Vizualizacija informacija u procesu zaključivanja afektivnog robota

Skočić, Lovre

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:149212

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-07**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Lovre Skočić

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
Dr.sc. Tomislav Stipančić, dipl. ing.

Student:
Lovre Skočić

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment,
inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-04/21-6/1	
Ur. broj: 15-1703-21	

DIPLOMSKI ZADATAK

Student: **LOVRE SKOČIĆ** Mat. br.: 0035203881

Naslov rada na hrvatskom jeziku: **Vizualizacija informacija u procesu zaključivanja afektivnog robota**

Naslov rada na engleskom jeziku: **Information visualisation in the process of the affective robot reasoning**

Opis zadatka:

Suvremene tehnike vizualizacije informacija unutar sučelja između čovjeka i robota omogućuju njihovu reprezentaciju tako da kontekstualno i intuitivno budu lakše shvaćene od strane ljudi. To je posebno važno kod većeg broja različitih klasa informacija gdje je presudno koristiti prikladan raspored vizualizacijskih elemenata te utvrditi prioritete kod prikaza informacija.
U radu je potrebno izraditi cjelovito rješenje za vizualizaciju informacija primijenjeno kod afektivnog robota PLEA koje će ljudskom korisniku dati uvid u unutarnje procese zaključivanja robota. Rješenje treba uključivati sljedeće korake:

1. odrediti klase informacija koje će biti predmet vizualizacije,
2. osmisliti te oblikovati korisničko sučelje za vizualizaciju,
3. osmisliti i implementirati vizualizacijske elemente,
4. povezati razvijeno rješenje s afektivnim robotom PLEA,
5. omogućiti vizualizaciju informacija u sinkronom (engl. on-line) te asinkronom (engl. off-line) modu rada.

Razvijeno rješenje potrebno je eksperimentalno verificirati na opremi dostupnoj u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
6. svibnja 2021.

Rok predaje rada:
8. srpnja 2021.

Predviđeni datum obrane:
12. srpnja do 16. srpnja 2021.

Zadatak zadao:
doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva:
prof. dr. sc. Biserka Runje

SADRŽAJ

IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem mentoru *dr.sc.* Tomislavu Stipančiću i kolegi Leonu Korenu, *mag.ing.mech.*, na savjetima, uloženom vremenu, primjedbama i korekcijama koje su pridonijele boljitku i kvaliteti ovog rada.

Hvala obitelji, prijateljima i svima koji su bili uz mene ovih 6 godina.

_____

Lovre Skočić

SAŽETAK

PLEA je afektivni robot razvijen na FSB-u, koji omogućava audio-vizualnu komunikaciju s korisnikom te putem video i audio analize prepoznaje emocije na korisniku. Prepoznaje 7 osnovnih vrsta emocija (sreća, tuga, ljutnja, iznenađenje, strah, gađenje i neutralnost) te prikazuje u postotku koliko je svaka od tih emocija dominantna u određenom trenutku. Analiza emocija se radi zasebno na temelju vizualne analize te na temelju audio analize, nakon čega se vrši multimodalna analiza i izračunava se ukupno emocionalno stanje korisnika. Rezultati svih triju modalnost spremaju se na server u bazu podataka iz koje ih je kasnije moguće dohvatiti za daljnju obradu i analizu.

Cilj ovog rada bio je prikupljene podatke učiniti što preglednijim te lakšim za korištenje i analizu. Velika količina brojčanih vrijednosti je razumljiva računalima, ali ljudima se dosta teško snaći s takvim podacima. Ljudi mnogo lakše pametne vizualne informacije, stoga je napravljena vizualizacija podataka kako bi putem grafikona bilo što jednostavnije shvaćanje dobivenih podataka. Sva zabilježena emocionalna stanja prikazana su tablicom gdje se vidi ID stanja, vrijeme kad je stanje zabilježeno, IP adresa te ime korisnika kod kojeg je stanje zabilježeno. Odabirom jednog od stanja otvara se njegov detaljan prikaz gdje je moguće odabrati modalnost koju se želi prikazati, vrstu grafikona za prikaz te je moguće vremenski pratiti i uspoređivati s ostalim zabilježenim stanjima.

Ključne riječi: Afektivni robot, emocije, vizualizacija

SUMMARY

PLEA is an affective robot developed on FSB, which enables audio-visual communication with the user and it recognizes the user's emotions using audio and visual analysis. It can recognize 7 basic emotions (happy, sad, angry, surprise, fear, disgust, and neutral) and displays in percentage how much is each emotion dominant in every moment. Emotions analysis is done separately with video and audio analysis and after that multimodal analysis is performed which gives the final emotional state of the user. Results of all three modalities are saved on the server in a database, from which it is possible to read all the data afterward for further analysis.

The purpose of this thesis was to make gathered data more user-friendly and easier for analysis. Big amount of numerical data is easily understood by computers, but humans have difficulties making sense out of them. Humans understand visual information much better, that's why information visualization is done in this thesis to display data with charts for easier understanding. All emotional states are displayed in the table with their ID, time when they were recorded, IP address and name of a user. Clicking at one of the states, a detailed view of emotional state is displayed where it is possible to choose modality which we want to see, type of a chart and it is possible to track each emotion during the time on a line graph and compare it to the other states.

Keywords: Affective robot, emotions, visualization

POPIS SLIKA

POPIS TABLICA

1. UVOD

PLEA je afektivni[1] robot napravljen na FSB[2]-u koji može percipirati emocije na osobi s kojom komunicira te ostvariti povratnu audio-vizualnu vezu s njom. U današnje vrijeme roboti su uglavnom reaktivni, tj. pogonjeni su eksplicitnim informacijama. Takvi roboti reagiraju na programirani set podražaja iz okoline, a ako se dogodi neka neplanirana situacija za koju nisu programirani, onda dolazi do neželjene situacije u kojoj se reaktivni robot neće snaći i neće znati što napraviti. Za razliku od reaktivnih, adaptivni roboti se mogu djelomično snaći u neplaniranim situacijama, ovisno o tome koliko su dobro programirani. Adaptivni roboti su roboti budućnosti, njih će biti moguće vidjeti kao pametne i u interakciji s ljudima i okolinom, razumijevajući koncepte.

Naučiti robota koncepte stvarnog života i shvaćanje konteksta je kompliciran posao. Ljudi zaključuju implicitno (kontekstualno), a računala eksplicitno, tj. razumiju samo 0 ili 1. Kako bi omogućili računalima da stavljaju informacije u određeni kontekst, potrebno je napraviti prevoditelja koji je prevoditi implicitno u eksplicitno.

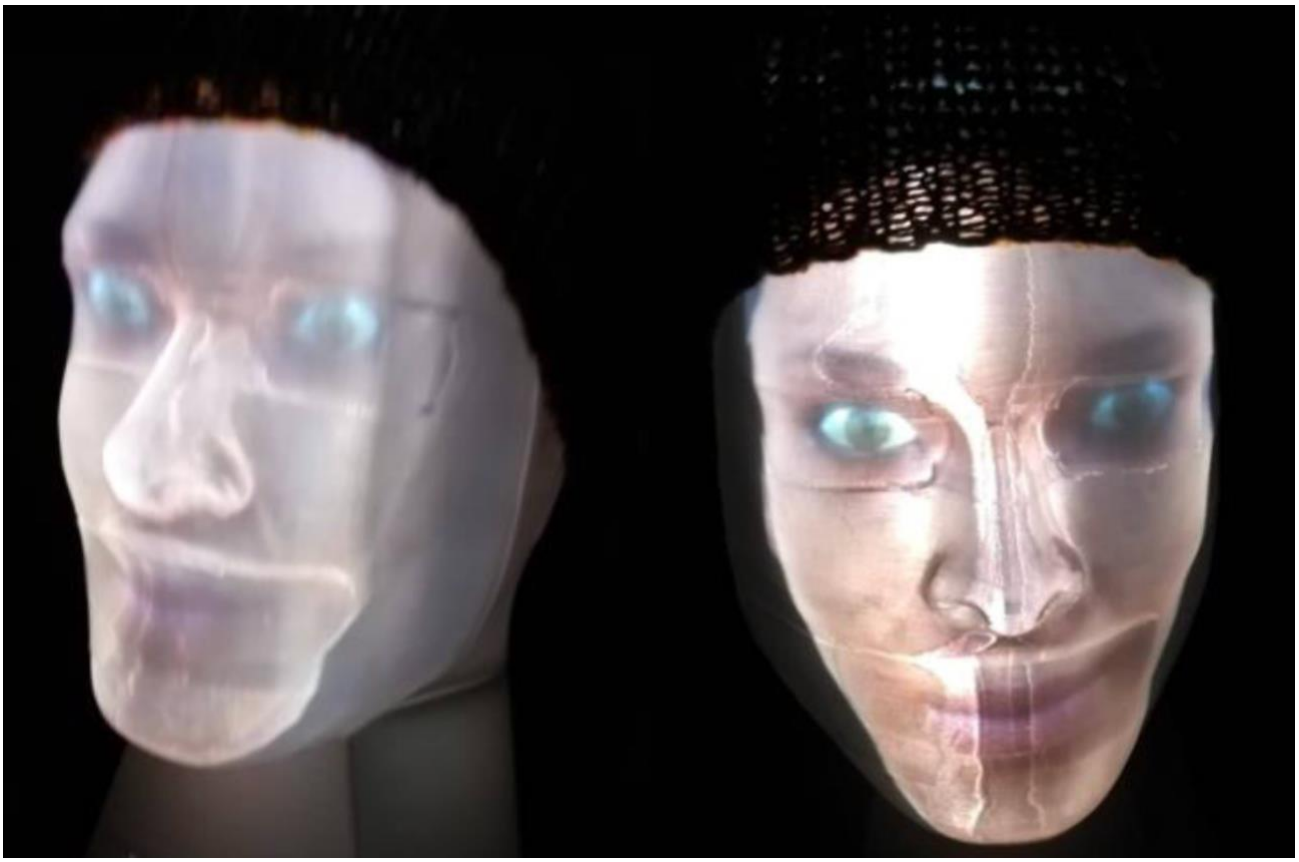
PLEA radi na tele operiranom pristupu *'Wizard of Oz',* u kojem nastavnik upravlja s robotom s udaljenog mjesta, a korisnik ima dojam da je robot stvarno živ jer ne vidi osobu koja upravlja s njim. PLEA je smještena zajedno s korisnikom (studentima) u jednoj prostoriji, dok se nastavnik može nalaziti u svom uredu ili nekoj drugoj udaljenoj prostoriji te upravlja robotom i komunicira s korisnicima iz te prostorije.

Opremljena je s kamerom i mikrofonima koji proučavaju korisnika i prostor oko njega, prikupljaju neverbalne i verbalne komunikacijske znakove koji se potom analiziraju u računalu multimodalnom analizom te se na temelju njih donosi zaključak o emocionalnom stanju studenta. Preko ugrađenih mikrofona PLEA također locira i izvor zvuka te se uvijek okreće prema njemu kako bi uspostavila kontakt s očima. U nju su ugrađeni i zvučnici koji omogućuju slanje glasovnih poruka, tj. audio komunikaciju.

[1] Afektivno računarstvo je studija i razvoj sustava i uređaja koji mogu prepoznati, interpretirati, procesirati i simulirati ljudske emocije.

[2] Fakultet strojarstva i brodogradnje

Kamera koja se nalazi na računala nastavnika koji upravlja PLEA-om, snima lice nastavnika te prepoznaje emocije na njemu. Na temelju prepoznatih emocija se zatim radi avatar koji imitira emocije nastavnika. Avatar se putem projektora i zrcala prikazuje na 3D licu robota te se dobiva dojam stvarne osobe kojoj se mijenja raspoloženje. Slikom 1.1 prikazan je avatar na PLEA-i.



Slika 1.1 Prikaz avatara na PLEA-i

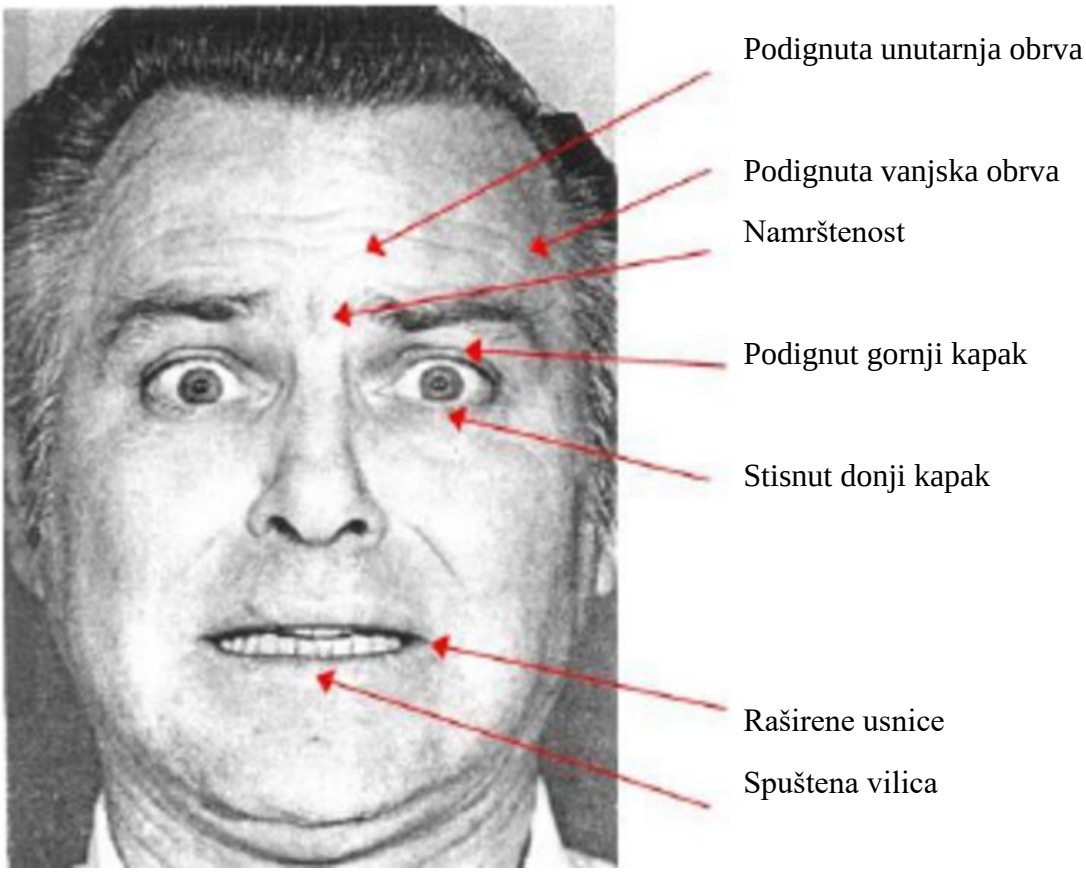
Multimodalne interakcije:

1. Vizualno čitanje emocija s lica:

   Sustav temeljen na FACS[3] i konvolucijskog neuronskoj mreži. FACS je izumio američki psiholog Paul Ekman, jedan od pionira u proučavanju ljudskih emocija. On je napravio atlas lica tako da je našao karakteristične točke na licu, tj. mišiće koji mogu biti opušteni ili stisnuti. Različite kombinacije stanja mišića generiraju grimasu, odnosno prikazuju emocije. Zatim se koristi konvolucijska neuronska mreža koja nauči

[3] Facial Action Coding System (sustav za označavanje facijalnih ekspresija)

i poveže te izraze lica s oznakama emocija. Primjer FACS analize prikazan je slikom 1.2.



Slika 1.2 FACS analiza izraza lica

2. Auditivna procjena:

   Mikrofonom se očitava razina buke u prostoriji i intonacija korisnika, na temelju kojih algoritam procjenjuje koliko je jaka svaka od emocija.

U algoritmu za multimodalnu analizu obrađuju se vizualno i auditivno stanje te se stapaju u jednu i algoritam izračunava ukupno emocionalno stanje za određeni trenutak. Trenutno se analiza emocija vrši samo na temelju audio i video analize, no u budućnosti je planirano proširenje i na neke kompleksnije analize.

Za vrijeme dok korisnik komunicira s robotom, svaku sekundu se bilježi emocionalno stanje prepoznatih emocija te se u bazu podataka spremaju vrijednosti dobivene audio i video analizom te ukupno stanje nakon provedene multimodalne analize. Kako bi prikupljeni podaci bili jednostavniji za razumijevanje ljudima, koji lakše uče i pamte vizualne informacije, u ovom radu napravljena je vizualizacija brojčanih podataka koje prikuplja PLEA. Sva zabilježena stanja prikazana su tablicom u kojoj su vidljivi osnovni podaci svakog zabilježenog trenutka, a klikom na jedno od zabilježenih stanja otvara se stranica na kojoj je dan detaljan prikaz stanja gdje su svi podaci prikazani u obliku histograma, tortnog grafikona i polarnog grafikona. Kako bi bilo moguće lakše pratiti promjene kroz vrijeme također je dan prikaz linijskim grafom koji prikazuje kako su se mijenjale pojedine emocije.

2. VIZUALIZACIJA PODATAKA

Vizualizacija informacija je novonastala disciplina koja koristi vizualne prikaze apstraktnih podataka kako bi omogućila ljudima lakše razumijevanje informacija i njihovo lakše pamćenje. Promatranje vizualnih podataka omogućuje ljudima jednostavnije poimanje poruke nego čitanje teksta ili brojčanog prikaza. Vizualizacija je sposobnost da se nešto predoči ili zamisli u slikama. Kako u današnjem svijetu ljudi primaju velike količine raznih informacija svakoga dana, važno je da informacije dođu u pravom obliku kako bi bile što efikasnije.

Sva živa bića većinu podražaja, odnosno primanja informacija iz okoline vrše pomoću osjetila. Ljudi imaju 5 osjetila – vid, njuh, sluh, opip i okus, ali nisu sva osjetila jednako razvijena i ne primamo jednaku količinu informacija putem svakog od njih. Čak 83 % informacija koje ljudski mozak dobije dolazi iz vida, 11 % informacija pruža sluh, njuh 3.5 %, opip 1.5 %, a dodir samo 1 %. Tako primjerice na otvorenom polju prosječan čovjek može vidjeti 80 km daleko, čuti oko 2.5 km, namirisati nešto na 10-20 m, a opip i okus su limitirani na domet ruke odnosno jezika [1].

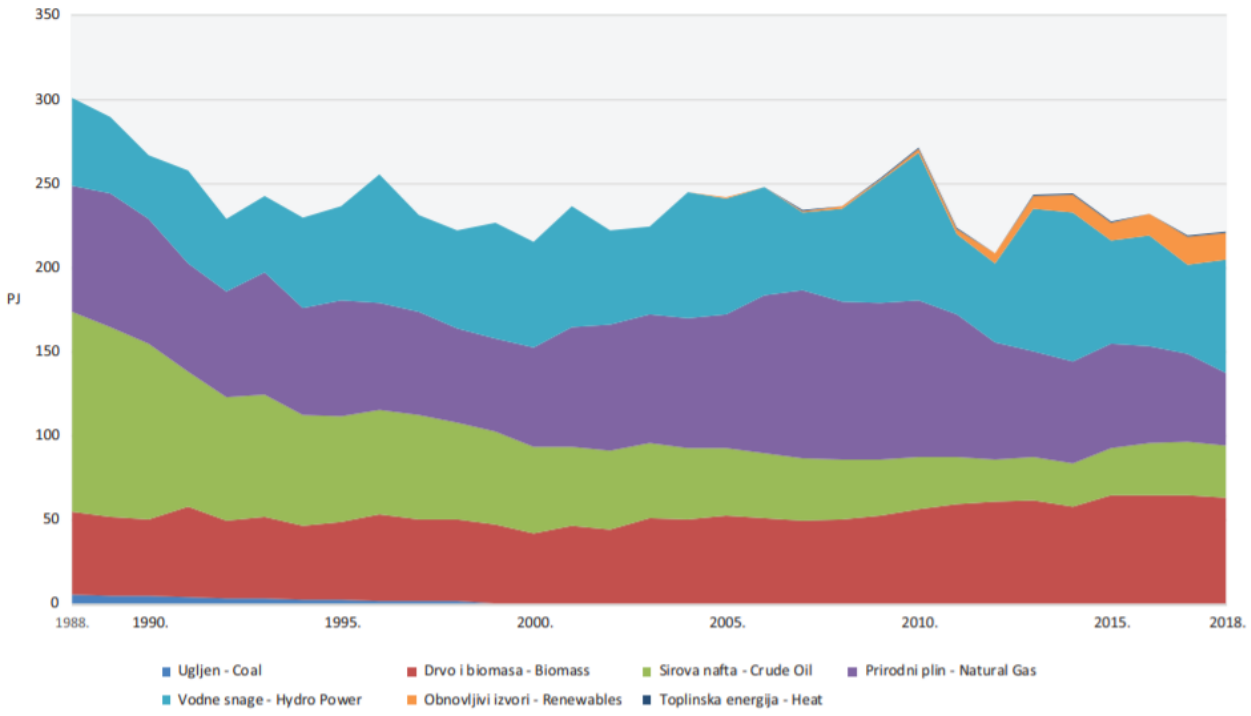
Zbog toga što vizualnim putem primamo najviše podataka iz okoline, korisno je koristiti se vizualizacijom podataka koja omogućuje primanje, analizu i obradu velike količine podataka i informacija u relativno kratkom vremenu. Također, vizualizacija podataka omogućuje jednostavnije primjećivanje greške i odstupanja u uzorcima te je na taj način moguće uočiti koji podaci su ključni odnosno kritični. Njena uloga može biti i pružanje pregleda određene skupine podataka i pretvaranje tih podataka u informacije na temelju kojih se donosi neki zaključak ili se definira novi problem na koji je potrebno pronaći odgovor. Vizualizacijom je moguće provjeriti i kvalitetu podataka ili procesa, ako neki podaci značajno odstupaju od ostalih moguće je da se dogodila greška prilikom mjerenja ili unosa podataka te je takve podatke potrebno provjeriti [2].

Za usporedbu razumijevanja tekstualnih i vizualnih informaciji bit će prikazna proizvodnja primarne energije u Hrvatskoj. Prikazat će se proizvodnja po pojedinim izvorima energije od 1988. do 2018. Radi pojednostavljenja u tekstualnom opisu neće biti objašnjene sve godine kako bi se smanjila količina podataka. Proizvodnja energije prikaza je u petadžulima (PJ), što iznosi 10^{15} džula.

1988. godine proizvedeno je ukupno 307 PJ energije, od čega je 4 PJ dobiveno iz ugljena, 49 PJ iz drva i biomase, 123 PJ iz sirove nafte, 79 PJ iz prirodnog plina i 52 PJ iz snage vode. 1994. godine proizvodnja je pala na 226 PJ, od čega je 1 PJ dobiven iz ugljena, 47 PJ iz drva i biomase, 62 PJ iz

sirove nafte, 65 PJ iz prirodnog plina i 51 PJ iz snage vode. 2010. godine proizvodnja je iznosila 271 PJ, od čega je 56 PJ dobiveno iz drva i biomase, 31 PJ iz sirove nafte, 94 PJ iz prirodnog plina, 87 PJ iz snage vode i 3 PJ iz ostalih obnovljivih izvora energije. 2014. godine ukupno je proizvedeno 244 PJ energije, od čega je 58 PJ dobiveno iz drva i biomase, 25 PJ iz sirove nafte, 61 PJ iz prirodnog plina, 89 PJ iz snage vode i 11 PJ iz ostalih obnovljivih izvora energije. 2018. godine proizvedeno je ukupno 221 PJ energije, od čega je 63 PJ dobiveno iz drva i biomase, 31 PJ iz sirove nafte, 43 PJ iz prirodnog plina, 67 PJ iz snage vode i 17 PJ iz ostalih obnovljivih izvora energije [3].

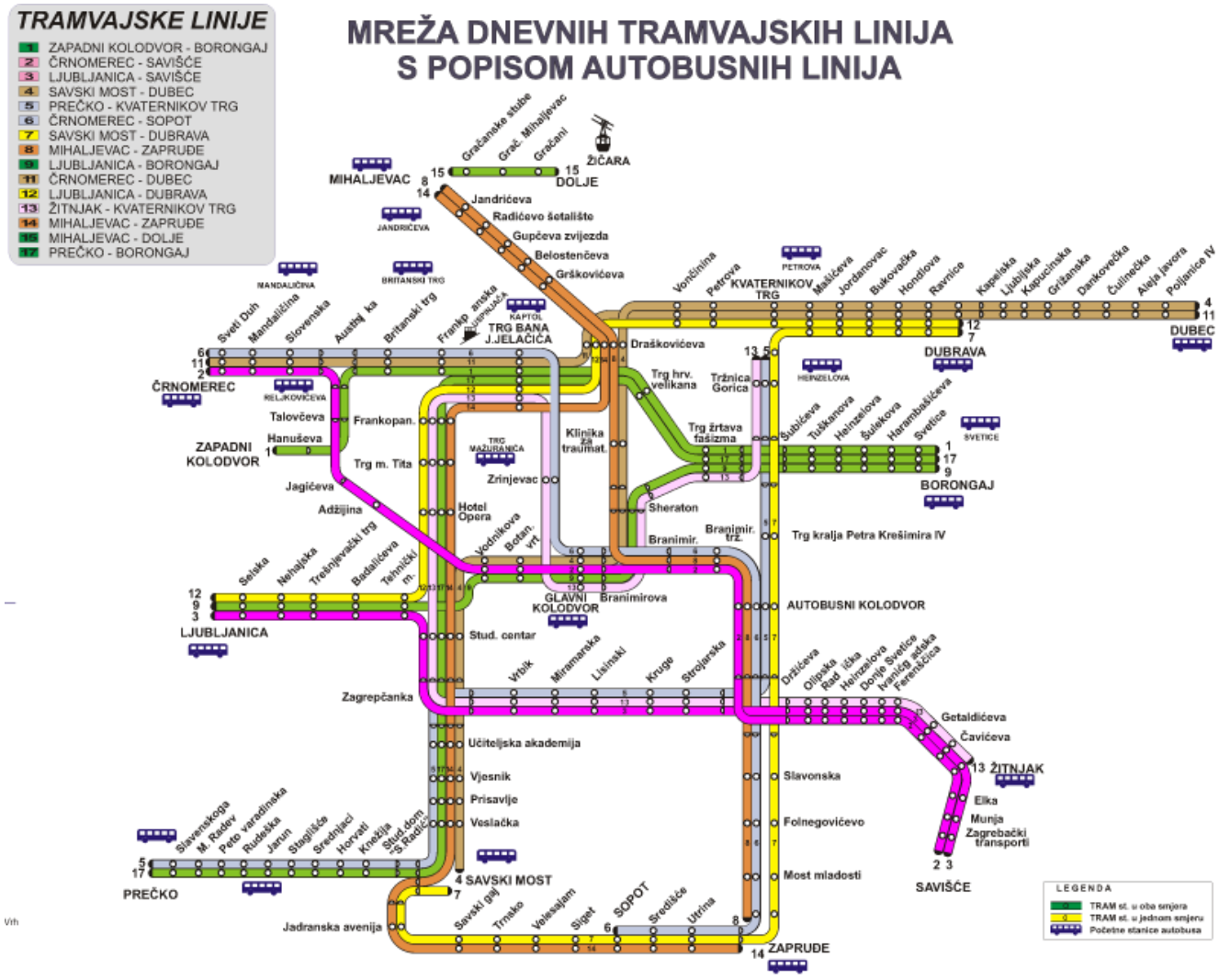
Grafički prikaz proizvodnje energije prikazan je slikom 2.1. Kao što je moguće vidjeti, grafički prikaz je mnogo pregledniji te lakši za razumijevanje i pronalaženje podataka koji su potrebni. Tekstualni opis je jako zamoran i teško se snaći u šumi napisanih brojeva, zato je puno bolje koristiti se vizualnim prikazima kad god je to moguće. Primjerice, da želite pronaći ukupnu dobivenu energiju iz drva i biomase 2010. godine i usporediti ju s energijom dobivenom iz drva i biomase 2018. godine, koliko se lakše snaći na grafu u odnosu na traženje podataka u opisnom zapisu? Naročito kada se uzme u obzir da su zbog pojednostavljenja u brojčanom zapisu napisane vrijednosti samo za 5 godina od 30 koliko ih je ukupno prikazano na grafu.



Slika 2.1 Proizvodnja primarne energije u Hrvatskoj 1988.-2018. [3]

Vizualizacija podataka direktno utječe na svakodnevni ljudski život. Ljudi se svakodnevno moraju snalaziti u raznim podacima koje primaju iz okoline, a njihovo vizualiziranje ubrzava i olakšava taj proces. Primjerice, tramvajske mape znatno olakšavaju traženje određene rute nego da su samo napisane sve stanice za svaku liniju, kreiranje tabličnih rasporeda pruža korisniku sve potrebne informacije samo iz jednog pogleda. Zbog toga, primjereno prikazivanje podataka omogućuje ljudima znatno efikasnije i brže korištenje tih podataka.

Vizualizirani podaci ne trebaju biti kompleksni, već ih je bolje što više pojednostavniti. Radi lakšeg snalaženja i pronalaženja bitnih informacija potrebno je izbaciti sve suvišne informacije s vizualnom prikaza. U prošlosti su se karte metro mreže crtale preko obične karte grada, tako su na njima bile prikazane i ceste, parkovi i mnogo drugih informacija koje su nepotrebne na karti metroa i samo otežavaju snalaženje. Zato su s modernih karti mreža metroa, tramvaja i sličnih uklonjeni svi nepotrebni podaci i same linije su najčešće prikazane horizontalno i vodoravno, a ne zakrivljenim oblicima kojima stvarno prometuju. Slikom 2.2 je prikazana mreža dnevnih tramvajskih linija u Zagrebu.



Slika 2.2 Mreža dnevnih tramvajskih linija u Zagrebu [4]

2.1 Vizualizacija u učenju

Kyvete Shatri i Kastriot Buza proveli su istraživanje o efikasnosti vizualnog učenja u odnosu na klasično učenje bez vizualnih grafika. Eksperiment je proveden na 100 studenata, od čega je 50 činilo kontrolnu skupinu, a 50 eksperimentalnu skupinu. Studenti su birani na temelju prosjeka tako da svi budu sličnih sposobnosti i predznanja. Obje skupine trebale su riješiti preliminarni test prije predavanja kako bi se utvrdilo njihovo predznanje. Zatim je kontrolna skupina imala predavanje bez vizualizacije podataka, dok je eksperimentalna skupina imala predavanje koristeći vizualizaciju pomoću raznih grafika. Nakon održanih predavanja svim studentima je dan završni test da se vidi

koliko su naučili na predavanju. Rezultati preliminarnog testa prikazani su u tablici 3.1, a rezultati završnog testa prikazani su u tablici 3.2 [5].

Tablica 2.1 Rezultati preliminarnog testa [5]

Kontrolna skupina			Eksperimentalna skupina		
Broj bodova	Broj studenata	Postotak	Broj bodova	Broj studenata	Postotak
0-50	10	20 %	0-50	11	22 %
51-70	25	50 %	51-70	26	52 %
71-100	15	30 %	71-100	13	26 %

Tablica 2.2 Rezultati završnog testa [5]

Kontrolna skupina			Eksperimentalna skupina		
Broj bodova	Broj studenata	Postotak	Broj bodova	Broj studenata	Postotak
0-50	14	28 %	0-50	5	10 %
51-70	26	52 %	51-70	30	60 %
71-100	10	20 %	71-100	15	30 %

Rezultati preliminarnog testa pokazuju da su u obje skupine bili studenti s približno jednakim znanjem, a rezultati završnog testa pokazuju pozitivnu stranu učenja s vizualizacijom i njen odraz na uspješnost učenja. Tema predavanja i testova je bio polimorfizam iz područja biologije.

To je razlog zašto postoji izreka 'slika govori tisući riječi'. Ljudi puno lakše procesuiraju vizualne informacije i na taj način mogu u kraćem vremenu primiti puno veću količinu podataka nego čitanjem teksta. Zato je od iznimnog značaja koristiti grafike i vizualizirati podatke prilikom učenja.

Dok uče, ljudi u prosjeku zapamte samo 10 % onoga što pročitaju, 20 % onoga što čuju, 30 % onoga što vide, 50 % onoga što čuju i vide, 70 % onoga što kažu i napišu te 90 % onoga što učine. Slike i ilustracije su isječci iskustva (analog of experience) i nalikuju na stvarne događaje, zato ih ljudski mozak bolje pamti od čitanja teksta [5]. Zato je, osim prakse, vizualizacija podataka ključ uspješnijeg učenja.

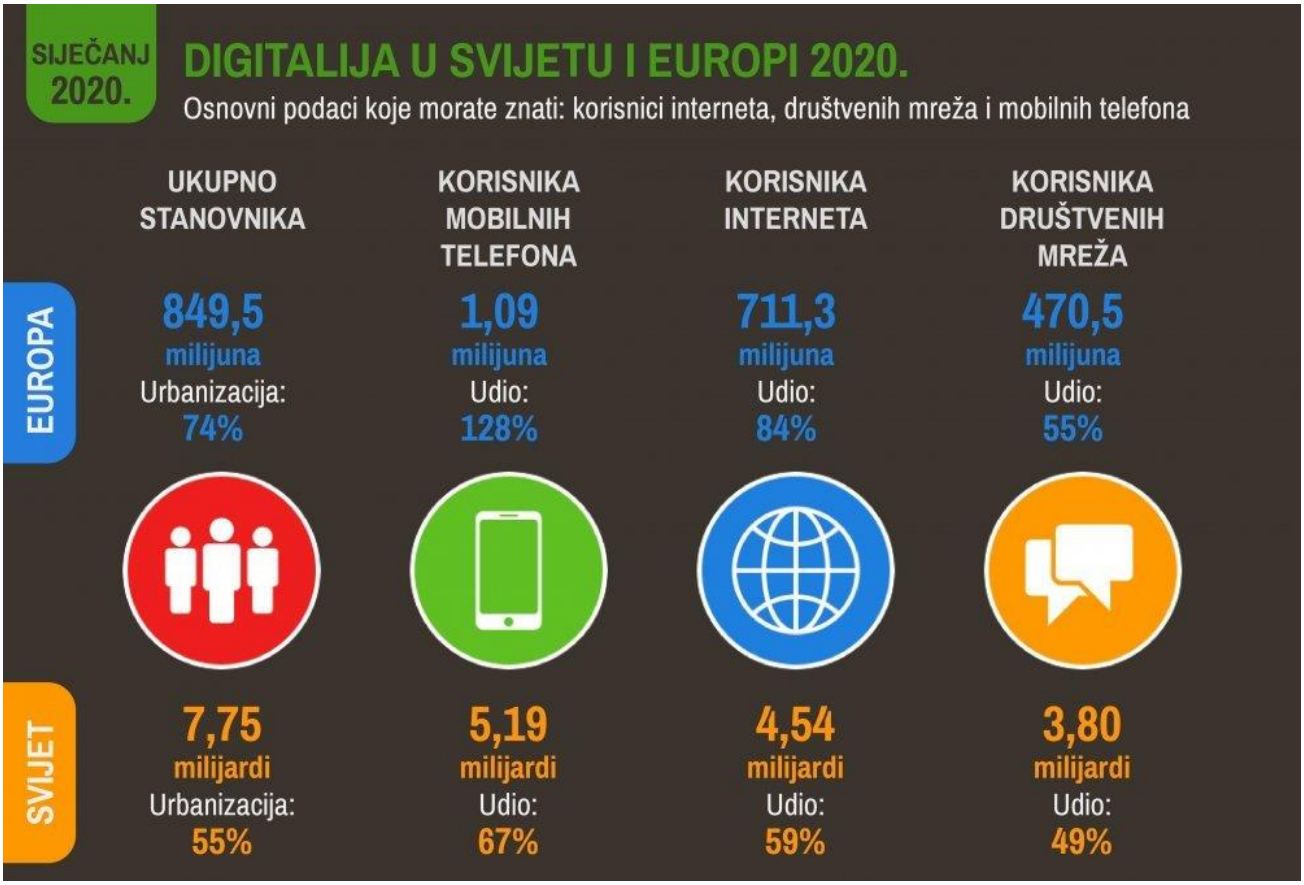
3. WEB PROGRAMIRANJE

Internet je globalni sustav koji povezuje računala i računalne mreže te omogućuje njihovu međusobnu komunikaciju. To je mreža koja se sastoji od milijuna manjih mreža kao što su kućne mreže, vladine, poslovne, akademske i druge. Početci interneta su u 1960-ima, kada je Američko Ministarstvo obrane razvijalo sustav za povezivanje vojnih računala u SAD-u. Internet se tek krajem 80-ih počeo razvijati za komercijalne svrhe, a glavnu ulogu u njegovu razvitku je imao Tim Berners-Lee koji je početkom 1990-ih napravio prvi internetski preglednik WorldWideWeb. Tim je također razvio alate koji su nužni za rad weba, HTTP[4] i HTML[5].

U današnje vrijeme preko 60 % ljudi na svijetu koristi internet, i taj broj iz dana u dan ubrzano raste. Internet svakog dana ima u prosjeku 900 000 novih korisnika. Dok se u prošlosti na internet povezivalo gotovo isključivo putem računala, zadnjih 10 godinama taj trend se sasvim promijenio te danas preko 90 % korisnika interneta uz računalno koristi i mobitel za spajanje na internet. Broj korisnika interneta i mobilnih telefona u svijetu i Europi 2020. godine prikazan je slikom 3.1 [6].

[4] Hyper Text Transfer Protocol (protokol za prijenos hiperteksta) – protokol koji omogućuje objavljivanje i prikaz hiperteksta na web stranicama.

[5] Hyper Text Markup Language (prezentacijski jezik hiperteksta) – programski jezik za oblikovanje hiperteksta i definiranje njegovog prikaza na web stranicama.



Slika 3.1 Digitalizacija u svijetu i Europi [7]

Web stranice imaju korisničku razinu[6] i razinu za pristup podacima[7]. Korisnička razina služi za prikaz podataka na web stranici i omogućuje korisniku interakciju s njima. Osnovni jezik kojim se definira struktura stranice i oblikuje sadržaj na njoj je HTML. Dva osnovna dijela stranice su uzglavlje i tijelo. U uzglavlju se definira pristup vanjskim datotekama, naslov stranice te metapodaci poput ključnih riječi, znakovnog pretvornika, autora, opisa, i slično. Tijelo je glavni dio stranice u kojem je definiran sav sadržaj koji se prikazuje, a sadržaj može biti podijeljen u razne strukturne elemente, paragrafe, tablice, i slično. Izgled sadržaja se oblikuje koristeći CSS[8] jezik. Elementima na stranici se mogu dodavati različite klase i oznake, kojima se kasnije pomoću CSS-a definiraju karakteristika poput boje, veličine fonta, pozadine, margina i drugih. Funkcionalnost na stranici se programiraju pomoću jezika JavaScript (JS). JavaScriptom se može pristupiti raznim elementima na stranici i mijenjati njihov

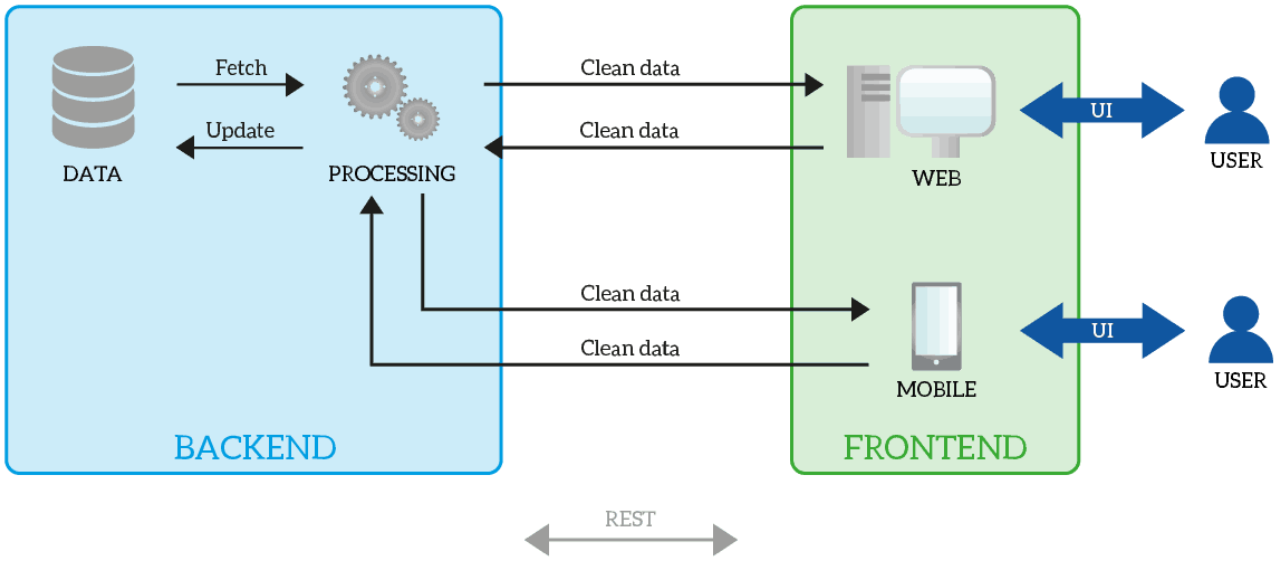
[6] Eng. Front-end

[7] Eng. Back-end

[8] Cascading Style Sheets (kaskadni stilski predložak)

sadržaj, kreirati vremenske događaje ili događaje koji se događaju primjerice na klik miša, računati matematičke i logičke operacije te pristupati podacima iz lokalnih ili vanjskih izvora.

Razina za pristup podacima osigurava osnovnu računalnu logiku web stranice. Na njoj se u pozadini definiraju postavke stranice, osnovna logika stranice, obrađuju podaci, definira se koja će se stranica prikazati na korisničkoj razini te koji će joj podaci biti poslani. Pristupna razina se često povezuje s bazom podataka te omoguće dodavanje podataka na nju i čitanje podataka koji se kasnije šalju na samu web stranicu. Jezici koji se najčešće koriste za ovu razinu su .Net, Python, Ruby, PHP i Java. Prikaz komunikacije korisničke razine i razine za pristup podacima prikazan je slikom 3.2.



Slika 3.2 Komunikacija korisničke i pristupne razine [8]

U ovom radu neće se dublje ulaziti u samo web programiranje, ali u nastavku će biti objašnjeno što su to SVG, D3.js i Django, koji su neophodni za ovaj rad.

3.1 SVG

SVG[9] je element koji se koristi za vektorsko definiranje grafike na webu. SVG-om se mogu jednostavno kreirati razni geometrijski oblici, grafikona, dijagrami, slike i slično. U ovom radu SVG će biti osnova za izradu svih grafikona.

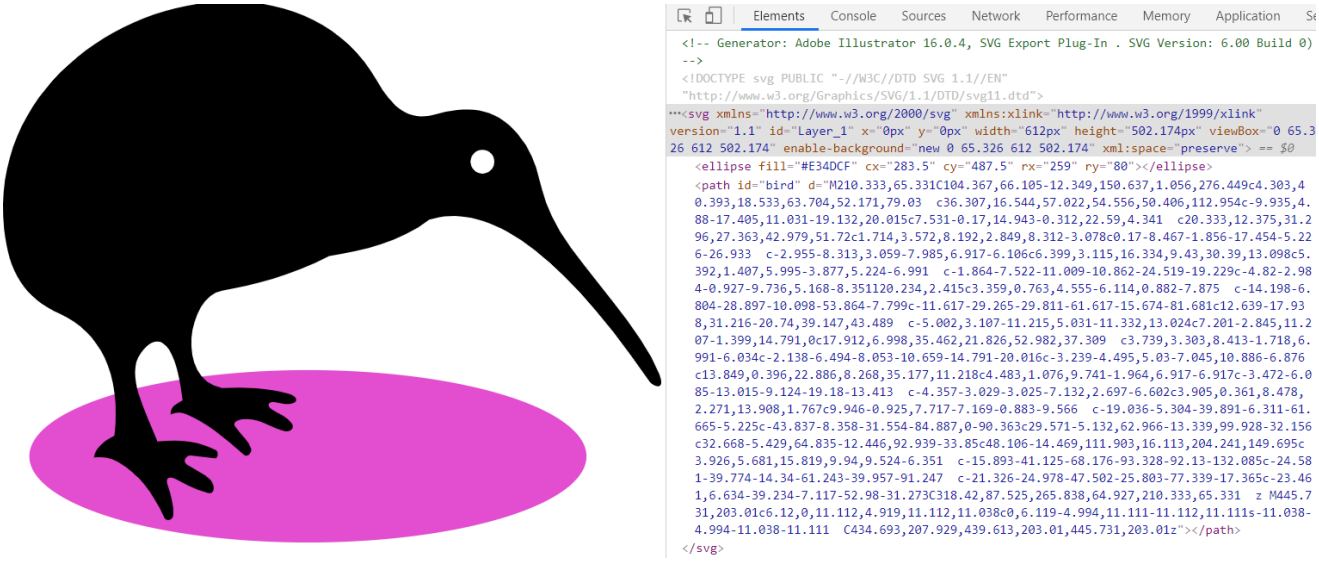
Prednosti korištenja slika baziranih na SVG elementima [9]:

- Skalabilne su, tj. ne gube kvalitetu kada ih se poveća ili zumira.
- Moguće ih je animirati koristeći JavaScript.
- Mogu se printati na bilo kojoj rezoluciji.
- Moguće ih je indeksirati za tražilice.
- Moguće ih je mijenjati kako se mijenjaju podaci.

SVG slike nisu zapravo slike, već set koordinata koji definiraju konturu slike i boje kojima će oblici biti ispunjeni. SVG definira vektorsku grafiku u XML[10] formatu. Iz tog razloga SVG format karakterizira puno manja veličina datoteke u odnosu na klasične jpg ili png formate te stoga zauzimaju manje prostora i omogućavaju brže učitavanje stranice. No ukoliko se radi o SVG slici s kompliciranom geometrijom, njeno generiranje je nešto kompleksnije te se stoga gubi spomenuta prednost bržeg učitavanja. Primjer SVG slike i njenog koda kojim je generirana prikazan je slikom 3.3.

[9] Scalable Vector Graphics (skalabilna vektorska grafika)

[10] Extensible Markup Language (jezik za označavanje podataka)



Slika 3.3 SVG slika i njen kod

Svi kompleksi SVG elementi kreiraju se kombiniranjem osnovnih elemenata. Neki od osnovnih elemenata i parametri koji im se definiraju su [10]:

- Pravokutnik – Potrebno je definirati 4 osnovna parametra (poziciju gornjeg lijevog kuta pravokutnika na x i y osi, širinu i visinu) te je moguće definirati 2 dodatna (radijusi zaobljenosti vrhova).
- Kružnica – Potrebno je definirati radijus te udaljenost centra kružnice na x i y osi.
- Elipsa – Potrebno je definirati radijus po x i y osi te udaljenost centra elipse na x i y osi.
- Linija – Potrebno je definirati x i y koordinate početne i krajnje točke.
- Putanja – Najčešće korišteni element i njime se mogu kreirati i svi jednostavniji elementi. Može se koristiti za generiranje linija, krivulja, lukova i ostalih elemenata. Najčešće ima jako puno ulaznih parametara poput koordinata i načina na koji su te koordinate povezane (linija, radijus...).

Osim navedenih parametara, pri kreiranju svim elementima se mogu dodati i atributi poput boje linije, boje ispune oblika, prozirnosti, debljine linije i ostalih. Slikom 3.4 prikazano je generiranje jednostavne kružnice. Prvo je kreiran SVG element visine i širine 120 piksela sa zelenom pozadinom, zatim je u njemu kreirana kružnica kojoj je središte udaljeno 50 piksela od gornjeg i 50 piksela od lijevog ruba SVG elementa, radijusa 40 piksela, obrubljena crnom linijom širine 3 piksela i ispunjena crvenom bojom.

```
<!DOCTYPE html>
<html>
<body>

<svg height="120" width="120"
style="background-color: green">
  <circle cx="50" cy="50" r="40"
stroke="black" stroke-width="3" fill="red" />
</svg>

</body>
</html>
```

Slika 3.4 Generiranje SVG kružnice

3.2 D3.js

D3.js[11] je JavaScript biblioteka kojoj je glavna svrha stvaranje interaktivnih i responzivnih dvodimenzionalnih vizualizacijskih objekata koristeći HTML, CSS, JavaScript i SVG elemente. D3.js omogućava kreiranje novih elemente te odabir već postojećih i njihovo modificiranje, translatiranje i animiranje. Moguće ju je povezati s lokalnom ili vanjskom bazom podataka te omogućiti mijenjanje sadržaja u realnom vremenu kako se mijenjaju ulazni podaci. Primjerice, korištenjem D3.js biblioteke moguće je iz skupa podataka kreirati HTML tablicu ili SVG grafikon.

Svaka operacija s D3.js bibliotekom započinje odabirom elementa na kojem ćemo vršiti promjene. Odabir se može vršiti po imenu klase, oznaci, id-u ili atributu, a odabrati se može prvi element s traženom karakteristikom koristeći funkciju *select* ili svi elementi s traženom karakteristikom koristeći funkciju *selectAll*. Na odabranim elementima zatim se mogu kreirati novi elementi poput SVG kružnica, linija i ostalih te im je moguće dodjeljivati razne atribute i translatirati ih. Također, moguće je kreirati funkcije koje se pokreću određenim događajima, primjerice klikom miša na određeno polje ili prelaskom miša preko određenog elementa, te je tako omogućeno korisniku da vrši

[11] Data-driven Documents (dokumenti pogonjeni podacima)

interakciju u kojoj se određeni atributi ili cijeli elementi tečno mijenjaju kroz vrijeme animirajući prikaz.

Kao ulazne podatke D3.js može prihvatiti nizove, objekte i tekstualni niz[12], kao i velike skupine podataka u JSON[13] formatu ili CSV[14] datoteci. JSON je standardni format za zapis objekata i njihovo jednostavno prenošenje, a koristi se u gotovo svim web aplikacijama pri komunikaciji sa serverom. Može sadržavati jedan objekt ili niz objekata iz kojih je kasnije jednostavno izvući željene podatke i dalje ih procesuirati. Kao što samo ime govori, CSV je format u kojem su vrijednosti odvojene zarezom. Svaki redak predstavlja jednu stavku koja može imati jedno ili više polja koja su odvojena zarezom. U CSV najčešće se spremaju tablični podaci koje je zatim jednostavno slati i čitati.

Velika prednost D3.js-a u odnosu na druge vizualizacijske alate je kompatibilnost. D3.js ne koristi zasebni rječnik, već on dolazi iz ostalih web standarda: HTML-a, CSS-a i SVG-a. To omogućava da se svi elementi kreirani pomoću D3.js-a mogu kasnije uređivati vanjskim datotekama i drugim softverima što olakšava pristupačnost i efikasnost. Tako je moguće ukomponirati i nove tehnologije u već postojeće elemente bez potrebe za mijenjanjem kompletnog koda. Također, D3.js ima jednostavan način otkrivanja pogrešaka u kodu jer je moguće koristiti JavaScript konzolu koja je integrirana u svaki web preglednik i njome pregledavati kako se izvršavao kod [11].

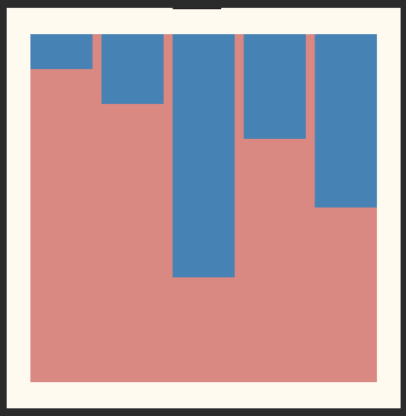
D3.js je pogodan za obradu veće količine podataka koja dolazi u obliku niza. Postoji implementirana funkcija koja omogućuje izvršavanje određenih funkcija na svakom od unesenih elemenata u nizu. Odabere se ulazni set podataka te se za određeni atribut može kreirati funkcija koja ima parametre vrijednost elementa u nizu i njegov indeks. Na taj način se mogu kreirati razni prikazi koji ovise o veličini ulaznih podataka i njegovoj poziciji u nizu. Slikom 3.5 dan je jednostavan primjer korištenja funkcije u kojoj ulazne podatke čini niz od 5 vrijednosti te se zatim na SVG elementu kreiraju pravokutnici za svaku vrijednost ulaznog niza čija visina ovisi o vrijednosti ulaznog podatka.

[12] Eng. string

[13] JavaScript Object Notation (JavaScript zapis objekata)

[14] Comma-Separated Values (vrijednosti odvojene zarezom)

```
var ulaz = [1, 2, 7, 3, 5];

var svg = d3.select("#lineDiv")
    .append("svg")
        .attr("height", "200")
        .attr("width", "195")
        .attr("style", "background-color:#d98882")
        .attr("id", "mojSVG");

svg.selectAll("#mojSVG")
        .data(ulaz)
        .enter()
        .append("rect")
        .attr("x", function(d, i) {return i*40;})
        .attr("y", 0)
        .attr("width", 35)
        .attr("height", function(d) {return d*20;})
        .attr("fill", "steelblue");
```



Slika 3.5 D3.js funkcija ulaznih podataka

3.3 Django

Django je radna okolina[15] za razvoj web stranica bazirana na Python jeziku, koja omogućava brzi razvoj sigurnih web stranica koje su jednostavne za održavanje. Djnago ima MVC[16] strukturu gdje su modeli prikazani s Django modelima koji su povezani s bazom podataka, predlošci[17] služe za prikaz, a kontroleri definiraju koji se predlošci otvaraju te koji modeli se s njima učitavaju. Dispečer URL[18] poveznica definira na kojoj se adresi koji kontroleri aktiviraju.

[15] Eng. Framework

[16] Model, View, Controller (Model, prikaz, kontroler)

[17] Eng. Template

[18] Uniform Resource Locator (jedinstveni lokator sadržaja) – jedinstvena poveznica koja na webu adresu definira adresu sadržaja

Odlike Djanga [12]:

- Brzina – Django je konstruiran tako da omoguće razvoj koncepta do funkcionalne stranice jako brzo. Kaže se da Django koriste perfekcionisti s vremenskim ograničenjem.
- Potpunost – Omogućuje razvoj korisničke razine i razine za pristup podacima te ih povezuje s bazom podataka.
- Sigurnost – Automatski se implementiraju sigurnosni mehanizmi koji pomaže programerima da izbjegnu mnoge česte pogreške koje hakeri mogu zloupotrijebiti.
- Skalabilnost – Django je prikladan za razvoj manjih stranica ali se vrlo lako može nadograditi i podržavati jako veliki stranice jer koristi strukturu u kojoj su svi dijelovi stranice odvojeni te ih je lako zamijeniti ili poboljšati ukoliko se poveća promet na stranici. Tako su primjerice Instagram i Washington Post stranice napravljene u Djangu i uspješno se nose s velikim prometom.
- Raznovrsnost – Django je iznimno fleksibilan i omogućuje korisnicima isporuku sadržaja u raznim formatima (HTML, RSS, JSON, XML...) te je moguće raditi stranice za gotovo sve operativne sustave poput Windowsa, Linuxa i Mac OS-a. Stoga je pogodan za razvoj aplikacija raznih primjena, kao što su poslovne stranice, vladine, obrazovne, socijalne mreže i slično.

Django je pogodan za početnike koji razvijaju web aplikacije jer ima veliku zajednicu korisnika i dobro dokumentaciju. Tako se na internetu može pronaći puno sadržaja vezanih uz Django, vodiča i savjeta. Jedno od načela kojim Django nastoji olakšati i ubrzati rad je 'ne ponavljaj se'. Tako se određeni predlošci mogu koristiti za više prikaza kako bi se izbjeglo višestruko pisanje istog koda.

4. WEB APLIKACIJA PLEA

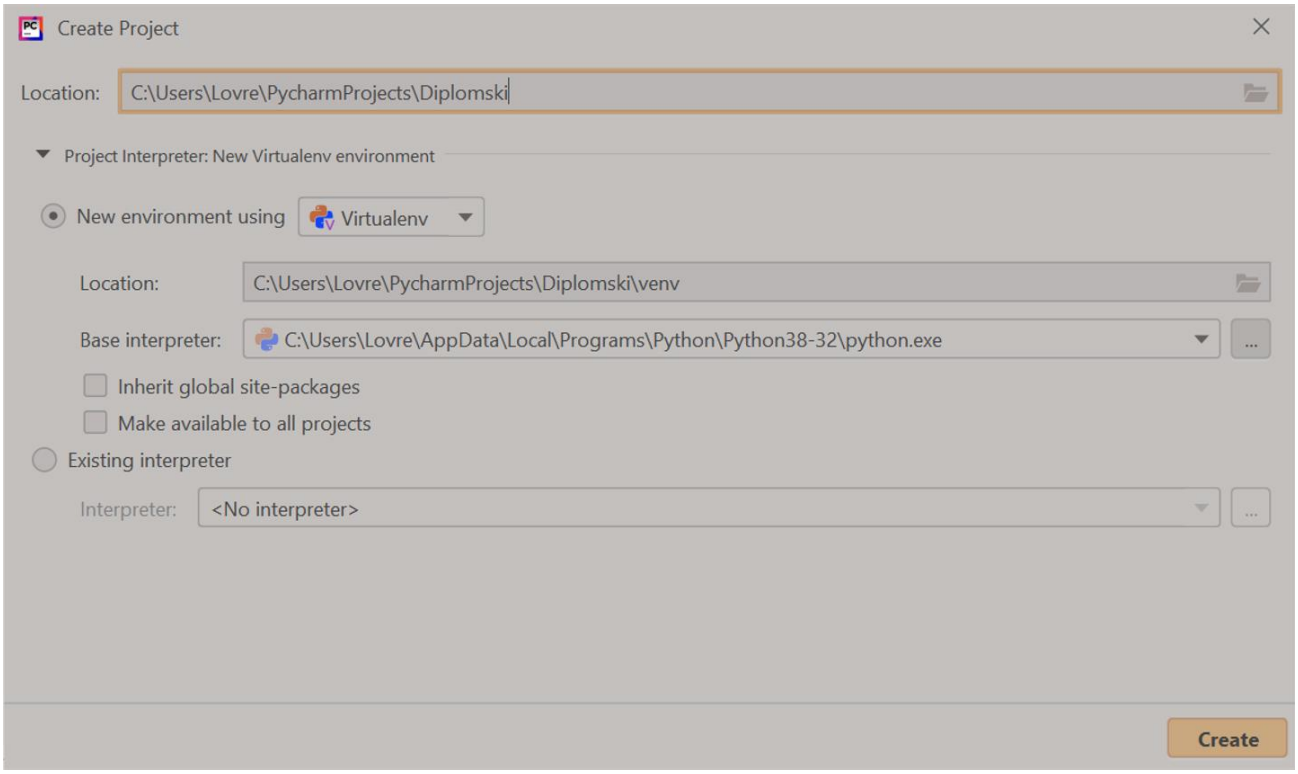
Web aplikacija za vizualizaciju podataka koje prikuplja afektivni robot PLEA napravljena je u PyCharm integriranom razvojnom sučelju. PyCharm sučelje prilagođeno je za korištenje na različitim platformama poput Windowsa, Linuxa i macOS-a te olakšava korisniku rad analizom koda, isticanjem grešaka u kodu, omogućava kreiranje projekata i preglednu strukturu web aplikacije te korištenje raznih okvira za razvoj web aplikacija poput Django-a, web2py-a i Flask-a.

Sve podatke koje prikuplja PLEA spremaju su SQL bazu koja je podignuta na online server te se podaci iz nje povlače na web aplikaciju za vizualizaciju podataka. Dok je PLEA spojena na internet svi podaci se spremaju u stvarnom vremenu, a dok PLEA nema pristup internetu onda sprema podatke u lokalnu datoteku te ih učita u bazu na serveru prilikom idućeg spajanje na internet.

4.1 Struktura web aplikacije

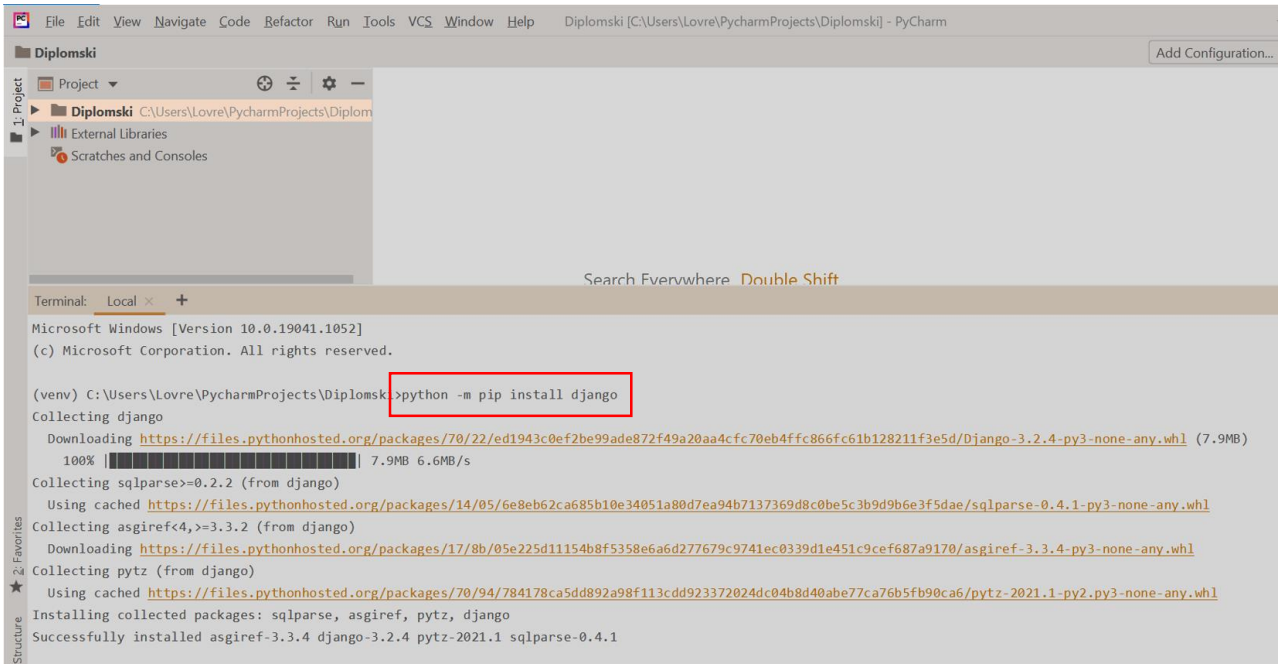
Prvi korak je kreiranje projekta u PyCharmu u koji će se kasnije dodavati svi ostali elementi. U prozoru za kreiranje projekta potrebno je unijeti ime projekta kojim se automatski kreira i lokacija na disku. PyCharm omogućava automatsko kreiranja virtualne okoline[19] koristeći zadnju instaliranu verziju Pythona, bez potrebe za zasebnom instalacijom. Ovim putem se instaliraju brojne biblioteke u projekt koje će olakšati daljnje programiranje, a putem python naredbi moguće je dodavati i ostale biblioteke kao što će biti prikazano u nastavku. Kreiranje projekta s virtualnim okruženjem prikazano je slikom 4.1.

[19] Eng. Virtual environment



Slika 4.1 Kreiranje projekta u PyCharmu

Nakon kreiranja projekta, potrebno je instalirati Django u njemu. Instaliranje Django-a i ostalih potrebnih paketa moguće je izvršiti u terminalu PyCharma koristeći naredbu *"python –m pip"*. Prvi dio naredbe označava korištenje Python interpretera, a *pip* je naredba za upravljanje Python bibliotekama. Naredbu trebamo pokrenuti na istoj lokaciji na kojoj smo i instalirali projekt, u ovom slučaju: *"C:\Lovre\PycharmProjects\Diplomski"*. Django i svi ostali paketi koji se instaliraju unutar projekta spremljeni su u mapu virtualnog okruženja. Instaliranje Django-a putem PyCharm terminala prikazano je slikom 4.2.



Slika 4.2 Instaliranje Django-a putem PyCharm terminala

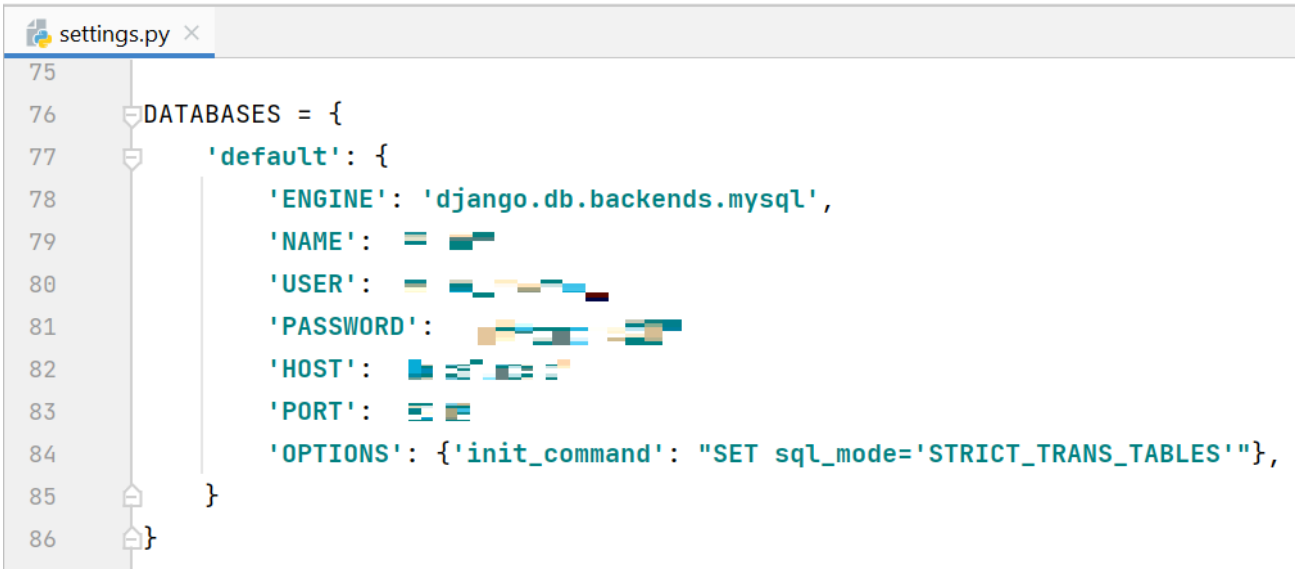
Idući korak je kreiranje Django projekta unutar PyCharm projekta. Django projekt kreira strukturu buduće web aplikacije i datoteke kojima se njome upravlja i povezuje ostale dijelove, od kojih su najbitnije *settings.py* i *urls.py*. Projekt PLEA je također kreiran u PyCharm terminalu, koristeći naredbu *"django-admin startproject PLEA"*. Daljnje naredbe će se izvršavati u projektu PLEA, te je stoga potrebno prebaciti radni direktorij iz PyCharm projekta jednu razinu dublje. Prebacivanje se izvršava u terminalu naredbom *"cd PLEA"*. Povratak jednu razinu unazad u radnom direktoriju može se izvršiti naredbom "..". Nakon što se promijeni radni direktorij promijenit će se i putanja koju je moguće vidjeti u terminalu. Prebacivanje radnog direktorija prikazano je slikom 4.3.

```
(venv) C:\Users\Lovre\PycharmProjects\Diplomski>django-admin startproject PLEA

(venv) C:\Users\Lovre\PycharmProjects\Diplomski>cd PLEA

(venv) C:\Users\Lovre\PycharmProjects\Diplomski\PLEA>
```

Slika 4.3 Prebacivanje radnog direktorija

U datoteci *settings.py* definiraju se glavne postavke stranice i lokacije drugih datoteka. Tako se primjerice definira lokacija datoteke *urls.py* u kojoj su definirane poveznice, lokacija predložaka,

statičnih datoteka i podatkovne baze s kojom je aplikacija spojena. Prilikom kreiranja Django projekta automatski se generira i lokalna SQL baza. Lokalna baza je pogodna za razvojnu fazu aplikacije, no za funkcionalnu aplikaciju najčešće se Django spaja s vanjskom SQL bazom na serveru čija se lokacija i pristupni podaci definiraju u ovoj datoteci. U ovom radu korištena je vanjska SQL baza, a njen pristup prikazan je slikom 4.4. Pristupni podaci su zamagljeni zbog sigurnosnih razloga.



Slika 4.4 Povezivanje sa SQL bazom

Jedan Django projekt može sadržavati više web aplikacijama ovisnosti o zahtjevima i kompleksnosti projekta. Za ovaj rad je dovoljna samo jedna web aplikacija koja je nazvana website, a kreirana je unutar Django projekta naredbom u terminalu *"python manage.py startapp website"*. Kreiranu aplikaciju je potrebno deklarirati u glavnom *settings.py* folderu, pod varijablom *"INSTALLED_APPS"*.

Unutar web aplikacije kreiraju se web predlošci i ostale datoteke, a kako bi se lako grupirale i ispravno prikazivale potrebno je kreirati mape *"templates"* za predloške i *"static"* za statične datoteke poput JavaScript i CSS datoteka. U mapi za predloške se zatim mogu kreirati .html datoteke koje definiraju konačan izgled web stranice.

Za spremanje i dohvaćanje podataka iz baze koriste se modeli, koji su zapravo Python klase i služe kako bi podaci ostali pohranjeni na serveru i nakon što ugasimo stranicu te da ih kasnije možemo jednostavno dohvatiti. U bazi podataka svaki objekt koji se kreira iz pojedinog modela sprema se kao novi redak u tablici. Iz tog razloga potrebno je unaprijed definirati model i unijeti njegovu strukturu,

tj. atribute u bazu podataka kako bi se ispravno definirala tablica s odgovarajućim brojem redaka. Nije moguće naknadno dodati novi atribut nekom objektu ako već ne postoji stupac za taj atribut definiran u tablici, stoga je potrebno za svaku promjenu iznova napraviti update modela u bazi podataka. Definiranje početnog modela i sve kasnije izmjene rade se postupkom koji se zove migracije, a predstavlja Python skriptu koja osigurava da je baza podataka sinkronizirana s kodom.

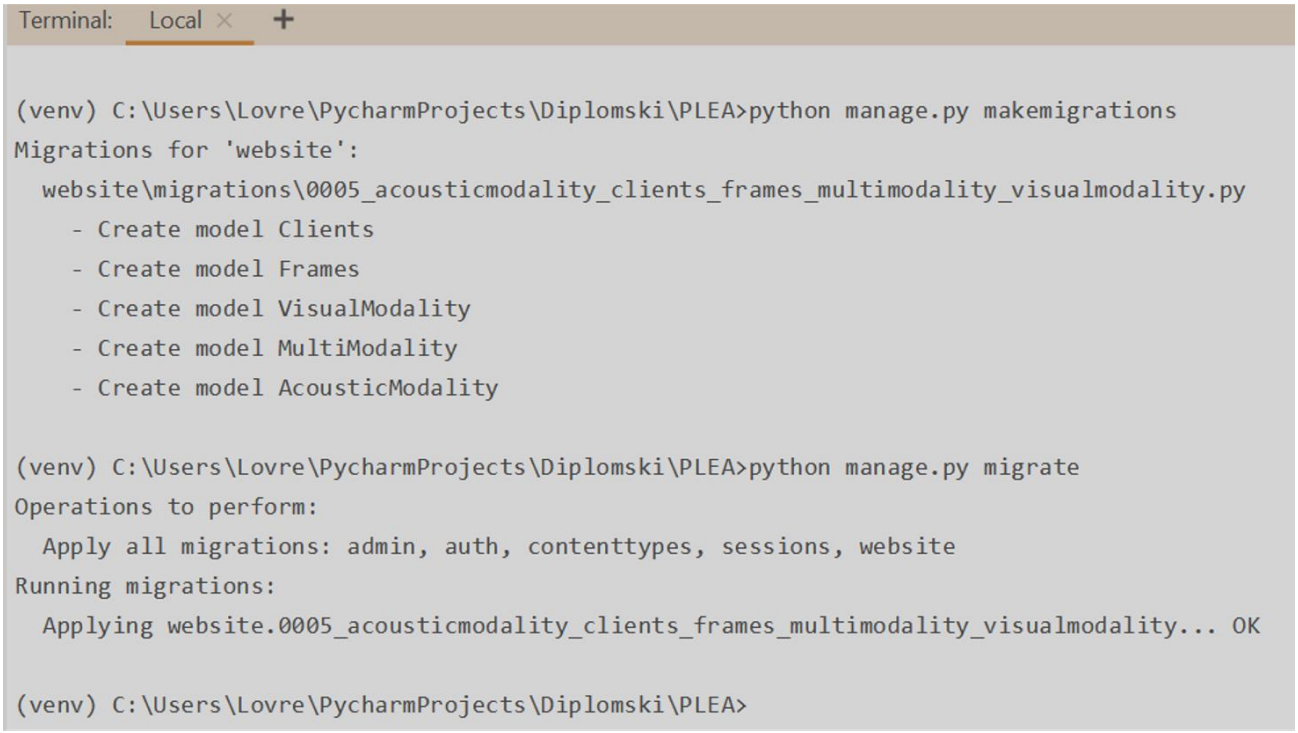
Prilikom kreiranje web aplikacije Django automatski stvori i datoteku *models.py* u kojoj se definiraju modeli. Svaki model se kreira kao klasa koja je izvedena iz Django-ve klase modeli, kako bi Python automatski prepoznao da se radi o klasi koja će služiti za spremanje podataka u bazu. Nakon što se definira model (klasa) sa svim željenim atributima pokreću se migracije komandom u terminalu *"python manage.py makemigrations"*. Ova komanda usporedi našu postojeću bazu podataka sa željenim modelom i definira koje se sve promjene moraju izvršiti kako bi se one uskladile. Nakon izvršavanja ove komande u web aplikaciji se kreira nova mapa zvana *migrations* i u njoj Python datoteka u kojoj su definirane sve promjene koje je potrebno izvršiti na bazi podataka. U ovom trenutku nikakve promjene se još nisu dogodile na bazi, nego su one samo definirane u Python datoteci. Slikom 4.5 prikazna je kreirana datoteka migracija u kojoj su definirane promjene koje je potrebno izvršiti na bazi podataka.

0003_acousticmodality_clients_frames_multimodality_visualmodality.py

```
class Migration(migrations.Migration):

    dependencies = [
        ('website', '0003'),
    ]

    operations = [
        migrations.CreateModel(
            name='Clients',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('uid', models.CharField(default='', max_length=50)),
                ('nickname', models.CharField(default='', max_length=32)),
                ('first_log', models.PositiveBigIntegerField(default=0)),
            ],
        ),
        migrations.CreateModel(
            name='Frames',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('timestamp', models.PositiveBigIntegerField(default=0)),
                ('ip', models.CharField(default='', max_length=255)),
                ('client', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='website.clients')),
            ],
        ),
```

Slika 4.5 Python datoteka migracija

Kako bi se promjene stvarno unijele u bazu podataka potrebno je pozvati komandu *"python manage.py migrate"*, koje izvrši sve prethodno definirane promjene. Moguće je pripremiti više odvojenih migracijskih datoteka te ih unijeti u bazu sve istovremeno. Definiranje i pokretanje svih migracije u terminalu prikazano je slikom 4.6.



Slika 4.6 Definiranje i pokretanje migracija

U ovom radu korišteno je 5 modela za spremanje podataka u bazu. Prvi model se zove *Clients*[20] i služi za spremanje podataka o klijentima, tj. korisnicima PLEA-e. Atributi modela *Clients* su:

- *uid* – predstavlja jedinstveni id korisnika
- *nickname* – ime (nadimak) koji je dodijeljen korisniku
- *first_log* – vrijeme kada se korisnik prvi put prijavio.

Atributi *uid* i *nickname* su polja znakova, dok je *first_log* brojčana vrijednost u Unix formatu koji služi za zapis vremena. U Unix formatu vrijeme je zapisano u broju milisekundi koliko je prošlo od početnog datuma koji je dogovoren kao 1.1.1970. Datumi koji su prije početnog datuma zapisani su kao negativna brojčana vrijednost, tj. koliko je milisekundi određeni trenutak prije početnog datuma.

[20] Klijenti

Zapis je osim milisekundama moguće predočiti i sekundama, nanosekundama ili drugom skalom po istom principu. Tako je primjerice Unix zapis u milisekundama 1575909015000 predstavlja datum 2019-12-09 05:30:15.

Drugi model u bazi je *Frames*[21], koji predstavlja trenutke u kojima su zabilježena emocionalna stanja. Model *Frames* je u *many-to-one*[22] korelaciji s modelom *Clients*, što znači da za svakog pojedinog klijenta može biti vezano više trenutaka. Atributi ovog modela su:

- *client* – označava poveznicu s određenim *Client* modelom
- *timestamp* – vrijeme kada je trenutak zabilježen
- *ip* – IP adresa s koje je trenutak zabilježen.

Timestamp atribut je također brojčana vrijednost u Unix formatu kao i *first_log*, *ip* je polje znakova, a *client* je posebno Django-vo polje koje označava povezanost s drugim modelom. Client atribut ima definiranu Django funkciju *"on_delete=models.CASCADE"*, koja će izbrisati sve *Frames* modele iz baze ako se izbriše klijent s kojim su ti trenutci povezani.

U svakom zabilježenom trenutku u bazu se spremaju emocionalna stanja triju modalnosti, vizualne, auditivne i ukupne. Svaka od tih modalnosti spremljena je u bazu putem zasebnog modela koji se zovu *VisualModality*, *AcousticModality*, *MultyModality*. Sve tri modalnosti povezane su s jednim trenutkom po principu *one-to-one*[23], što znači da svaki zabilježen trenutak može biti povezan sa samo jednim vizualnim, auditivnim i ukupnim modelom. Svaki od ova 3 modela ima 8 istih atributa:

- *frame* – poveznica s *Frames* modelom
- *angry* (ljut)
- *sad* (tužan)
- *neutral* (neutralan)
- *surprised* (iznenađen)
- *fear* (prestrašen)
- *disguised* (gadi se)

[21] Trenutci

[22] Više na jedan

[23] Jedan na jedan

- *happy* (sretan).

Frame atribut je posebno Django polje za povezivanje s drugim modelom, a ostalih 7 atributa su definirani kao brojčane vrijednosti i izražavaju u postotku koliko intenzivno korisnik osjeća određenu emociju. S obzirom da se zbog bolje preciznosti vrijednosti emocija spremaju u vrijednostima 0-10000, njihove vrijednosti su deklarirane zasebnom klasom, *CustomIntField*, koja je deklarirana tako da dohvaćene vrijednosti iz baze podijeli s 100 prije nego ih proslijedi dalje. Modeli *Frames* i *VisualModality* prikazani su slikom 4.7.

models.py

```
18  class Frames(models.Model):
19      client = models.ForeignKey(Clients, on_delete=models.CASCADE)
20      timestamp = models.PositiveBigIntegerField(default=0)
21      ip = models.CharField(max_length=255, default='')
22
23
24  class VisualModality(models.Model):
25      frame = models.OneToOneField(Frames, on_delete=models.CASCADE)
26      angry = CustomIntField(default=0)
27      sad = CustomIntField(default=0)
28      neutral = CustomIntField(default=0)
29      surprised = CustomIntField(default=0)
30      fear = CustomIntField(default=0)
31      disguised = CustomIntField(default=0)
32      happy = CustomIntField(default=0)
```

Slika 4.7 Django model emocija

U datoteci *urls.py* definirane su poveznice koje aktiviraju određene kontrolere, koji zatim pokreću određeni prikaz ili šalju neki drugi odgovor stranici na stranicu. U ovu datoteku potrebno je uvesti sve kontrolere iz njihove datoteke, kako bi ih mogli pozivati kada je to potrebno. Putanja svake poveznice ima 2 obavezna parametra i jedan opcionalni. Obavezni parametri su sufiks adrese web aplikacije i kontroler koji se poziva. Opcionalni parametar je naziv koji se može dodijeliti kako bi se u ostalim dijelovima aplikacije moglo jednostavnije referencirati na traženu poveznicu. Slikom 4.8 prikazano je definiranje poveznica.

urls.py

```
from Website.views import homepage, emotional_state, get_ids, get_clients, get_frames, get_frame,\
    get_visual_modality, get_audio_modality, get_total_modality, get_visual_array, get_audio_array, get_total_array

urlpatterns = [
    path('admin/', admin.site.urls),
    path('homepage', homepage, name="homePage"),
    path('state/<int:id>', emotional_state, name="emotionalState"),
    path('getVisual', get_visual_array, name="getVisual"),
    path('getAudio', get_audio_array, name="getAudio"),
    path('getTotal', get_total_array, name="getTotal"),
    path('getIds', get_ids, name="getIds"),
    path('getClients', get_clients, name="getClients"),
    path('getFrames', get_frames, name="getFrames"),
    path('getFrame/<int:id>', get_frame, name="getFrame"),
    path('getVisualModality/<int:id>', get_visual_modality, name="getVisualModality"),
    path('getAudioModality/<int:id>', get_audio_modality, name="getAudioModality"),
    path('getTotalModality/<int:id>', get_total_modality, name="getTotalModality"),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Slika 4.8 Definiranje poveznica

Kontroleri definiraju predloške koji će se prikazati i informacije koje će se s njima proslijediti. U kontrolere je potrebno uvesti model Emocija koji će se kasnije prosljeđivati u određene prikaze, Django funkcije za definiranje web odgovora i izbacivanje grešaka. U ovom radu su neke od korištenih funkcija: *render* (prikaz određenog predloška), *get_object_or_404* (dohvati objekt ili baze ili odgovori greškom 404 ako se objekt ne može dohvatiti), *redirect* (preusmjeravanje), i *JsonResponse* (Json odgovor). U ovom radu Json odgovor se koristi za dohvaćanje određenih emocionalnih stanja. JavaScript funkcijom se s web stranice šalje zahtjev na određeni URL, zatim se poziva određeni kontroler koji iz modela iz baze uzima određeno emocionalno stanje, od njega kreira listu te ju vraća na stranicu kao Json odgovor koji onda dohvaća JavaScript te ga dalje obrađuje. Primjer kontrolera kojim se iz baze dohvaća model *AcousticModality* te se sve njegove vrijednosti prosljeđuju kao Json odgovor dan je slikom 4.9, a sam Json odgovor koji se proslijeđuje na stranicu prikazan je slikom 4.10.

views.py

```
def get_audio_array(request):
    AudioArray = list(AcousticModality.objects.values())
    return JsonResponse(AudioArray, safe=False)
```

Slika 4.9 Kontroler s Json odgovorom

[{"id": 1, "frame_id": 1, "angry": 11, "sad": 22, "neutral": 33, "surprised": 44, "fear": 55, "disguised": 66, "happy": 77}, {"id": 2, "frame_id": 2, "angry": 23, "sad": 12, "neutral": 55, "surprised": 88, "fear": 4, "disguised": 0, "happy": 12}, {"id": 3, "frame_id": 3, "angry": 3, "sad": 77, "neutral": 0, "surprised": 55, "fear": 0, "disguised": 15, "happy": 22}]

Slika 4.10 Json odgovor

Okidač za slanje ovakvog Json odgovora je AJAX[24] zahtjev koji se šalje putem JavaScripta. AJAX služi za dohvaćanje podataka sa servera te mijenjanje sadržaje stranice s prikupljenim podacima, bez potrebe da se osvježava čitava stranica. Strukturu AJAX zahtjeva čini poveznica na koju će zahtjev biti poslan, metoda, odnosno šaljemo li ili primamo zahtjev te funkcije koje će se izvršiti ukoliko se zahtjev uspješno izvrši te dobijemo odgovor servera, odnosno funkcije koje će se izvršiti ako zahtjev ne bude uspješan. Izgled klasičnog zahtjeva koji je korišten u ovom radu prikazan je slikom 4.11, a funkcije koje se izvršavaju u slučaju uspjeha su prikrivene radi bolje preglednosti.

tableFill.js

```
5   $.ajax({
6       url: '/getClients',
7       method: 'GET',
8       success: function (data) {...},
48      error: function (error) {
49          console.log(error);
50      },
51  });
```

Slika 4.11 AJAX struktura

Osim putem Json odgovora, stanje iz baze možemo poslati direktno iz baze na stranicu prilikom njenog učitavanja. Na ovaj način moguće je poslati cjelokupni model (što se ne preporučuje jer na taj način šalje velika količina podataka koja nije potrebna te se tako usporava proces) ili njegov dio, kao što je u ovom radu korišteno dohvaćanja samo jednog stanja putem njegovog id-a. Slikom 4.12 prikazano je dohvaćanje stanja iz modela *Frame* koji ima traženi id, te se zatim prikazuje novi

[24] Asynchronous JavaScript And Xml (Asinkroni JavaScript i Xml)

predložak kojem se prosljeđuje dohvaćeni objekt. Ukoliko se ne uspije dohvatiti traženi objekt, Django će izbaciti grešku 404.

views.py

```
16  # send initial Frame when loading emotional state page
17  def emotional_state(request, id):
18      Frame = get_object_or_404(Frames, pk=id)
19      return render(request, "Main/emotionalState.html", {"Frame": Frame})
```

Slika 4.12 Generiranje predloška uz objekt iz modela

Django aplikacije također imaju i automatski generirano administratorsko sučelje čija je uobičajena adresa */admin*, ali može se promijeniti u datoteci *urls.py*. Pristup administratorskom sučelju omogućen je samo administratorima, stoga je potrebno kreirati administratorski račun u Django. On se kreira naredbom *"python manage.py createsuperuser"*. Nakon pozivanja naredbe traži se unos korisničkog imena, e-mail adrese i lozinke. U administratorskom sučelju moguće je kreirati račune za ostale korisnike i korisničke grupe. Također, moguće je pristupiti bazi podataka te uređivati, dodavati i brisati objekte postojećih modela. Kako bi pristupili prethodno kreiranim modelima potrebno ih je deklarirati u datoteci *admin.py* naše web aplikacije. U njoj je potrebno uvesti prethodno deklarirani model naredbom *"from .models import Emotions"* te ga registrirati u administratorsko sučelje naredbom *"admin.site.register(Emotions)"*.

Pokretanje web aplikacije na lokalnoj adresi vrši se naredbom *"python manage.py runserver"* u terminalu. Izvršavanjem te naredbe pokreće se provjere koda te ukoliko je sve uredu aplikacije se podiže online te joj se može pristupiti u web pregledniku na adresi *http://127.0.0.1:8000*. Ovo je lokalna adresa računala, što znači da će aplikaciji biti moguće pristupiti samo s računala na kojem je podignuta te ovakav način podizanja aplikacije služi isključivo za razvojne svrhe. Završetak rada, tj. gašenje aplikacije vrši se tipkom ctrl + c. Pokretanje aplikacije na lokalnoj adresi prikazano je slikom 4.13.

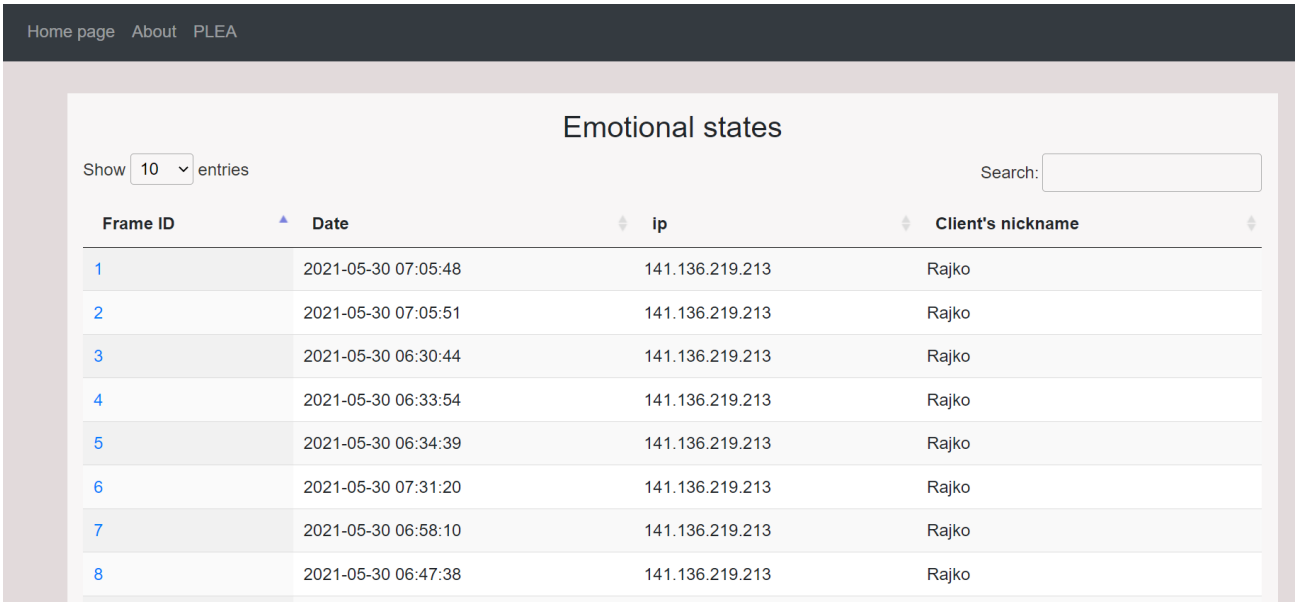
```
(venv) C:\Users\Lovre\PycharmProjects\Plea\Plea>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 24, 2021 - 10:41:48
Django version 3.1.7, using settings 'Plea.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Slika 4.13 Pokretanje web aplikacije na lokalnoj adresi

4.2 Izgled web aplikacije

Navigacijska traka sadrži 3 poveznice: *Home page*, *About* i PLEA. *Home page* služi za povratak na početnu stranicu, *About* otvara u novom prozoru poveznicu na vanjsku stranicu gdje je opisana PLEA, a PLEA otvara poveznicu na kojoj se može pristupiti PLEA-i i komunicirati s njom.

Početna stranica web aplikacije PLEA za vizualizaciju podataka je tablični prikaz svih zabilježenih emocionalnih stanja. U tablici su prikazani ID svakog zabilježenog trenutka, vrijeme kada je zabilježeno, IP adresa na kojoj je trenutak zabilježen te ime korisnika kod koga je očitano emocionalno stanje. Tablicu je moguće sortirati s obzirom na bilo koje svojstvo te je također implementirana tražilica koja omogućava pretragu točno određenog ID-a, vremena, ip-a ili imena korisnika. Zadani prikaz je namješten na prikaz 10 stanja po stranici kako bi se smanjila količina nepotrebnih informacija na stranici, no taj broj se može promijeniti u padajućem izborniku te tako namjestiti prikaz na 25, 50 ili 100 prikaza po stranici. U ID svakog stanja integrirana je poveznica, te se tako klikom na ID stanja otvara stranica za njegov detaljan prikaz. Početna stranica prikazana je slikom 4.14.



Slika 4.14 Izgled početne stranice

Tablični prikaz podataka napravljen je korištenjem DataTables Javascript biblioteke. U Html datoteci definiran je samo kostur tablice, tj. zaglavlje u kojem su definirani stupci i njihovi nazivi. U Javascript datoteci se šalje AJAX zahtjev u bazu te se iz baze vraća popis svih zabilježenih trenutaka kao Json odgovor. Zatim se pomoću DataTables biblioteka dobiveni podatci pune u tablicu u odgovarajuće stupce te se u stupac ID implementira poveznicama klikom na koju se prelazi na stranicu za prikaz odgovarajućeg emocionalnog stanja odabranog trenutka. DataTables biblioteku nije potrebno instalirati, kao ni ostale korištene biblioteke poput D3.js-a, već se koriste njihove online verzije čije se korištenje i adresa za pristup definira u zaglavlju web stranice. Definiranje DataTables biblioteke, te ostalih korištenih poput Bootstrap-a, D3.js-a, jquery-a i drugih prikazano je slikom 4.15.

homePage.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>PLEA</title>

    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
    />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <!-- data table script and css -->
    <link
      rel="stylesheet"
      type="text/css"
      href="https://cdn.datatables.net/1.10.24/css/jquery.dataTables.min.css"
    />
    <script src="https://cdn.datatables.net/1.10.24/js/jquery.dataTables.min.js"></script>
    <script src="https://momentjs.com/downloads/moment.js"></script>

    {% load static %}
    <link rel="stylesheet" href="{% static 'main.css' %}" />
  </head>
```

Slika 4.15 Definiranje online biblioteka

Na stranici za prikaz emocionalnog stanja naznačeno je vrijeme kada je to stanje zabilježeno te klijent na kojem je stanje očitano. Osim toga, prikazan je popis svih zabilježenih emocija (sreća, tuga, ljutnja, iznenađenje, strah, gađenje i neutralno) te njihove vrijednosti izražene u postocima. Kako bi se što lakše uspoređivalo pojedina stanja, dodani su gumbovi za prelazak na prikaz prethodnog[25] stanja te idućeg[26] stanja.

Prijelaz na iduće ili prethodno stanje se odvija na način da se prvo na server pošalje AJAX zahtjev kojim se dohvaća popis svih ID-jeva trenutaka iz baze te se zatim nakon što se pronađe traženo stanje pokreće funkcija updateFrame koja dohvaća emocionalno stanje novog trenutka. Kako bi si izbjeglo

[25] Eng. previous

[26] Eng. next

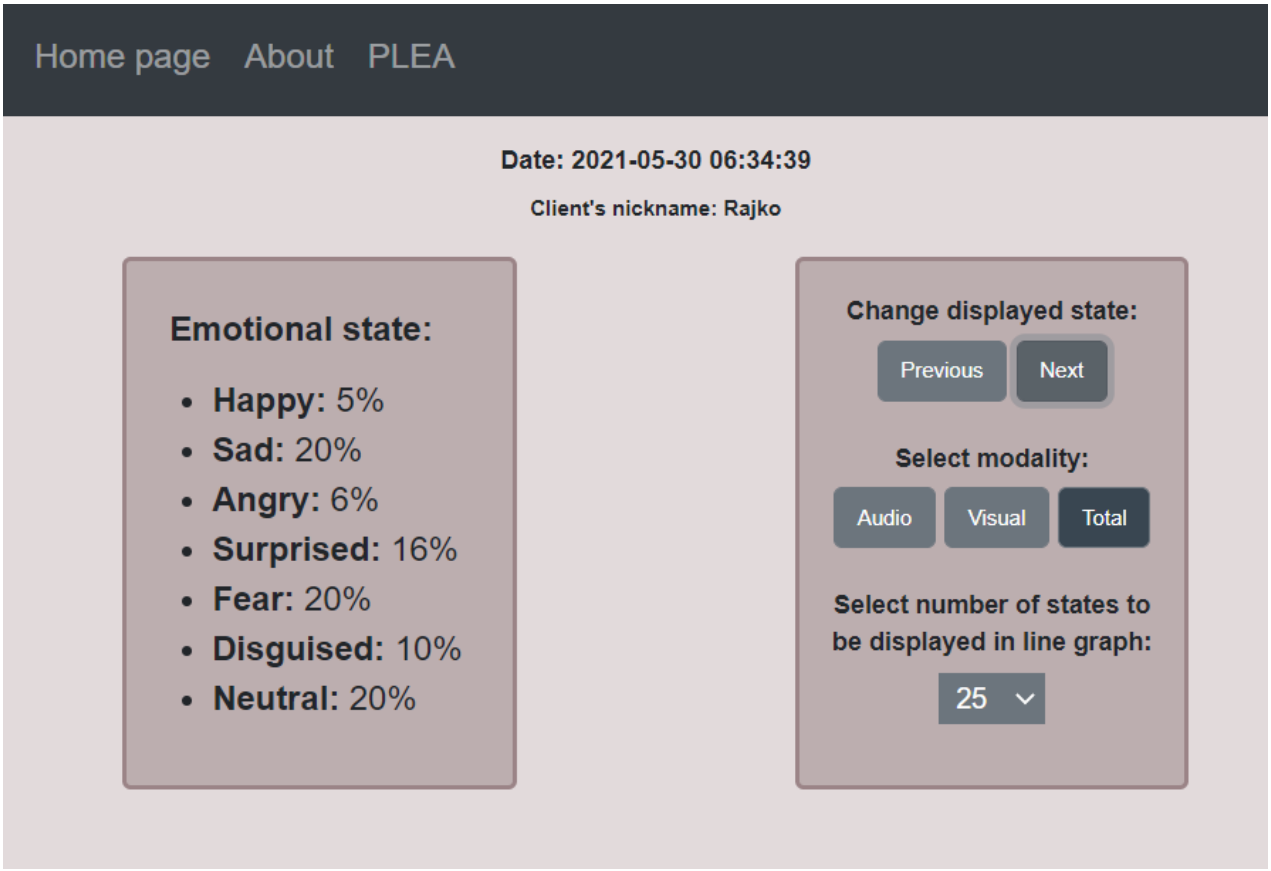
ponovo učitavanje cijele stranice prilikom prijelaza na iduće ili prethodno stanje, JavaScriptom se unose sve potrebne promjene u polja na Html stranici te se osvježavaju grafovi na postojećoj stranici s novim podacima. Ukoliko je već prikazano prvo stanje, a klikne se gumb za prijelazak na iduće iskočit će obavijest da je to nemoguće jer je već prikazano prvo stanje. Ista stvar vrijedi i za stanje i gumb za prijelaz na iduće. Funkcija koja se poziva klikom na gumb za prijelaz na prethodno stanje prikazana je slikom 4.16.

buttonEvents.js

```
273  var idList;
274  var indexNumber;
275  $('#previousButton').click(function () {
276    $.ajax({
277      url: '/getIds',
278      method: 'GET',
279      success: function (data) {
280        idList = data;
281        indexNumber = idList.findIndex(function findId(id) {
282          return id == $('#frameId').val();
283        });
284        if (indexNumber > 0) {
285          indexNumber = idList[indexNumber - 1];
286          $('#frameId').val(indexNumber);
287          updateFrame();
288        } else {
289          alert('This is the first state!');
290        }
291      },
292      error: function (error) {
293        console.log(error);
294      },
295    });
296  });
```

Slika 4.16 Funkcija za prijelaz na prethodno stanje

Osnovni prikaz prikazuje konačno[27] emocionalno stanje nakon provedene modalne analize, ali također je moguće odabrati prikaz stanja dobivenog samo audio ili vizualnom analizom. U kartici gdje se nalaze prethodno spomenuti gumbovi, također se nalazi i padajući izbornik za odabir broja stanja prikazanih u linijskom grafu koje će biti opisan naknadno. Brojčani prikaz emocionalnog stanja i gumbovi prikazani su slikom 4.17.



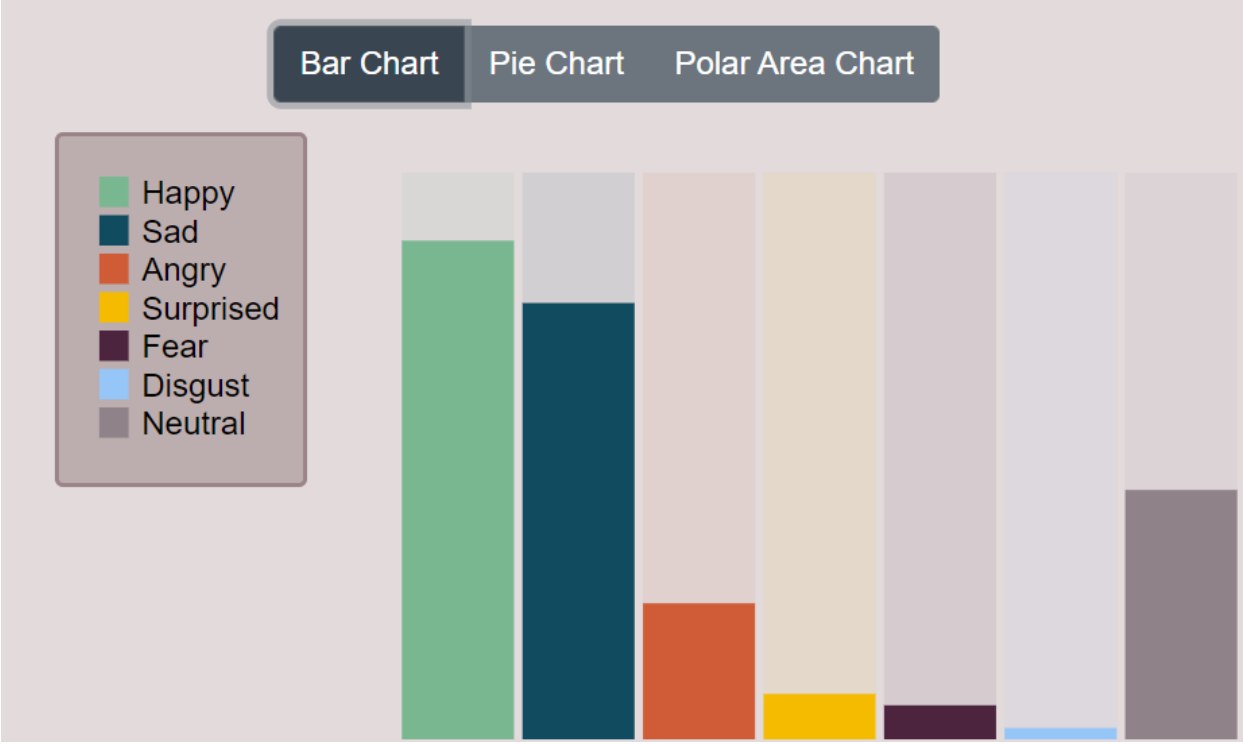
Slika 4.17 Brojčani prikaz i gumbovi

Promjena prikazane modalnost odvija se na način da se na server šalje AJAX zahtjev kojim se dohvaća emocionalno stanje odabrane modalnosti za trenutno prikazani trenutak. Kako bi se izbjeglo dohvaćanje veće količine nepotrebnih podataka prilikom slanja zahtjeva pošalje se i ID prikazanog trenutka te se iz baze šalje samo to jedno stanje. Primljeni podaci se zatim spremaju u Html varijable kojim se prikazuje brojčani prikaz svake od emocije te se ažurira varijabla *chartData* koja se koristi za generiranje grafikona. Nakon ažuriranja podataka pokreće se funkcija za ažuriranje grafikona te se

[27] Eng. total

šalje još jedan AJAX zahtjev na server kojim se dohvaćaju podaci odabrane modalnost za linijski graf, nakon čega se i on ažurira.

Osim brojčanih vrijednosti, emocionalno stanje također je prikazano i grafikonima kako bi se lakše mogli vizualizirati njihovi međusobni odnosi i dominacija. Moguć je odabir prikaza između histograma[28] (stupčanog grafikona), tortnog grafikona[29] i polarnog grafikona[30]. Prikazi sva 3 grafikona prikazani su slikama 4.18, 4.19 i 4.20.

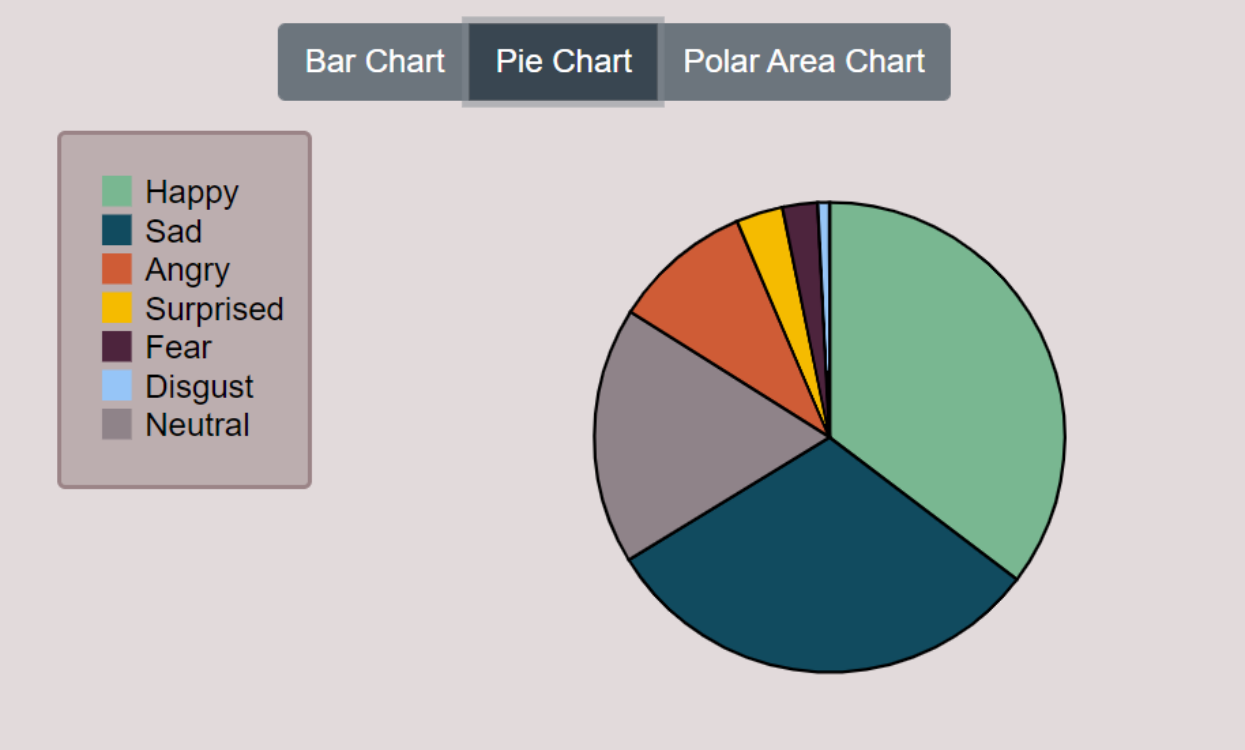


Slika 4.18 Histogram

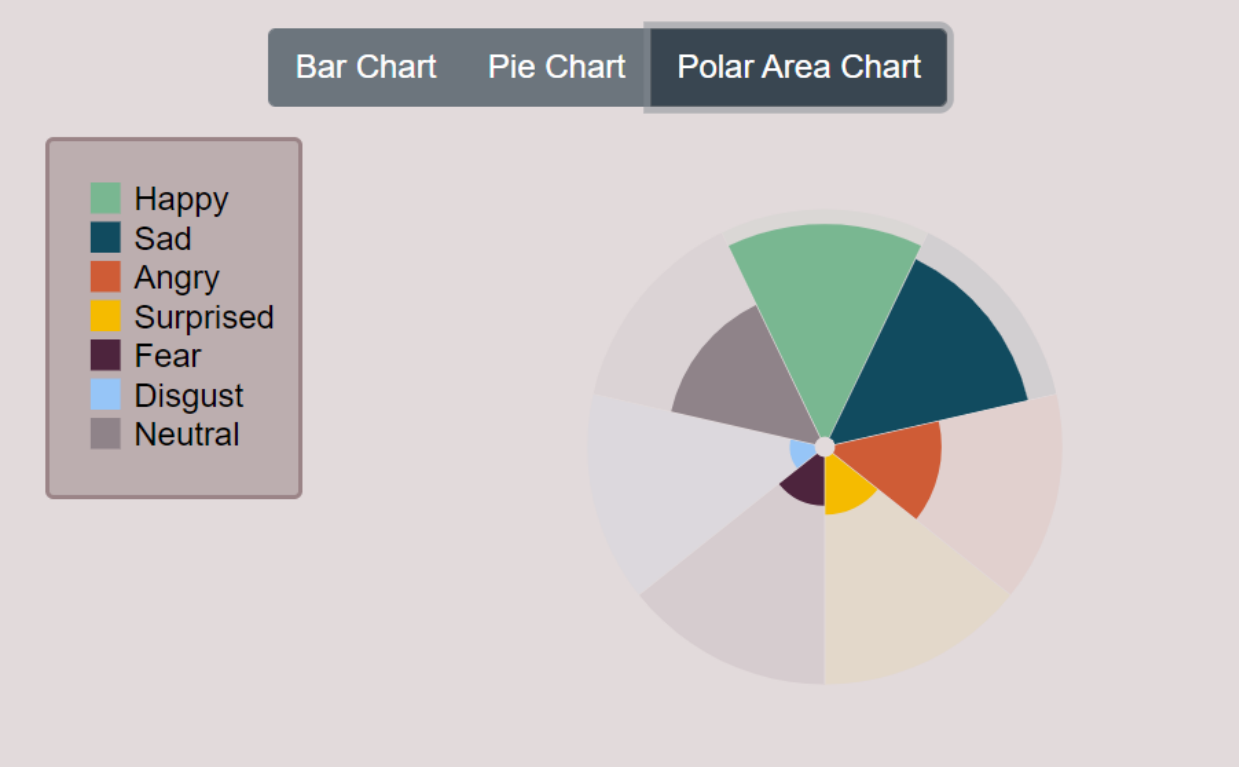
[28] Eng. Bar chart

[29] Eng. Pie chart

[30] Eng. Polar area chart



Slika 4.19 Tortni grafikon



Slika 4.20 Polarni grafikon

Svi grafikoni su generirani pomoću D3.js biblioteke. Ulazne podatke čini jedno emocionalno stanje, tj. 7 emocije zabilježenih u određenom trenutku s odabranom modalnošću. Ostali parametri poput visine i širine SVG elemente te radijusi su definirani na početku JavaScript datoteke. Za ovu svrhu u Html datoteci kreiran je div element koji ima ID *"chartDiv"*, koji se kasnije odabere pomoću D3.js funkcije i u njemu se kreira SVG element s grafikonom. Primjer koda kojim se kreira tortni grafikon prikazan je slikom 4.21.

charts.js

```
110 function PieChart() {
111   var svg = d3
112     .select('#chartDiv')
113     .append('svg')
114     .attr('width', width)
115     .attr('height', height)
116     .attr('id', 'chart')
117     .append('g')
118     .attr('transform', 'translate(' + width / 2 + ',' + height / 2 + ')');
119
120   var pie = d3.pie().value(function (d) {
121     return d.value;
122   });
123   var data_ready = pie(d3.entries(chartData));
124
125   svg
126     .selectAll('#chart')
127     .data(data_ready)
128     .enter()
129     .append('path')
130     .attr('d', d3.arc().innerRadius(0).outerRadius(radius))
131     .attr('fill', function (d, i) {
132       return colorSelect(i);
133     })
134     .attr('stroke', 'black')
135     .style('stroke-width', '1.5px');
136 }
```

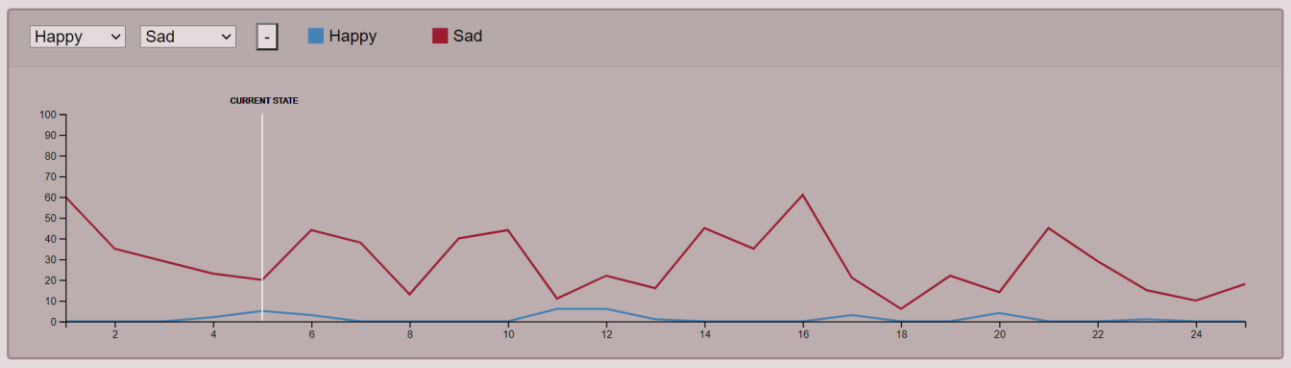
Slika 4.21 Kreiranje tortnog grafikona

Prvi korak pri kreiranju tortnog grafikona je odabir željenog elementa, u ovom slučaju je to element s ID-jem *"chartDiv"*, kojem se zatim pridodaje SVG element širine *width* i visine *height*. Varijable *width* i *height* su definirane na početku datoteke s vrijednostima 300, odnosno 233. SVG elementu su doda ID *"chart"* te se on translatira na sredinu elementa. Zatim se drugom naredbom odabere kreirani element te se u njega ubaci niz podataka *chartData*, koji je također prethodno definiran, te se po ulaznim podacima kreiranju putanje crne boje širine 1.5 px. Unutarnji radijus je 0, kako bi se izbjegao prazni prostor u sredini grafikona, a vanjski radijus definiran je varijablom *radius* koja iznosi 115 px. Odabir boje svakog dijela tortnog grafikona definiran je vanjskom funkcijom *colorSelect*, koja vraća odgovarajuću boju za svaki atribut ulaznih podataka.

Osim grafikona za prikaz odnosa emocija svakog pojedinačnog stanja, također je dan i prikaz linijskim grafom koji omogućava praćenje određene emocije tijekom vremena. Ovaj graf omogućava detektiranje trenutaka koji su doveli do naglih promjena emocionalnog stanja i praćenje kako se korisnikovo raspoloženje mijenja tijekom vremena. Osim praćenja promjena pojedinačne emocije, omogućeno je i dodavanje druge linije u graf kako bi se moglo pratiti i uspoređivati dvije emocije. Emocije se gotovo uvijek mijenjaju u zavisnosti jedna prema drugoj pa je ovakav prikaz koristan za uočiti kako promjena jedne emocije utječe na promjenu druge.

U padajućem izborniku moguće je odabrati koja se emocija prikazuje svakom od linija u grafu. Osnovni prikaz linijskog grafa uključuje 15 emocionalnih stanja, a u padajućem izborniku moguće je promijeniti prikaz na 25 ili 50 emocionalnih stanja. Trenutno odabrano emocionalno stanje uvijek će se nastojati prikazati u sredini linijskog grafa, tako primjerice ako se odabere prikaz 25 stanja, bit će prikazano 12 prethodnih i 12 stanja nakon odabranog. No ukoliko to nije moguće, primjerice ako se prikazuje zadnje stanje iz baze podataka onda će biti prikazano 24 prethodna stanja uz ono odabrano.

U grafu je na osi y prikazan intenzitet emocije izražen u postotku, dok je na osi x prikazan broj stanja gdje svaki broj predstavlja jedno emocionalno stanje. Svijetlom vertikalnom linijom je označeno trenutno odabrano stanje. Linijski graf na kojem je prikazano 50 stanja sreće i tuge prikazan je slikom 4.22.



Slika 4.22 Linijski graf

Izgled sadržaja na stranici, odnosno boja pozadine, font teksta, veličina fonta, obrubi, margine i ostale karakteristike definirani su CSS-om. Kako bi se izbjeglo nepotrebno ponavljanje istih atributa za više sličnih elemenata kreirana je jedna glavna CSS datoteka u kojoj su definirane klase i razni parametri za svaku klasu. U Html datoteku se poziva glavna CSS datoteka te se Html elementima pridružuju samo klase definirane u CSS-u. Primjer definiranja nekih klasa i atributa prikazan je slikom 4.23.

main.css

```
8   .colorTable {
9     background-color: #f8f6f6;
10  }
11  .card {
12    border: 2px solid #9b8487;
13    background-color: #bcaeaf;
14  }
15  .cardText {
16    font-size: 11.9px;
17    margin-top: 18px;
18    margin-bottom: 5px;
19  }
20  .cardButton {
21    font-size: 10px;
22    padding-left: 10px;
23    padding-right: 10px;
24  }
25  .cardPrimary {
26    margin: 15px;
27    height: 250px;
28    width: 185px;
29    display: inline-block;
30  }
```

Slika 4.23 CSS klase

5. ZAKLJUČAK

Afektivni robot PLEA omogućava komunikaciju s korisnikom pri čemu se multimodalnom analizom očitava emocionalno stanje korisnika te se vrijednosti emocija svake sekunde pohranjuje u bazu podataka. Kako bi prikupljeni podaci bili što pregledniji i jednostavniji za korištenje, u sklopu ovog rada napravljena je web aplikacija za vizualizaciju podataka u kojoj se emocionalno stanje korisnika prikazuje grafikonima iz kojih se jasno vide međusobni omjeri emocija u svakom trenutku te grafom za vremensko praćenje određene emocije na kojem se lako mogu uočiti prijelomne točke u kojima je došlo do promjena u raspoloženju korisnika.

Osnovna primjena ovog robota i sustava za vizualizaciju bila bi u podučavanju na daljinu. Ovim putem nastavnik bi mogao pratiti koliko su učenici zainteresirani te kako doživljavaju pojedine informacije, dok bi istovremeno i učenici mogli vidjeti emocije profesora koje su prikazane avatarom na robotskoj glavi.

Osim u učenju, u budućnosti se ovakav robot može primijeniti i u drugim indutrijama, primjerice za testiranje novih reklama kako bi se vidjelo kako ih doživljavaju potencijalni korisnici te kako im se raspoloženja mijenja u svakom trenutku prikazivanja reklame. Također, primjena bi se mogla naći i u zabavnom programu i stand-upu, gdje bi komičari mogli testirati svoje štoseve prije samog nastupa i vidjeti kako publika reagira na njih.

LITERATURA

[1] See What I'm Saying: The Extraordinary Powers of Our Five Senses, Dr. L.D. Rosenblum, Dr. Harold Stolovitch, Dr Erica Keeps, 2010

[2] Information Visualization – Perception for Design, Colin Ware, Morgan Kaufmann, 225 Wyman Street, Waltham, MA, SAD, 2013

[3] Energija u Hrvatskoj – godišnji energetski pregled, Ministarstvo zaštite okoliša i energetike, 2018.

[4] https://www.zet.hr, pristupljeno: 2021-05-24

[5] The Use of Visualization in Teaching and Learning Process for Developing Critical Thinking of Students, Kyvete Shatri, Kastriot Buza, 2017

[6] https://datareportal.com/global-digital-overview, pristupljeno: 2021-05-26

[7] https://www.tportal.hr/media/thumbnail/w1000/1172482.jpeg, pristupljeno: 2021-05-26

[8] https://www.ictshore.com/wp-content/uploads/2018/08/sfw0001-01-Web_application_structure.png, pristupljeno: 2021-05-26

[9] Beginning SVG, Alex Libby, 2018

[10] https://developer.mozilla.org/en-US/docs/Web/SVG, pristupljeno: 2021-05-27

[11] https://d3js.org, pristupljeno: 2021-05-27

[12] https://www.djangoproject.com, pristupljeno: 2021-05-31

PRILOG

Kod datoteke urls.py:

```
from django.contrib import admin
from django.urls import path

from django.conf import settings
from django.conf.urls.static import static


from Website.views import homepage, emotional_state, get_ids, get_clients, get_frames,
get_frame,\
    get_visual_modality, get_audio_modality, get_total_modality, get_visual_array,
get_audio_array, get_total_array

urlpatterns = [
    path('admin/', admin.site.urls),
    path('homepage', homepage, name="homePage"),
    path('state/<int:id>', emotional_state, name="emotionalState"),
    path('getVisual', get_visual_array, name="getVisual"),
    path('getAudio', get_audio_array, name="getAudio"),
    path('getTotal', get_total_array, name="getTotal"),
    path('getIds', get_ids, name="getIds"),
    path('getClients', get_clients, name="getClients"),
    path('getFrames', get_frames, name="getFrames"),
    path('getFrame/<int:id>', get_frame, name="getFrame"),
    path('getVisualModality/<int:id>', get_visual_modality, name="getVisualModality"),
    path('getAudioModality/<int:id>', get_audio_modality, name="getAudioModality"),
    path('getTotalModality/<int:id>', get_total_modality, name="getTotalModality"),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Kod datoteke models.py:

```
from django.db import models


class Clients(models.Model):
    uid = models.CharField(max_length=50, default='')
    nickname = models.CharField(max_length=32, default='')
    first_log = models.PositiveBigIntegerField(default=0)

    def __str__(self):
        return f"{self.nickname}"


class Frames(models.Model):
    client = models.ForeignKey(Clients, on_delete=models.CASCADE)
    timestamp = models.PositiveBigIntegerField(default=0)
    ip = models.CharField(max_length=255, default='')

    def __str__(self):
```

```
        return f"Id: {self.id}, Client: {self.client}, at: {self.timestamp}"


class CustomIntField(models.PositiveSmallIntegerField):

    def from_db_value(self, value, expression, connection):
        return int(value/100.0)


class VisualModality(models.Model):
    frame = models.OneToOneField(Frames, on_delete=models.CASCADE)
    angry = CustomIntField(default=0)
    sad = CustomIntField(default=0)
    neutral = CustomIntField(default=0)
    surprised = CustomIntField(default=0)
    fear = CustomIntField(default=0)
    disguised = CustomIntField(default=0)
    happy = CustomIntField(default=0)

    def __str__(self):
        return f"{self.frame}"


class AcousticModality(models.Model):
    frame = models.OneToOneField(Frames, on_delete=models.CASCADE)
    angry = CustomIntField(default=0)
    sad = CustomIntField(default=0)
    neutral = CustomIntField(default=0)
    surprised = CustomIntField(default=0)
    fear = CustomIntField(default=0)
    disguised = CustomIntField(default=0)
    happy = CustomIntField(default=0)

    def __str__(self):
        return f"{self.frame}"


class MultiModality(models.Model):
    frame = models.OneToOneField(Frames, on_delete=models.CASCADE)
    angry = CustomIntField(default=0)
    sad = CustomIntField(default=0)
    neutral = CustomIntField(default=0)
    surprised = CustomIntField(default=0)
    fear = CustomIntField(default=0)
    disguised = CustomIntField(default=0)
    happy = CustomIntField(default=0)

    def __str__(self):
        return f"{self.frame}"
```

Kod datoteke views.py:

```
from __future__ import print_function
from django.shortcuts import render, get_object_or_404
```

```
from django.http import JsonResponse
from Website.models import VisualModality, Frames, Clients, AcousticModality,
MultiModality
import sys


def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)


def homepage(request):
    return render(request, "Main/homePage.html")


# send initial Frame when loading emotional state page
def emotional_state(request, id):
    Frame = get_object_or_404(Frames, pk=id)
    return render(request, "Main/emotionalState.html", {"Frame": Frame})


# get arrays for line graph
def get_visual_array(request):
    VisualArray = list(VisualModality.objects.values())
    return JsonResponse(VisualArray, safe=False)


def get_audio_array(request):
    AudioArray = list(AcousticModality.objects.values())
    return JsonResponse(AudioArray, safe=False)


def get_total_array(request):
    TotalArray = list(MultiModality.objects.values())
    return JsonResponse(TotalArray, safe=False)


# get list of frames for home page
def get_frames(request):
    FrameList = list(Frames.objects.values())
    return JsonResponse(FrameList, safe=False)


# get list of clients to find clients name from id
def get_clients(request):
    ClientList = list(Clients.objects.values())
    return JsonResponse(ClientList, safe=False)


# get single frame emotional state
def get_visual_modality(request, id):
    VisualList = list(VisualModality.objects.filter(frame_id=id).values())
    return JsonResponse(VisualList, safe=False)


def get_audio_modality(request, id):
```

```
    AudioList = list(AcousticModality.objects.filter(frame_id=id).values())
    return JsonResponse(AudioList, safe=False)


def get_total_modality(request, id):
    TotalList = list(MultiModality.objects.filter(frame_id=id).values())
    return JsonResponse(TotalList, safe=False)


# get id list for previous/next button and line graph
def get_ids(request):
    idList = list(MultiModality.objects.values_list('frame_id', flat=True))
    return JsonResponse(idList, safe=False)


# get selected frame
def get_frame(request, id):
    Frame = list(Frames.objects.filter(id=id).values())
    return JsonResponse(Frame, safe=False)
```

Kod datoteke homePage.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>PLEA</title>

    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
    />
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.min.js"></script>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <!-- data table script and css -->
    <link
      rel="stylesheet"
      type="text/css"
      href="https://cdn.datatables.net/1.10.24/css/jquery.dataTables.min.css"
    />
    <script
src="https://cdn.datatables.net/1.10.24/js/jquery.dataTables.min.js"></script>
    <script src="https://momentjs.com/downloads/moment.js"></script>

    {% load static %}
    <link rel="stylesheet" href="{% static 'main.css' %}" />
  </head>
  <body>
    <nav class="navbar navbar-expand-md bg-dark navbar-dark">
      <ul class="navbar-nav">
        <li class="nav-item">
```

```
          <a href="" class="nav-link">Home page</a>
        </li>
        <li class="nav-item">
          <a
            class="nav-link"
            href="https://www.art-ai.io/programme/plea/"
            target="_blank"
            >About</a
          >
        </li>
        <li class="nav-item">
          <a class="nav-link" href="https://pro11.fsb.hr/" target="_blank">PLEA</a>
        </li>
      </ul>
    </nav>

    <div class="container-fluid" style="padding-top: 30px">
      <div class="container colorTable" style="padding: 15px">
        <h3 style="text-align: center">Emotional states</h3>
        <table id="table_id" class="display" width="100%">
          <thead>
            <tr>
              <th>Frame ID</th>
              <th>Date</th>
              <th>ip</th>
              <th>Client's nickname</th>
            </tr>
          </thead>
        </table>
      </div>
    </div>
  </body>

  <script src="{% static 'tableFill.js' %}"></script>
</html>
```

Kod datoteke emotionalState.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Emotional state</title>

    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
    />
    <script src="https://d3js.org/d3.v4.min.js"></script>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://momentjs.com/downloads/moment.js"></script>
    <!-- Function for radial charts: -->
    <script src="https://cdn.jsdelivr.net/gh/holtzy/D3-graph-gallery@master/LIB/d3-
```

```
scale-radial.js"></script>

    {% load static %}
    <link rel="stylesheet" href="{% static 'main.css' %}" />
    <script src="{% static 'graphColors.js' %}"></script>
  </head>
  <body>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a href="{% url 'homePage' %}" class="nav-link">Home page</a>
        </li>
        <li class="nav-item">
          <a
            class="nav-link"
            href="https://www.art-ai.io/programme/plea/"
            target="_blank"
            >About</a
          >
        </li>
        <li class="nav-item">
          <a class="nav-link" href="https://pro11.fsb.hr/" target="_blank"
            >PLEA</a
          >
        </li>
      </ul>
    </nav>

    <div class="container-fluid">
      <div class="row">
        <div class="col-md-6" style="text-align: center">
          <p
            class="font-weight-bold"
            style="margin: 6px; margin-top: 12px; font-size: 12px"
          >
            Date: <span id="frameTime">{{Frame.timestamp}}</span>
          </p>
          <p
            class="font-weight-bold"
            style="margin: 0px; font-size: 10px; display: none"
          >
            Client's ID: <span id="clientId">{{Frame.client_id}}</span>
          </p>
          <p class="font-weight-bold" style="margin: 0px; font-size: 10px">
            Client's nickname: <span id="clientName"></span>
          </p>
          <p style="display: none">
            <input type="text" id="frameId" value="{{Frame.id}}" />
          </p>
          <div class="row">
            <div class="col">
              <div class="card cardPrimary" style="text-align: left">
                <div class="card-body" style="font-size: 15.9px">
                  <b>Emotional state:</b>
                  <ul style="margin-top: 10px; padding-left: 20px">
```

```
                <li><b>Happy:</b> <span id="HappySpan"></span>%</li>
                <li><b>Sad:</b> <span id="SadSpan"></span>%</li>
                <li><b>Angry:</b> <span id="AngrySpan"></span>%</li>
                <li><b>Surprised:</b> <span id="SurprisedSpan"></span>%</li>
                <li><b>Fear:</b> <span id="FearSpan"></span>%</li>
                <li><b>Disguised:</b> <span id="DisguisedSpan"></span>%</li>
                <li><b>Neutral:</b> <span id="NeutralSpan"></span>%</li>
              </ul>
            </div>
          </div>
        </div>
        <div class="col">
          <div class="card cardPrimary">
            <div
              class="card-body"
              style="text-align: center; padding: 15px"
            >
              <p class="font-weight-bold cardText" style="margin-top: 0px">
                Change displayed state:
              </p>

              <button
                type="button"
                class="btn btn-secondary cardButton"
                id="previousButton"
              >
                Previous
              </button>
              <button
                type="button"
                class="btn btn-secondary cardButton"
                id="nextButton"
              >
                Next
              </button>

              <p class="font-weight-bold cardText">Select modality:</p>

              <button
                type="button"
                class="btn btn-secondary cardButton"
                id="AudioModality"
              >
                Audio
              </button>
              <button
                type="button"
                class="btn btn-secondary cardButton"
                id="VisualModality"
              >
                Visual
              </button>
              <button
                type="button"
                class="btn btn-secondary cardButton"
```

```
                  id="TotalModality"
                >
                  Total
                </button>

                <p class="font-weight-bold cardText">
                  Select number of states to be displayed in line graph:
                </p>
                <select
                  class="form-select btn-secondary"
                  aria-label="Default select example"
                  style="width: 50px; font-size: 13px; padding: 3px"
                  id="lineGraphStatesNumber"
                >
                  <option selected>15</option>
                  <option>25</option>
                  <option>50</option>
                </select>
              </div>
            </div>
          </div>
        </div>

        <div class="col-md-6" style="text-align: center; padding-top: 15px">
          <div class="btn-group">
            <button type="button" class="btn btn-secondary" id="barButton">
              Bar Chart
            </button>
            <button type="button" class="btn btn-secondary" id="pieButton">
              Pie Chart
            </button>
            <button type="button" class="btn btn-secondary" id="polarButton">
              Polar Area Chart
            </button>
          </div>

          <div class="row" style="margin-top: 15px; height: 300px">
            <div
              class="col-md-4"
              style="padding: 0px; padding-bottom: 15px; text-align: center"
            >
              <div class="card cardLegend">
                <div
                  class="card-body legendDiv"
                  id="legendDiv"
                  style="padding: 0px"
                ></div>
              </div>
            </div>
            <div
              class="col-md-8"
              id="chartDiv"
              style="padding: 0px; text-align: center"
            ></div>
```

```
          </div>
        </div>
      </div>

      <div class="row">
        <div class="col-md-12" style="padding: 15px; text-align: center">
          <div class="card" style="width: 100%; display: inline-block">
            <div class="card-header" style="text-align: left">
              <select id="selectButton" class="colorMain"></select>
              <select
                id="selectButton2"
                style="display: none; margin-left: 10px"
                class="colorMain"
              ></select>
              <button
                id="2ndLineAdd"
                class="colorMain"
                style="
                  font-size: 14.7px;
                  padding-top: 0.5px;
                  padding-bottom: 0px;
                  margin-left: 15px;
                "
              >
                +
              </button>
              <div style="display: inline-block" id="lineHeader"></div>
              <p style="display: none">
                <button id="updateButton"></button>
                <button id="updateButton2"></button>
              </p>
            </div>
            <div
              class="card-body text-primary"
              id="lineDiv"
              style="margin: 15px; padding: 0px"
            ></div>
          </div>
        </div>
      </div>
    </div>
  </body>
  <script src="{% static 'charts.js' %}"></script>
  <script src="{% static 'buttonEvents.js' %}"></script>
  <script src="{% static 'DateClientFill.js' %}"></script>
</html>
```

Kod datoteke charts.js

```
var height = 300,
  width = 233,
  radius = 230 / 2,
  innerRadius = 5,
  chartData = [
```

```
    $('#HappySpan').text(),
    $('#SadSpan').text(),
    $('#AngrySpan').text(),
    $('#SurprisedSpan').text(),
    $('#FearSpan').text(),
    $('#DisguisedSpan').text(),
    $('#NeutralSpan').text(),
  ];

//functions for legend and creating charts
function Legend() {
  // create Legend svg
  var svgLabel = d3
    .selectAll('.legendDiv')
    .append('svg')
    .attr('width', '100%')
    .attr('height', height);

  //create colored rectangles for legend
  svgLabel
    .selectAll('rect')
    .data(chartData)
    .enter()
    .append('rect')
    .attr('x', 20)
    .attr('y', function (d, i) {
      return 20 + i * 19;
    })
    .attr('width', 14.5)
    .attr('height', 15)
    .attr('fill', function (d, i) {
      return colorSelect(i);
    });

  //add text description to rectangles
  svgLabel
    .selectAll('text')
    .data(chartData)
    .enter()
    .append('text')
    .text(function (d, i) {
      return textSelect(i);
    })
    .attr('x', 41)
    .attr('y', function (d, i) {
      return 33.5 + i * 19;
    })
    .attr('font-family', 'sans-serif');
}

function BarChart() {
  //scale height of bars
  var yScaled = d3.scaleLinear().domain([0, 100]).range([0, 280]);

  // create svg for chart
```

```
  var svgChart = d3
    .select('#chartDiv')
    .append('svg')
    .attr('width', '100%')
    .attr('height', height)
    .attr('id', 'chart');

  //fill svg with bars
  svgChart
    .selectAll('#chart')
    .data(chartData)
    .enter()
    .append('rect')
    .attr('x', function (d, i) {
      return i * 14.142857 + 1 + '%'; //width of each bar is 13.142857% (7 bars + 8x1%
for padding = 100%)
    })
    .attr('y', function (d) {
      return height - yScaled(d);
    })
    .attr('width', '13.142857%')
    .attr('height', function (d) {
      return yScaled(d);
    })
    .attr('fill', function (d, i) {
      return colorSelect(i);
    });

  //fill svg with shadow bars
  svgChart
    .selectAll('#chart')
    .data(chartData)
    .enter()
    .append('rect')
    .attr('x', function (d, i) {
      return i * 14.142857 + 1 + '%'; //width of each bar is 13.142857% (7 bars + 8x1%
for padding = 100%)
    })
    .attr('y', function (d) {
      return height - yScaled(100);
    })
    .attr('width', '13.142857%')
    .attr('height', function (d) {
      return yScaled(100);
    })
    .attr('fill', function (d, i) {
      return colorSelect(i);
    })
    .style('opacity', 0.08);
}

function PieChart() {
  var svg = d3
    .select('#chartDiv')
    .append('svg')
```

```
    .attr('width', width)
    .attr('height', height)
    .attr('id', 'chart')
    .append('g')
    .attr('transform', 'translate(' + width / 2 + ',' + height / 2 + ')');

  var pie = d3.pie().value(function (d) {
    return d.value;
  });
  var data_ready = pie(d3.entries(chartData));

  svg
    .selectAll('#chart')
    .data(data_ready)
    .enter()
    .append('path')
    .attr('d', d3.arc().innerRadius(0).outerRadius(radius))
    .attr('fill', function (d, i) {
      return colorSelect(i);
    })
    .attr('stroke', 'black')
    .style('stroke-width', '1.5px');
}

function PolarChart() {
  //create svg for chart
  var svg = d3
    .select('#chartDiv')
    .append('svg')
    .attr('width', width)
    .attr('height', height)
    .attr('id', 'chart')
    .append('g')
    .attr('transform', 'translate(' + width / 2 + ',' + height / 2 + ')');

  //scale radius
  var y = d3.scaleRadial().range([innerRadius, radius]).domain([0, 100]);

  //add slices
  svg
    .append('g')
    .selectAll('path')
    .data(chartData)
    .enter()
    .append('path')
    .attr('fill', function (d, i) {
      return colorSelect(i);
    })
    .attr(
      'd',
      d3
        .arc()
        .innerRadius(innerRadius)
        .outerRadius(function (d) {
          return y(d);
```

```
        })
        .startAngle(function (d, i) {
          return ((i * 51.4285714286 - 51.4285714286 / 2) * Math.PI) / 180;
        }) //51.4285714286*7=360 degree
        .endAngle(function (d, i) {
          return (
            (((i + 1) * 51.4285714286 - 51.4285714286 / 2) * Math.PI) / 180
          );
        })
        .padAngle(0.1)
        .padRadius(innerRadius)
    );

  //add shadow fills
  svg
    .append('g')
    .selectAll('path')
    .data(chartData)
    .enter()
    .append('path')
    .attr('fill', function (d, i) {
      return colorSelect(i);
    })
    .attr(
      'd',
      d3
        .arc()
        .innerRadius(innerRadius)
        .outerRadius(radius)
        .startAngle(function (d, i) {
          return ((i * 51.4285714286 - 51.4285714286 / 2) * Math.PI) / 180;
        })
        .endAngle(function (d, i) {
          return (
            (((i + 1) * 51.4285714286 - 51.4285714286 / 2) * Math.PI) / 180
          );
        })
        .padAngle(0.1)
        .padRadius(innerRadius)
    )
    .style('opacity', 0.08);
}

//create Legend and Pie Chart when page loads
PieChart();
Legend();

// vars for data arrays
var happyArray;
var sadArray;
var angryArray;
var surpriseArray;
var fearArray;
var disgustArray;
var neutralArray;
```

```
var idArray;

// get emotion states and put them in arrays
$.ajax({
  url: '/getTotal',
  method: 'GET',
  success: function (data) {
    happyArray = data.map(function (x) {
      return x.happy;
    });
    sadArray = data.map(function (x) {
      return x.sad;
    });
    angryArray = data.map(function (x) {
      return x.angry;
    });
    surpriseArray = data.map(function (x) {
      return x.surprised;
    });
    fearArray = data.map(function (x) {
      return x.fear;
    });
    disgustArray = data.map(function (x) {
      return x.disguised;
    });
    idArray = data.map(function (x) {
      return x.id;
    });
    neutralArray = data.map(function (x) {
      return x.neutral;
    });
    // initial line graph function only after are arrays are get from server
    lineGraphFun();
  },

  error: function (error) {
    console.log(error);
  },
});

// function for drawing line graph
function lineGraphFun() {
  // initial lines data
  data = lineData(happyArray);
  data2 = lineData(sadArray);

  // graph dimensions and margins
  var margin = { top: 30, right: 20, bottom: 20, left: 40 },
    width, // width is defined later
    height = 250 - margin.top - margin.bottom;

  // x and y axis scale
  var xScale = d3
    .scaleLinear()
    .range([0, width])
```

```
    .domain(
      d3.extent(data, function (d, i) {
        return i + 1;
      })
    );
  var yScale = d3.scaleLinear().range([height, 0]).domain([0, 100]);

  // define axes
  var xAxis = d3.axisBottom().scale(xScale);
  var yAxis = d3.axisLeft().scale(yScale);

  // create line
  var line = d3.line();

  // create svg for graph
  var svg = d3
    .select('#lineDiv')
    .append('svg')
    .attr('height', height + margin.top + margin.bottom);

  // translate for margin space
  var lineGraph = svg
    .append('g')
    .attr('transform', 'translate(' + margin.left + ',' + margin.top + ')');

  // draw the lines from data
  var path = lineGraph
    .append('path')
    .data([data])
    .style('fill', 'none')
    .style('stroke', 'steelblue')
    .style('stroke-width', '2px')
    .attr('id', 'path1');

  var path2 = lineGraph
    .append('path')
    .data([data2])
    .style('fill', 'none')
    .style('stroke', '#9b1c31')
    .style('stroke-width', '2px')
    .attr('display', 'none')
    .attr('id', 'path2');

  // x axis
  var xAxisEl = lineGraph
    .append('g')
    .attr('transform', 'translate(0,' + height + ')');

  // y axis
  var yAxisEl = lineGraph.append('g').call(yAxis);

  // function for drawing chart
  function drawChart() {
    // reset the width
    width =
```

```
      parseInt(d3.select('#lineDiv').style('width'), 10) -
      margin.left -
      margin.right;

    // reset svg width
    svg.attr('width', width + margin.left + margin.right);

    // reset x range and draw new axis
    xScale.range([0, width]);
    xAxis.scale(xScale);
    xAxisEl.call(xAxis);

    // define new line path and draw it
    line
      .x(function (d, i) {
        return xScale(i + 1);
      })
      .y(function (d) {
        return yScale(d);
      });

    path.attr('d', line(data));
    path2.attr('d', line(data2));

    // current state mark
    d3.selectAll('#pointer').remove();
    lineGraph
      .append('g')
      .selectAll('rect')
      .data(angryArray)
      .enter()
      .append('rect')
      .attr('x', xScale(indexNr + 1) - 0.5)
      .attr('y', yScale(100))
      .attr('width', 1)
      .attr('height', 199)
      .attr('fill', '#F8F6F6')
      .attr('id', 'pointer')
      .style('opacity', 0.2);

    d3.selectAll('#pointerText').remove();
    lineGraph
      .append('g')
      .selectAll('rect')
      .data(angryArray)
      .enter()
      .append('text')
      .text('CURRENT STATE')
      .attr('x', xScale(indexNr + 1) - 31)
      .attr('y', yScale(105))
      .attr('id', 'pointerText')
      .style('font-size', '8px');
  }

  // hide/show 2nd path
```

```
$('#2ndLineAdd').click(function () {
  // toggle 2nd line, line header and select2 button
  $('#path2').toggle();
  $('#selectButton2').toggle();
  $('#lineHeader').toggle();

  // change +/- sign
  if ($('#2ndLineAdd').text() == '+') {
    $('#2ndLineAdd').text('-');
  } else {
    $('#2ndLineAdd').text('+');
  }

  // set option 2 different then option 1
  if ($('#selectButton').val() == 'Happy') {
    $('#selectButton2').val('Sad');
  } else {
    $('#selectButton2').val('Happy');
  }
  update2($('#selectButton2').val());
  update($('#selectButton').val());
});
$('#2ndLineAdd').text('+');

// select button options
var selectOptions = [
  'Happy',
  'Sad',
  'Angry',
  'Surprised',
  'Fear',
  'Disgust',
  'Neutral',
];

d3.selectAll('#selectButton, #selectButton2')
  .selectAll('myOptions')
  .data(selectOptions)
  .enter()
  .append('option')
  .text(function (d) {
    return d;
  })
  .attr('value', function (d) {
    return d;
  });

// getting chosen data array
function dataSelect(selectedOption) {
  if (selectedOption == 'Happy') {
    return happyArray;
  } else if (selectedOption == 'Sad') {
    return sadArray;
  } else if (selectedOption == 'Angry') {
    return angryArray;
```

```
    } else if (selectedOption == 'Surprised') {
      return surpriseArray;
    } else if (selectedOption == 'Fear') {
      return fearArray;
    } else if (selectedOption == 'Disgust') {
      return disgustArray;
    } else {
      return neutralArray;
    } //neutral
  }

  // update on data change
  function update(selectedGroup) {
    // update path with new data
    path
      .transition()
      .duration(1000)
      .attr('d', line(lineData(dataSelect(selectedGroup))));

    // disable other options in selector
    var op = document
      .getElementById('selectButton2')
      .getElementsByTagName('option');
    for (var i = 0; i < op.length; i++) {
      if (op[i].value == selectedGroup) {
        op[i].disabled = true;
      } else {
        op[i].disabled = false;
      }
    }

    d3.select('#LL').remove();
    d3.select('#LL2').remove();
    lineLegend();
    // data for resized path
    data = lineData(dataSelect(selectedGroup));
  }

  function update2(selectedGroup) {
    // update path with new data
    path2
      .transition()
      .duration(1000)
      .attr('d', line(lineData(dataSelect(selectedGroup))));

    // disable other options in selector
    var op = document
      .getElementById('selectButton')
      .getElementsByTagName('option');
    for (var i = 0; i < op.length; i++) {
      if (op[i].value == selectedGroup) {
        op[i].disabled = true;
      } else {
        op[i].disabled = false;
      }
```

```
    }

    d3.select('#LL').remove();
    d3.select('#LL2').remove();
    lineLegend();
    // data for resized path
    data2 = lineData(dataSelect(selectedGroup));
  }

  $('#updateButton').click(function () {
    update($('#selectButton').val());
    update2($('#selectButton2').val());
    drawChart();
  });

  // update on select button change
  d3.select('#selectButton').on('change', function (d) {
    update(d3.select(this).property('value'));
  });

  d3.select('#selectButton2').on('change', function (d) {
    update2(d3.select(this).property('value'));
  });

  // update on line graph state number change
  $('#lineGraphStatesNumber').on('change', function () {
    // redefine x scale
    xScale = d3
      .scaleLinear()
      .range([0, width])
      .domain(
        d3.extent(
          lineData(dataSelect($('#selectButton').val())),
          function (d, i) {
            return i + 1;
          }
        )
      );

    // redefine x axis and update
    xAxis = d3.axisBottom().scale(xScale);
    drawChart();
    update($('#selectButton').val());
    update2($('#selectButton2').val());
  });

  // draw initial chart
  drawChart();
  lineLegend();

  // redraw chart on resize
  window.addEventListener('resize', drawChart);
}

// function for line Legend
```

```
function lineLegend() {
  // svg for legend1
  var lineLeg = d3
    .selectAll('#lineHeader')
    .append('svg')
    .attr('width', '120px')
    .attr('height', '30px')
    .attr('id', 'LL');

  // rectangle for legend2
  lineLeg
    .append('rect')
    .attr('x', 25)
    .attr('y', 5)
    .attr('width', 15)
    .attr('height', 15)
    .attr('fill', 'steelblue');

  // text1 for legend
  lineLeg
    .append('text')
    .text($('#selectButton').val())
    .attr('x', 45)
    .attr('y', 18)
    .attr('font-family', 'sans-serif');

  // svg for legend2
  var lineLeg2 = d3
    .selectAll('#lineHeader')
    .append('svg')
    .attr('width', '120px')
    .attr('height', '30px')
    .attr('id', 'LL2');

  // rectangle for legend2
  lineLeg2
    .append('rect')
    .attr('x', 25)
    .attr('y', 5)
    .attr('width', 15)
    .attr('height', 15)
    .attr('fill', '#9b1c31');

  // text for legend2
  lineLeg2
    .append('text')
    .text($('#selectButton2').val())
    .attr('x', 45)
    .attr('y', 18)
    .attr('font-family', 'sans-serif');
}

// function for getting chosen length of data
var indexNr;
function lineData(inputData) {
```

```
  var newData;
  var lineGraphStatesNumber = $('#lineGraphStatesNumber').val();
  if (lineGraphStatesNumber == 50) {
    lineGraphStatesNumber = 51;
  } // to be even number of states on each side
  indexNr = idArray.findIndex(function findId(id) {
    return id == $('#frameId').val();
  });

  // indexNr is index in new data set for displaying current state line
  if (lineGraphStatesNumber > inputData.length) {
    newData = inputData;
  } else if (indexNr - (lineGraphStatesNumber - 1) / 2 < 0) {
    newData = inputData.slice(0, lineGraphStatesNumber);
  } else if (indexNr + (lineGraphStatesNumber - 1) / 2 > inputData.length) {
    newData = inputData.slice(
      inputData.length - lineGraphStatesNumber,
      inputData.length + 1
    );
    indexNr = lineGraphStatesNumber - inputData.length + indexNr;
  } else {
    newData = inputData.slice(
      indexNr - (lineGraphStatesNumber - 1) / 2,
      indexNr + (lineGraphStatesNumber - 1) / 2 + 1
    );
    indexNr = (lineGraphStatesNumber - 1) / 2;
  }
  return newData;
}

// line header is toggled off until 2nd line is shown
$('#lineHeader').toggle();
```

Kod datoteke buttonEvents.js

```
// initial chart and modality
var selectedBar = 'Pie';
var selectedModality = 'Total';

//on click delete old svg chart and create new
$('#barButton').click(function () {
  d3.select('#chart').remove();
  BarChart();
  selectedBar = 'Bar';

  $('#barButton').css('background-color', '#394651');
  $('#pieButton').css('background-color', '#6C757D');
  $('#polarButton').css('background-color', '#6C757D');
});

$('#pieButton').click(function () {
  d3.select('#chart').remove();
  PieChart();
  selectedBar = 'Pie';
```

```
  $('#barButton').css('background-color', '#6C757D');
  $('#pieButton').css('background-color', '#394651');
  $('#polarButton').css('background-color', '#6C757D');
});

$('#polarButton').click(function () {
  d3.select('#chart').remove();
  PolarChart();
  selectedBar = 'Polar';

  $('#barButton').css('background-color', '#6C757D');
  $('#pieButton').css('background-color', '#6C757D');
  $('#polarButton').css('background-color', '#394651');
});

//change modality
$('#AudioModality').click(function () {
  $.ajax({
    url: '/getAudioModality/' + $('#frameId').val(),
    method: 'GET',
    success: function (data) {
      $('#HappySpan').text(data[0].happy);
      $('#SadSpan').text(data[0].sad);
      $('#AngrySpan').text(data[0].angry);
      $('#SurprisedSpan').text(data[0].surprised);
      $('#FearSpan').text(data[0].fear);
      $('#DisguisedSpan').text(data[0].disguised);
      $('#NeutralSpan').text(data[0].neutral);

      chartData = [
        $('#HappySpan').text(),
        $('#SadSpan').text(),
        $('#AngrySpan').text(),
        $('#SurprisedSpan').text(),
        $('#FearSpan').text(),
        $('#DisguisedSpan').text(),
        $('#NeutralSpan').text(),
      ];

      selectedModality = 'Audio';
      updateChart();

      $('#AudioModality').css('background-color', '#394651');
      $('#VisualModality').css('background-color', '#6C757D');
      $('#TotalModality').css('background-color', '#6C757D');

      // get Audio modality array for line graph and update it
      $.ajax({
        url: '/getAudio',
        method: 'GET',
        success: function (data) {
          happyArray = data.map(function (x) {
            return x.happy;
          });
```

```
          sadArray = data.map(function (x) {
            return x.sad;
          });
          angryArray = data.map(function (x) {
            return x.angry;
          });
          surpriseArray = data.map(function (x) {
            return x.surprised;
          });
          fearArray = data.map(function (x) {
            return x.fear;
          });
          disgustArray = data.map(function (x) {
            return x.disguised;
          });
          idArray = data.map(function (x) {
            return x.id;
          });
          neutralArray = data.map(function (x) {
            return x.neutral;
          });

          $('#updateButton').click();
        },

        error: function (error) {
          console.log(error);
        },
      });
    },
    error: function (error) {
      console.log(error);
    },
  });
});

$('#VisualModality').click(function () {
  $.ajax({
    url: '/getVisualModality/' + $('#frameId').val(),
    method: 'GET',
    success: function (data) {
      $('#HappySpan').text(data[0].happy);
      $('#SadSpan').text(data[0].sad);
      $('#AngrySpan').text(data[0].angry);
      $('#SurprisedSpan').text(data[0].surprised);
      $('#FearSpan').text(data[0].fear);
      $('#DisguisedSpan').text(data[0].disguised);
      $('#NeutralSpan').text(data[0].neutral);

      chartData = [
        $('#HappySpan').text(),
        $('#SadSpan').text(),
        $('#AngrySpan').text(),
        $('#SurprisedSpan').text(),
        $('#FearSpan').text(),
```

```
        $('#DisguisedSpan').text(),
        $('#NeutralSpan').text(),
      ];

      selectedModality = 'Visual';
      updateChart();

      $('#AudioModality').css('background-color', '#6C757D');
      $('#VisualModality').css('background-color', '#394651');
      $('#TotalModality').css('background-color', '#6C757D');

      // get Visual modality array for line graph and update it
      $.ajax({
        url: '/getVisual',
        method: 'GET',
        success: function (data) {
          happyArray = data.map(function (x) {
            return x.happy;
          });
          sadArray = data.map(function (x) {
            return x.sad;
          });
          angryArray = data.map(function (x) {
            return x.angry;
          });
          surpriseArray = data.map(function (x) {
            return x.surprised;
          });
          fearArray = data.map(function (x) {
            return x.fear;
          });
          disgustArray = data.map(function (x) {
            return x.disguised;
          });
          idArray = data.map(function (x) {
            return x.id;
          });
          neutralArray = data.map(function (x) {
            return x.neutral;
          });

          $('#updateButton').click();
        },

        error: function (error) {
          console.log(error);
        },
      });
    },
    error: function (error) {
      console.log(error);
    },
  });
});
```

```
$('#TotalModality').click(function () {
  $.ajax({
    url: '/getTotalModality/' + $('#frameId').val(),
    method: 'GET',
    success: function (data) {
      $('#HappySpan').text(data[0].happy);
      $('#SadSpan').text(data[0].sad);
      $('#AngrySpan').text(data[0].angry);
      $('#SurprisedSpan').text(data[0].surprised);
      $('#FearSpan').text(data[0].fear);
      $('#DisguisedSpan').text(data[0].disguised);
      $('#NeutralSpan').text(data[0].neutral);

      chartData = [
        $('#HappySpan').text(),
        $('#SadSpan').text(),
        $('#AngrySpan').text(),
        $('#SurprisedSpan').text(),
        $('#FearSpan').text(),
        $('#DisguisedSpan').text(),
        $('#NeutralSpan').text(),
      ];

      selectedModality = 'Total';
      updateChart();

      $('#AudioModality').css('background-color', '#6C757D');
      $('#VisualModality').css('background-color', '#6C757D');
      $('#TotalModality').css('background-color', '#394651');

      // get Total modality array for line graph and update it
      $.ajax({
        url: '/getTotal',
        method: 'GET',
        success: function (data) {
          happyArray = data.map(function (x) {
            return x.happy;
          });
          sadArray = data.map(function (x) {
            return x.sad;
          });
          angryArray = data.map(function (x) {
            return x.angry;
          });
          surpriseArray = data.map(function (x) {
            return x.surprised;
          });
          fearArray = data.map(function (x) {
            return x.fear;
          });
          disgustArray = data.map(function (x) {
            return x.disguised;
          });
          idArray = data.map(function (x) {
            return x.id;
```

```
          });
          neutralArray = data.map(function (x) {
            return x.neutral;
          });

          $('#updateButton').click();
        },

        error: function (error) {
          console.log(error);
        },
      });
    },
    error: function (error) {
      console.log(error);
    },
  });
});

$('#TotalModality').click();

// update chart after modality has been changed
function updateChart() {
  if (selectedBar == 'Pie') {
    $('#pieButton').click();
  } else if (selectedBar == 'Bar') {
    $('#barButton').click();
  } else {
    $('#polarButton').click();
  }
}

//switch state buttons
var idList;
var indexNumber;
$('#previousButton').click(function () {
  $.ajax({
    url: '/getIds',
    method: 'GET',
    success: function (data) {
      idList = data;
      indexNumber = idList.findIndex(function findId(id) {
        return id == $('#frameId').val();
      });
      if (indexNumber > 0) {
        indexNumber = idList[indexNumber - 1];
        $('#frameId').val(indexNumber);
        updateFrame();
      } else {
        alert('This is the first state!');
      }
    },
    error: function (error) {
      console.log(error);
    },
```

```
  });
});

$('#nextButton').click(function () {
  $.ajax({
    url: '/getIds',
    method: 'GET',
    success: function (data) {
      idList = data;
      indexNumber = idList.findIndex(function findId(id) {
        return id == $('#frameId').val();
      });
      if (indexNumber < idList.length - 1) {
        indexNumber = idList[indexNumber + 1];
        $('#frameId').val(indexNumber);
        updateFrame();
      } else {
        alert('This is the last state!');
      }
    },
    error: function (error) {
      console.log(error);
    },
  });
});

// update frame after going to previous or next one
function updateFrame() {
  $.ajax({
    url: '/getFrame/' + $('#frameId').val(),
    method: 'GET',
    success: function (data) {
      var date = new Date(data[0].timestamp);
      date = moment(date).format('YYYY-MM-DD  hh:mm:ss');
      $('#frameTime').text(date);
      $('#clientId').text(data[0].client_id);
      $('#frameId').val(data[0].id);
      refreshState();
      $('#updateButton2').click();
    },
    error: function (error) {
      console.log(error);
    },
  });
}

// update with selected modality after frame change
function refreshState() {
  if (selectedModality == 'Total') {
    $('#TotalModality').click();
  } else if (selectedModality == 'Visual') {
    $('#VisualModality').click();
  } else {
    $('#AudioModality').click();
```

```
  }
}
```

Kod datoteke tableFill.js:

```
//fill the table from database on page load
$(document).ready(function () {
  var Clients;

  $.ajax({
    url: '/getClients',
    method: 'GET',
    success: function (data) {
      Clients = data;
      $('#table_id').DataTable({
        ajax: {
          url: '/getFrames',
          dataSrc: '',
        },

        columns: [
          {
            data: 'id',
            render: function (data, type, row, meta) {
              if (type === 'display') {
                data = '<a href="state/' + row.id + '">' + data + '</a>';
              }
              return data;
            },
          },
          {
            data: 'timestamp',
            render: function (data, type, row, meta) {
              var date = new Date(data);
              return moment(date).format('YYYY-MM-DD hh:mm:ss');
            },
          },
          {
            data: 'ip',
            render: function (data, type, row, meta) {
              return data;
            },
          },
          {
            data: 'client_id',
            render: function (data, type, row, meta) {
              return Clients.find((x) => x.id === data).nickname;
            },
          },
        ],
      });
    },
    error: function (error) {
      console.log(error);
```

```
    },
  });
});
```

Kod datoteke main.css:

```
body {
  font-family: sans-serif;
  background-color: #e2dadb;
}
.colorMain {
  background-color: #e2dadb;
}
.colorTable {
  background-color: #f8f6f6;
}
.card {
  border: 2px solid #9b8487;
  background-color: #bcaeaf;
}
.cardText {
  font-size: 11.9px;
  margin-top: 18px;
  margin-bottom: 5px;
}
.cardButton {
  font-size: 10px;
  padding-left: 10px;
  padding-right: 10px;
}
.cardPrimary {
  margin: 15px;
  height: 250px;
  width: 185px;
  display: inline-block;
}
.cardLegend {
  width: 125px;
  height: 175px;
  padding: 0px;
  display: inline-block;
}
```