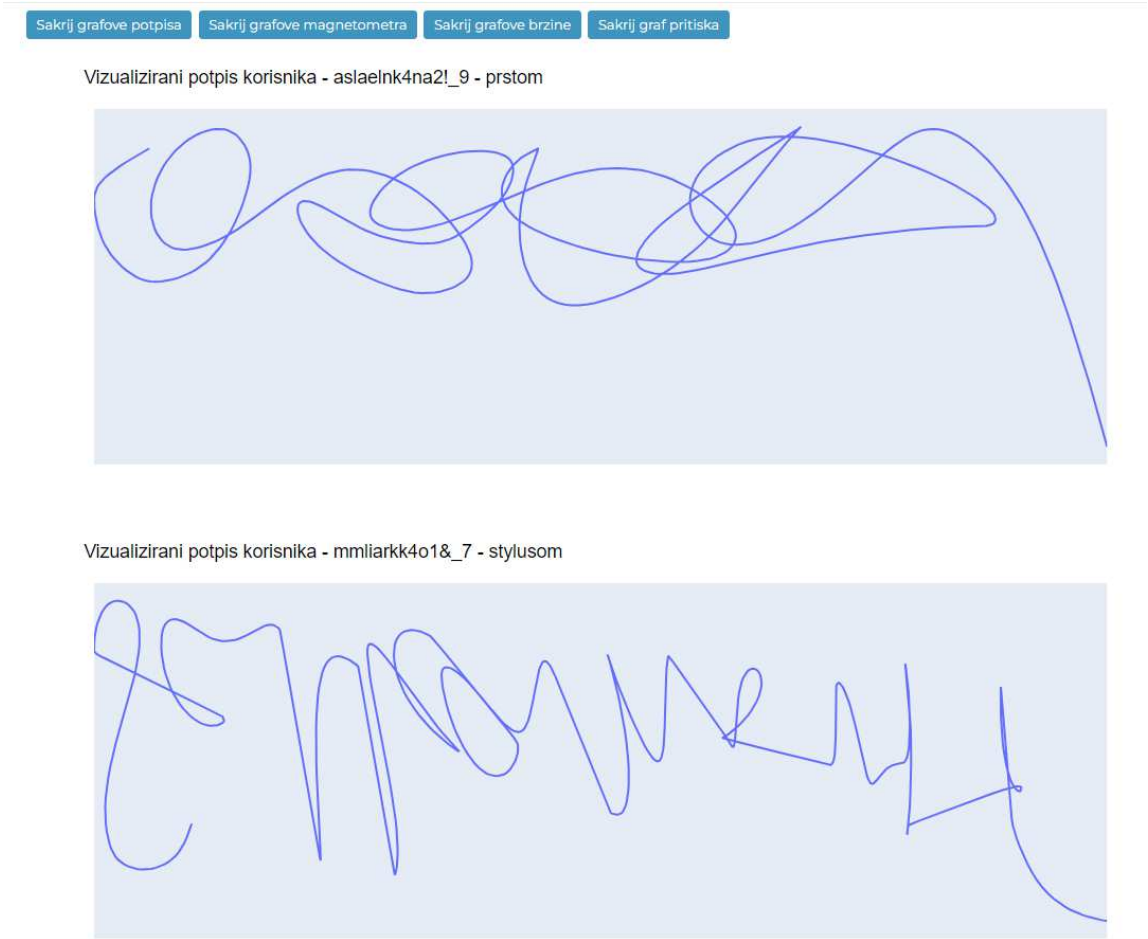
3.4.1 Analiza usporedbe potpisa dvaju korisnika

Kada je obrazac podnesen sa željenim korisnicima za usporedbu ili s istim korisnikom, ali različitom inačicom potpisa u novom prozoru, otvara se stranica prikazana na slici 3.15. Vrlo sličan dizajn uz gumbove za skrivanje grafova nalazi se i pri pregledu pojedinačnih korisnika i njihovih grafova iz poglavlja 3.1. U ovom primjeru analizirat će se potpis korisnika *aslaelnk4na2!_9* (*a9*), i *mmliarkk4o1&_7* (*m7*), gdje brojevi iza znaka "_" predstavljaju inačicu potpisa. Bitno je za napomenuti i način prikupljanja potpisa. Naime, korisnik *a9* potpisao se prstom, dok je korisnik *m7* potpis obavio digitalnom olovkom (engl. *stylus*). Iako se na prvi pogled ne čini znatna, kasnije ta činjenica igra veliku ulogu u usporedbi grafova magnetskog polja i pritiska.

Počevši od razlike u brzini x i y osi na nizu slika 3.16, najznatnija stvar je kraće trajanje potpisa korisnika *a9* prikazano crvenom bojom. Specifično vrijeme potpisa je 2.31 sekundi, što je za 0.7 sekundi kraće od korisnika *m7* čije je vrijeme potpisa 3.01 sekundi. Nadalje, na x osi kod korisnika *a9* nailazimo na velike varijacije, što indicira i veću akceleraciju, a sukladno time i potkrepljuje kraće vrijeme potpisa. Na y osi stvari su nešto sličnije, izuzev par slučaja na samom početku prije, nego što je prošlo 0.5 sekundi, na sredini oko 1.5 sekunde i na kraju potpisa korisnika *a9*. To se moglo i pretpostaviti pregledom potpisa *m7* na slici 3.15, gdje je vidljiv širok raspon točaka na ordinati s puno maksimuma i minimuma.

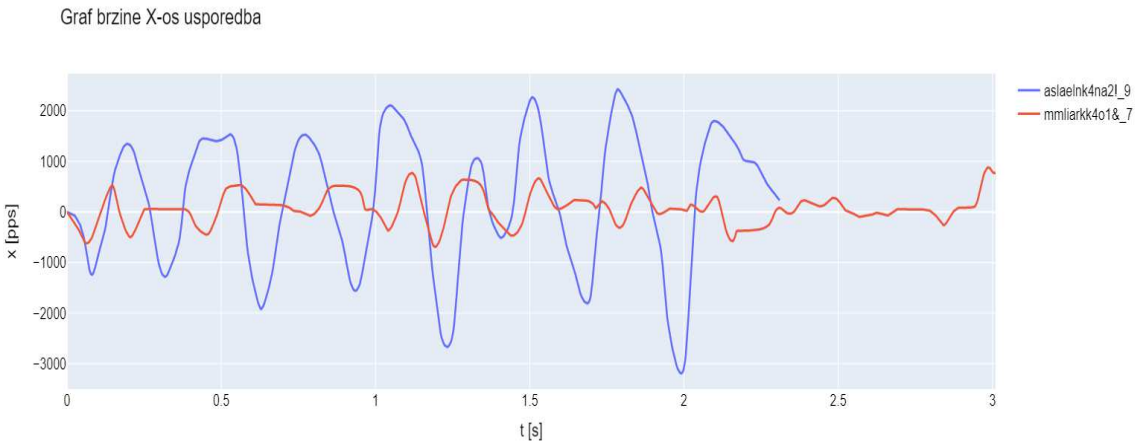
Tek kod grafova magnetskog polja vidi se značaj ove specifične integracije. Do sada zadovoljavajuće usporedbe mogle su se dobiti i otvaranjem dva zasebna prozora, za svakog korisnika po jedan, i na taj način uspoređivati grafove. No, ono što takvom pristupu nedostoje, jest kontekstualizacija. Na nizu slika 3.17 vidljiva je velika razlika između podataka korisnika *m7* i *a9*. Graf magnetskog polja potonjeg u ovoj usporedbi izgleda gotovo kao konstanta uz minimalne promjene na y osi. Kada bi se gledali zasebno, takve razlike na prvi pogled ne bi bile očite. Ako se prisjetimo da je način prikupljanja potpisa *a9* bio prst, stvari postaju nešto jasnije.

Zadnja usporedba koju se može načiniti je između pritiska i vidljiva je na slici 3.18. Još više negoli na prethodnom primjeru, ovdje je utjecaja imala razlika u načinu prikupljanja potpisa. Kod korisnika *m7* vidi se jasna slika uniformnih brijegova i

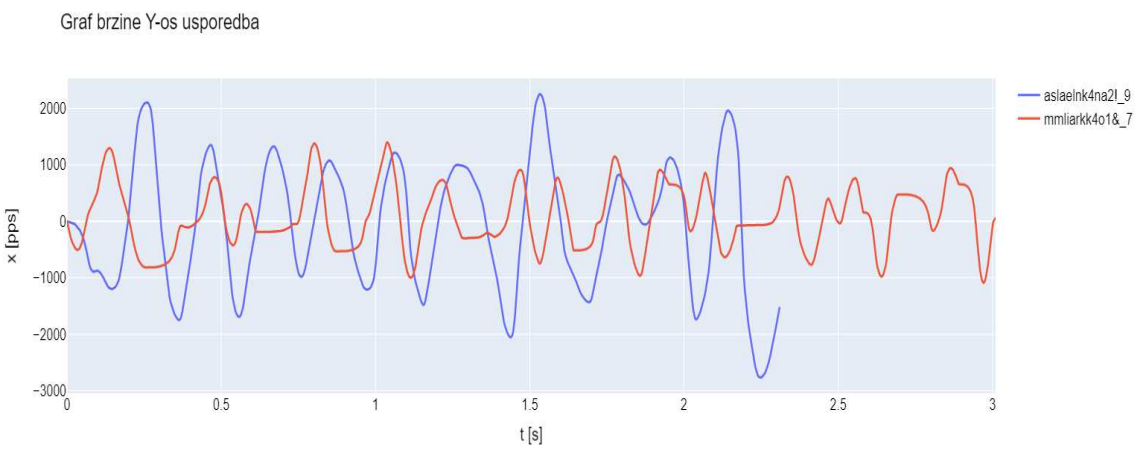


Slika 3.15 Prikaz isječka stranice usporedbe potpisa korisnika.

dolova koji savršeno odgovaraju samom potpisu; jači pritisak pri penjanju na y osi, a slabiji pri spustu. Graf pritiska korisnika *m7* je zapravo vrlo sličan njegovu grafu brzine y osi, tokom većeg ubrzanja korisnik ostavlja jači otisak po svoj prilici jer je jačim stiskom držao digitalnu olovku. S druge strane, graf korisnika *a9* je konstanta vrijednosti 1.00.

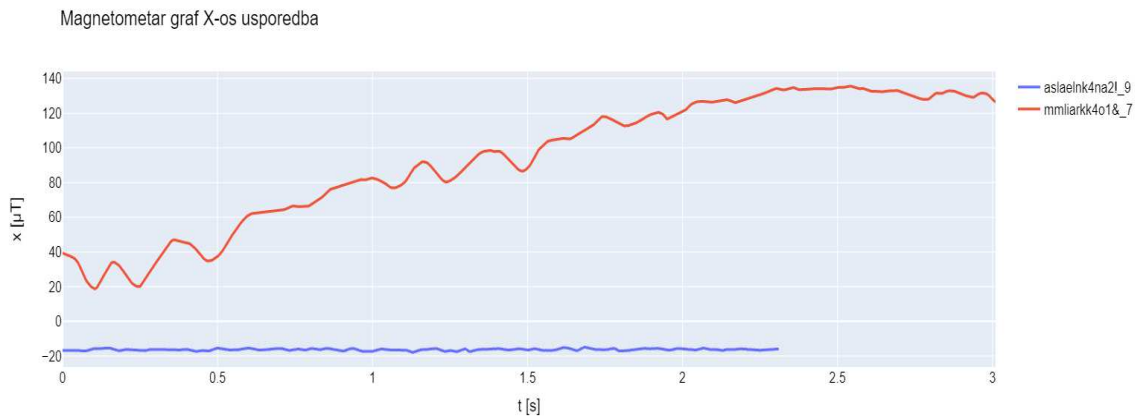


(a) Usporedba izvođenja dodirne geste (x os).

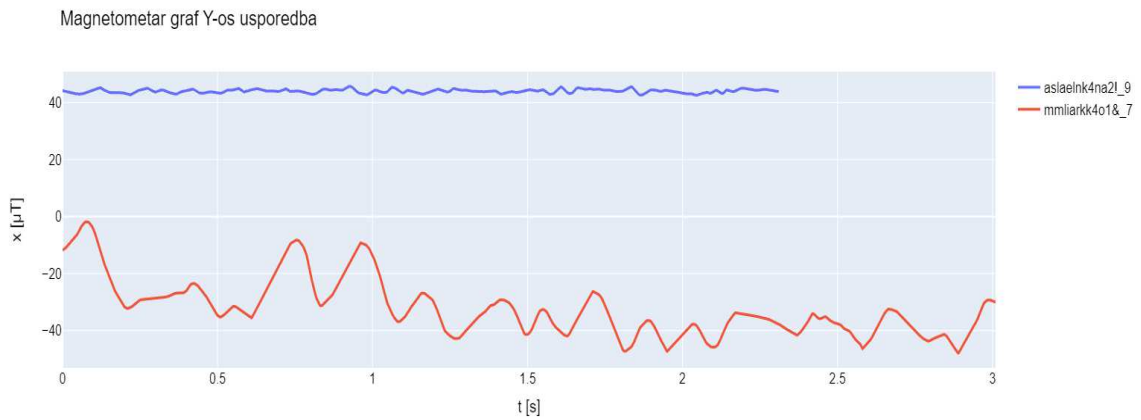


(b) Usporedba izvođenja dodirne geste (y os).

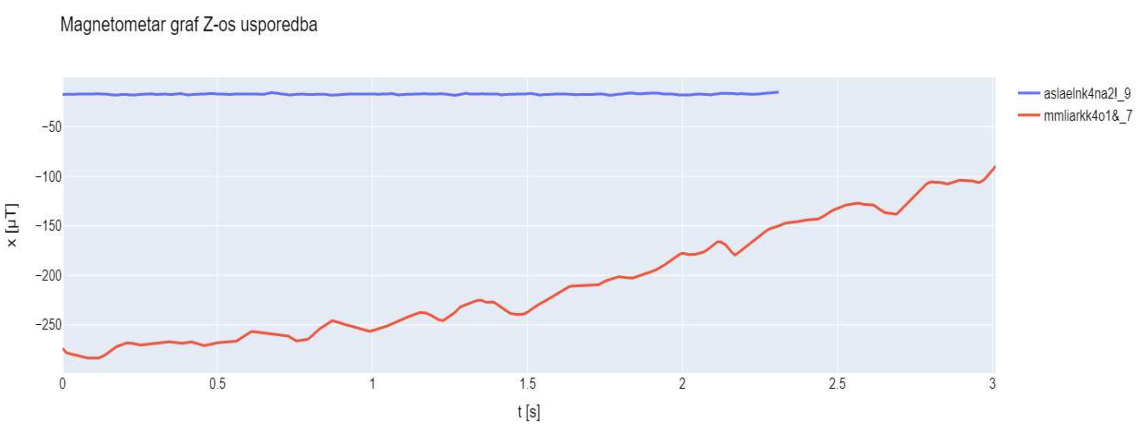
Slika 3.16 Usporedba brzine izvođenja dodirnih gesti kod potpisa korisnika *a9* i *m7*



(a) Usporedba magnetskog polja (x - koordinate).

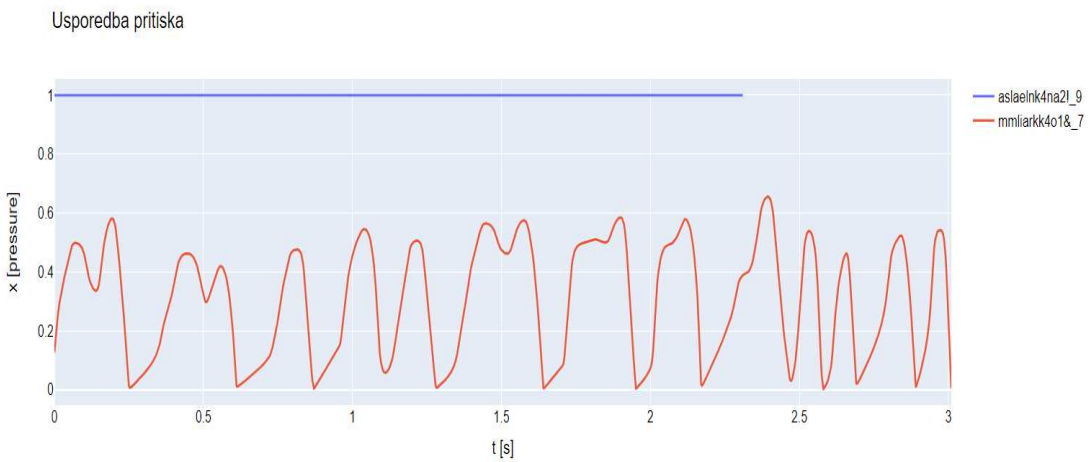


(b) Usporedba magnetskog polja (y - koordinate).



(c) Usporedba magnetskog polja (z - koordinate).

Slika 3.17 Usporedba grafova magnetskog polja očitanog pri potpisu korisnika *a9* i *m7*.



Slika 3.18 Usporedba pritiska tijekom izvođenja potpisa.

Poglavlje 4

Zaključak

Vlastoručni potpis, sa svim svojim složenim biometrijskim odrednicama, predstavlja jedan od ključnih elementa identiteta svakog pojedinca. Od posebnog je značaja i u ovom digitalnom dobu, gdje se potpis i dalje koristi kao autentifikacija i sigurnosni mehanizam.

U okviru ovog rada, cilj je bio vizualizirati biometrijske odrednice dobivene potpisom s dodirnih zaslona i stvoriti sustav za njihov prikaz i upravljanje. U radu je, demonstrirana uspješna implementacija korištenih tehnologija te je predstavljen sustav kojim je omogućeno dodavanje, uspoređivanje i filtriranje korisnika te njihovih inačica potpisa.

Uzimajući u obzir složenost biometrijskih odrednica potpisa i njihovu važnost u digitalnom svijetu, vizualizacijom tih odrednica ovaj rad daje put potencijalnoj primjeni u sigurnosnim sustavima i autentifikaciji. Nastavak istraživanja u ovom području može voditi ka razvoju naprednih sigurnosnih rješenja koja se oslanjaju na jedinstvene karakteristike vlastoručnih potpisa.

Bibliografija

[1] Quickstart, a minimal application. , s Interneta, https://flask.palletsprojects.com/en/2.3.x/quickstart/#a-minimal-application , rujan 2023.

[2] Plotly open source graphing library for python. , s Interneta, https://plotly.com/python/ , rujan 2023.

[3] Types of databases. , s Interneta, https://www.mongodb.com/databases/types , rujan 2023.

[4] A guide to horizontal vs vertical scaling. , s Interneta, https://www.mongodb.com/basics/horizontal-vs-vertical-scaling , rujan 2023.

[5] Jinja2 introduction. , s Interneta, https://jinja.palletsprojects.com/en/3.1.x/intro/ , rujan 2023.

[6] Templating basics. , s Interneta, https://docs.apitemplate.io/reference/learn-jinja2.html#templating-basics , rujan 2023.

[7] Scipy interpolate interp1d. , s Interneta, https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html , rujan 2023.

[8] B-spline curve in computer graphics. , s Interneta, https://www.geeksforgeeks.org/b-spline-curve-in-computer-graphics/ , rujan 2023.

[9] Python-b-spline-examples. , s Interneta, https://github.com/kawache/Python-B-spline-examples , rujan 2023.

[10] Plotly express in python. , s Interneta, https://plotly.com/python/plotly-express/ , rujan 2023.

[11] Line charts in python. , s Interneta, https://plotly.com/python/line-charts/ , rujan 2023.

[12] Graph objects in python. , s Interneta, https://plotly.com/python/graph-objects/ , rujan 2023.

Sažetak

Vlastoručni potpis, kada se izvodi prstom ili pokaznom napravom na dodirnom zaslonu, može biti adekvatno digitaliziran i opisan brojnim biometrijskim odrednicama. U ovome radu predstavljen je web sustav za pohranu i vizualizaciju takvih biometrijskih odrednica. Sustav omogućava uvoz novih zapisa, njihovu pohranu putem sustava za upravljanje bazama podataka te pretragu i vizualizaciju postojećih zapisa. Opisane su biometrijske odrednice vlastoručnih potpisa u standardnom zapisu koje su u sustavu vizualizirane odgovarajućim grafičkim prikazima poput vremenskog niza i trajektorije 3D vektora. Implementirana web aplikacija omogućava filtriranje sadržaja kako po individualnom korisniku tako i po pojedinoj inačici potpisa. Podržana je i mogućnost vizualne usporedbe biometrijskih odrednica potpisa različitih korisnika, kao i različitih inačica potpisa istog korisnika.

Ključne riječi **— vlastoručni potpis, dodirni zaslon, biometrija, web aplikacija, MongoDB**

Abstract

A handwritten signature made with a finger or a pointing device on a touchscreen can be suitably digitized and described by numerous biometric features. This paper presents a web system for storing and visualizing such biometric data. The system allows the import of new records, their storage by the database management system, and the search and visualization of existing records. Handwritten signature biometrics in the standard dataset are described, which are visualized within the system with appropriate graphical representations such as a time series and a 3D vector trajectory. The implemented web application allows filtering the content by individual users as well as by individual signature instances. Visual comparison of biometric signatures of different users as well as different signature instances of the same user is also supported.

Keywords **— handwritten signature, touchscreen, biometrics, web application, MongoDB**