UNIVERZITET U NOVOM SADU

TEHNIČKI FAKULTET

„MIHAJLO PUPIN“

ZRENJANIN

DIPLOMSKI RAD

TEMA: JAVA VEB APLIKACIJA ZA EVIDENCIJU UPISA U PREDŠKOLSKU USTANOVU

JAVA WEB APPLICATION FOR PRESCHOOL ENROLLMENT RECORDS

MENTOR: Doc. dr Ljubica Kazi

STUDENT: Ekreš Viktorija
BR. INDEKSA: IT 65/15
SMER: Informacione tehnologije - inženjerstvo

ZRENJANIN, 2019

Sadržaj

1. UVOD 4
2. TEORIJSKE OSNOVE 5
   2.1. Veb aplikacije 5
      2.1.1. Prednosti veb aplikacija 6
      2.1.2. Vrste veb aplikacija 7
   2.2. Višeslojne veb aplikacije 8
      2.2.1. Troslojna arhitektura 8
      2.2.2. MVC (Model View Controller) 10
      2.2.3. Veb servisi 11
3. OPIS PRIMENJENE TEHNOLOGIJE 13
      3.1.1. Spring Boot Framework 13
      3.1.2. Angular 4/5 14
      3.1.3. HTML 14
      3.1.4. CSS 15
      3.1.5. Bootstrap 16
      3.1.6. IntelliJ razvojno okruženje 16
4. ANALIZA SEMANTIČKE OBLASTI 17
   3.1. Rad predškolske ustanove 17
   4.1. Zakon o predškolskom vaspitanju 19
   4.2. Upis dece u predškolsku ustanovu 20
      4.2.1. Pravilnik o bližim uslovima za utvrđivanje prioriteta za upis 20
5. POSTOJEĆA REŠENJA 22
6. REALIZOVAN PRIMER 27
   6.1. Prijava projekta 27
   6.2. UML dijagrami 30
   6.3. Korisničko uputstvo 32
   6.4. Implementacija 39
      6.4.1. Baza podataka 39
      6.4.2. UML modeli za opis implementacije 42
      6.4.3. Tabelarni prikaz višeslojne arhitekture softvera 43
      6.4.4. Ključni delovi programskog koda 45

7. ZAKLJUČAK 56
8. LITERATURA 57

1. Uvod

Jedna od važnih uloga informacionih tehnologija je da organizacijama obezbedi fleskibilnost i prilagodljivost poslovnog sistema, olakšavajući rešavanje određenih problema automatizacijom i najmanjih poslovnih aktivnosti. Danas se većina poslovnih procesa oslanja na neku vrstu računarskog sistema, a zahvaljujući eksponencijalno rastućoj upotrebi interneta organizacije se okreću i ka sistemima baziranim na mreži. Primena automatizovanih informacionih sistema u poslovanju predstavlja kritičan faktor uspeha organizacije u smislu povećanja produktivnosti, efikasnosti i kvaliteta.

Cilj diplomskog rada je da se realizuje prototip veb aplikacije koja bi omogućila unapređenje efikasnosti upravljanja procesom upisa u okviru predškolske ustanove. Snimak stanja, prikupljanje dokumentacije i specifikacija zahteva je urađena na primeru Predškolske ustanove Zrenjanin, koja je uvidela vrednost tehnologije i automatizovanih sistema i napravila značajan korak napred ka uspešnom digitalnom poslovanju.

U skladu sa regulativama i zakonom pomenutim u analizi semantičke oblasti, realizovano je rešenje koje predstavlja višeslojnu veb aplikaciju zasnovanu na REST arhitekturi koristeći MVC (Model View Controller) dizajn patern. Oba ova tehnološka pojma su detaljnije objašnjena u narednoj sekciji. Zatim se u nastavku može videti detaljno prikazan korisnički interfejs izrađen uz pomoć određenih programskih biblioteka za brži i jednostavniji razvoj atraktivnih veb stranica. Iako je izgled aplikacije veoma bitan aspekt razvoja softverskih sistema, najbitnija je njena funkcionalnost, odnosno deo programa koji organizacijama omogućava lakše poslovanje. Ključni delovi koda koji čine ovu aplikaciju funkcionalnom i spremnom za upotrebu su prikazani u poslednjoj sekciji rada.

2. Teorijske osnove

Kao što je i prethodno napomenuto, rastuća upotreba interneta je zauvek promenila način na koji organizacije pokušavaju da opstanu i ostanu konkurentni na tržištu, a pored toga i da ubrzaju određene poslovne procese povećavajući efikasnost i produktivnost. Većina organizacija se danas odlučuje za veb aplikaciju kao deo svog informacionog sistema, za razliku od desktop aplikacija koje su ranije bile zastupljenije u poslovnim sistemima. U nastavku će biti opisane najveće prednosti koje veb aplikacije donose u odnosu na neke druge opcije koje su na raspolaganju, kao i vrste veb aplikacija od kojih svaka sadrži različite funkcionalnosti, način kreiranja i upotrebu.

2.1. Veb aplikacije

Veb aplikacije predstavljaju programe određene namene pokrenute od strane veb servera, kojima korisnici pristupaju pomoću internet pretraživača (Google Chrome, Firefox, Opera itd.) Nazivaju se „veb“ aplikacije iz tih razloga što je potrebna internet konekcija za interakciju sa njima. Nema potrebe da se instaliraju na računarima jer svaki sadrži bar jedan internet pretraživač.

Većina veb aplikacija je zasnovana na klijent-server arhitekturi, gde klijent predstavlja program koji se koristi za pristup aplikaciji i interakciju sa njom (osoba koja koristi pretraživač), dok server predstavlja mesto gde se podaci skladište i preuzimaju. Klijentska strana se takođe naziva i Front End deo aplikacije. Ono što front end deo obuhvata je grafički interfejs koji se može videti, kliknuti, dodirnuti, i u suštini sve što je korisniku na „dohvat ruke“. Iako je Front End često povezivan sa samim izgledom veb stranice, Front End development zahteva mnogo više od samog dizajniranja.

Slika 1. Front End i Back End deo veb aplikacije [1]

Front End programeri koriste HTML (Hyper Text Markup Language) i CSS (Cascading Style Sheets) da stvore vizuelno privlačan izgled stranice (što je veoma bitan segment u današnje vreme gde postoji velik izbor veb stranica) i JavaScript programski jezik kako bi učinili

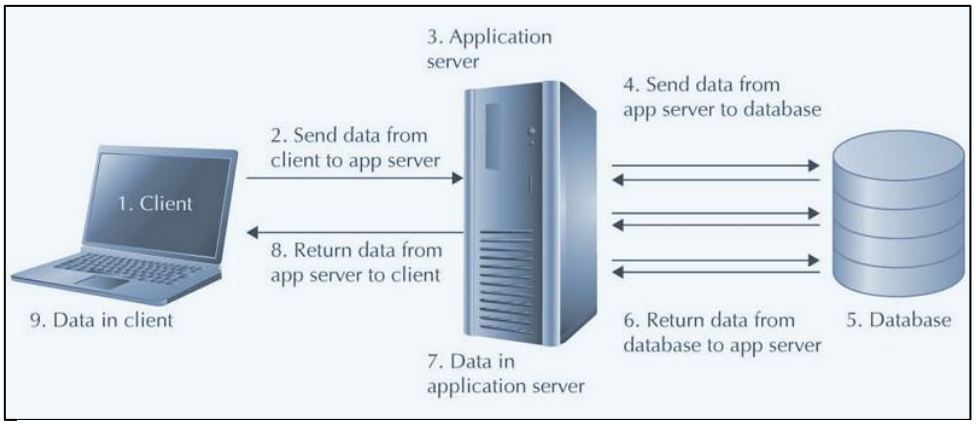
stranicu interaktivnom. Međutim ne bave se bazom podataka i samom funkcionalnošću aplikacije. Time se bavi Back End programer. [2]

Back End je serverska strana aplikacije koja radi u pozadini, i nije vidljiva korisniku. Ona sadrži svu logiku aplikacije i konstantno komunicira sa bazom podataka i sa Front End delom kako bi odgovarala na akcije korisnika i izvršavala određene delove koda. Kod je pisan u nekom od programskih jezika poput Jave, PHP-a, Ruby-ja itd. [3]

Način izvršavanja veb aplikacija je opisan u nastavku.

Veb aplikaciji je potreban veb server za upravljanje i obrađivanje klijentskih zahteva, aplikacioni server za izvršavanje dobijenog zahteva i baza podataka koja služi za skladištenje informacija. [4]

1. Korisnik preko internet pretraživača prosleđuje zahtev veb serveru za nekom određenom informacijom
2. Veb server prosleđuje zahtev odgovarajućem aplikacionom serveru
3. Server izvršava zadati zahtev (izvršava upite nad bazom podataka, obrađuje podatke) zatim generiše rezultat traženih podataka
4. Aplikacioni server vraća traženu informaciju nazad veb serveru
5. Veb server odgovara klijentu tako što mu prezentuje informaciju na grafičkom interfejsu



Slika 2. Prikaz komunikacije između klijenta i servera [5]

2.1.1. Prednosti veb aplikacija

Veb aplikacije sa sobom donose velik broj prednosti. Neke od njih su [6]:

- Interoperabilnost – što znači da se aplikacija može koristiti na svim platformama bez obzira na kojem operativnom sistemu ili uređaju se pokreće. Iako postoje i nativne aplikacije (pravljene za specifičan operativni sistem), uglavnom se veb aplikacije prave tako da budu kompatibilne sa svim uređajima.
- Nije potrebna instalacija – Veb aplikacije se izvršavaju direktno unutar internet pretraživača, tako da nije potrebno da se preuzima i instalira program na računaru.
- Pristup – Jedan od glavnih razloga zašto se organizacije odlučuju za veb aplikaciju je to što se aplikaciji može pristupati bilo gde, sa računara, telefona, tableta i da svi podaci budu sinhronizovani. Jedino je potrebna internet konekcija.

- Budžet – Budžet je veoma bitan aspekt kada se radi o razvoju aplikacija. Često se organizacije odlučuju za veb iz razloga što ne treba da se uloži mnogo novca da bi aplikacija bila prisutna na svim platformama.
- Lakše i jednostavnije je pristupiti aplikaciji korisnicima mobilnih uređaja.

2.1.2. Vrste veb aplikacija

Klasifikacija veb aplikacija se vrši na osnovu načina na koji se sadržaj prikazuje. Prema tome, veb aplikacije se mogu podeliti na 5 tipova.

1. Statička veb aplikacija

Statičke veb aplikacije se izvršavaju unutar internet pretraživača bez potrebe za serverskom stranom, što znači da se kod u celini izvršava na klijentskoj strani. One prikazuju veoma malo sadržaja, interaktivnosti i nisu toliko fleksibilne. Uglavnom su kreirane uz pomoć HTML-a i CSS-a, međutim neki animirani objekti poput banera, GIF-ova i video klipova se mogu naći na ovim stranicama. Takođe se može koristiti jQuery i Ajax za njihov razvoj. Kao što je već spomenuto, ove aplikacije nisu fleksibilne i teško se menja sadržaj na njima. Ukoliko je potrebna promena, jedini način da se to uradi jeste da se ukloni a zatim ponovo postavi na server zajedno sa izmenama. Iz tih razloga održavanje statičkih veb aplikacija je teško, i količina podataka koja se šalje i prima dovodi do rizika za loše performanse. [7] Primeri statičkih veb aplikacija uključuju profesionalni portfolio, reklamne stranice za kompanije itd.

2. Dinamička veb aplikacija

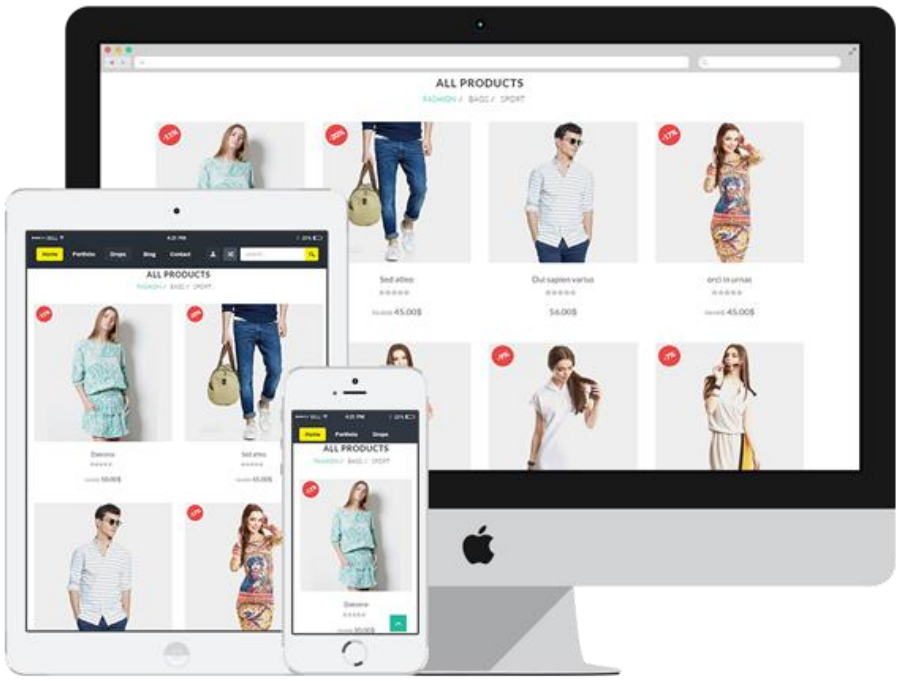
Dinamičke veb aplikacije su na tehničkom nivou daleko komplikovanije od statičkih. Za učitavanje podataka dinamičke veb aplikacije koriste bazu podataka, stoga se sadržaj automatski ažurira pri svakom korisničkom pristupu. Izgled ovakvih aplikacija nije predefinisan, već se dinamički oblikuje logikom koja je definisana na serverskoj strani aplikacije. Uglavnom sadrže administrativni panel (koji se naziva i CMS – Content Management System) odakle administratori mogu da upravljaju sadržajem aplikacije kao što su tekst i slike. Za dinamičke veb stranice se koriste razni programski jezici, najčešće PHP i ASP. [7]

3. Online prodavnica (e-commerce)

Proces razvoja veb aplikacije koja predstavlja online prodavnicu je komplikovaniji iz razloga što se mora uključiti i sam proces plaćanja elektronskim kreditnim karticama, PayPal uslugama ili drugim metodama online plaćanja. Takođe postoji administrativni panel, gde administrator prodavnice može da ažurira proizvode, briše ili dodaje, kao i da upravlja narudžbinama i uplatama. Primer online prodavnice može se videti na slici 3.

4. Content Management System aplikacije

Za lako upravljanje sadržajem na veb stranicama, koristi se CMS – Content Management System. Danas jedan od najkorišćenijih CMS-a na svetu jeste WordPress. Pored WordPressa, često se koriste Joomla i Drupal. Ova vrsta veb aplikacije se najčešće koristi za lične ili profesionalne blogove, stranice za objavljivanje vesti, artikala, medija itd.



Slika 3. Primer online prodavnice [8]

5. Progresivne veb aplikacije

Progresivne veb aplikacije se ne fokusiraju na novim principima arhitekture softvera, već na karakteristikama koje poboljšavaju performanse i prilagodljivost mobilnim uređajima. Keširanje i bolji prenos podataka preko HTTP protokola su ključna poboljšanja. Dakle, jedan od ciljeva progresivnih veb aplikacija jesu da poboljšaju iskustvo korisnicima mobilnih uređaja, kao i korisnicima sa slabijom ili lošijom internet konekcijom. [7]

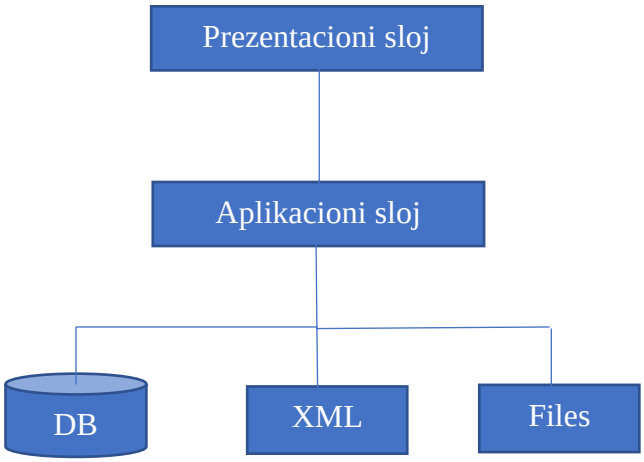
2.2. Višeslojne veb aplikacije

U softverskom inženjerstvu, višeslojna arhitektura (često nazvana i n-tier arhitektura) je klijent-server arhitektura gde su prezentacija, procesiranje aplikacije i funkcije upravljanja podacima fizički odvojene. Najčešće korišćena višeslojna arhitektura je troslojna arhitektura čiji je dijagram prikazan na slici 4. N-tier aplikaciona arhitektura obezbeđuje model softvera gde programeri mogu da kreiraju fleksibilne aplikacije pogodne za održavanje i ponovno korišćenje komponenti. Razdvajanjem aplikacije na slojeve, dobija se mogućnost izmene i dodavanja specifičnih slojeva, umesto da se celokupna aplikacija ponovo implementira zbog promena. U troslojnoj arhitekturi najčešće razlikujemo prezentacioni sloj (presentation tier), sloj domenske logike (domain logic tier) i sloj podataka (data storage tier).

2.2.1. Troslojna arhitektura

U objektno-orijentisanom dizajnu višeslojne arhitekture softverske podrške informacionog sistema, najčešći su slojevi:

- Prezentacioni sloj (presentation layer) – korisnički interfejs, sloj pogleda (view layer)
- Aplikacioni sloj (application layer) – sloj servisa (service layer), controller layer
- Poslovni sloj (Business layer) – sloj poslovne logike
- Sloj pristupa podacima (data access layer)



Slika 4. Dijagram troslojne arhitekture [9]

1) Prezentacioni sloj

Prezentacioni sloj troslojne softverske arhitekture zadužen je za predstavljanje podataka korisniku, kao i za neposredni prijem komandi korisnika u toku korišćenja aplikacije. Osnovne funkcije obuhvataju formatiranje i predstavljanje podataka, kao i organizaciju dostupnosti funkcija kroz personalizaciju funkcija različitim tipovima korisnika. Strukture podataka služe za prijem i predstavljanje podataka i mogu se značajno razlikovati u odnosu na strukture podataka koje se nalaze u poslovnom sloju i sloju podataka. Posebne klase objketno-orijentisane implementacije obezbeđuju odgovarajuće strukture podataka, kao i funkcije prikaza i organizacije funkcija, odnosno prijema komandi sa korisničkog interfejsa. Te klase mogu biti univerzalne pa se mogu koristiti u implementaciji različitih tipova korisničkih interfejsa (desktop, veb, mobilne aplikacije). Takođe, u ovom sloju pripadaju i standardne klase za grafičko uređenje korisničkog interfejsa. [10]

2) Sloj servisa (middleware)

Middleware je računarski softver koji obezbeđuje servise softverskim aplikacijama van onih koje su na raspolaganju od strane operativnog sistema. Olakšava softverskim developerima da implementiraju komunikaciju i ulaz-izlaz, kako bi se mogli fokusirati na specifične namene njihovih aplikacija. Termin je najčešće korišćen u značenju softvera koji omogućuje komunikaciju i upravljanje podacima u distibuiranim aplikacijama. Middleware je definisan i kao „servisi koji su iznad transportnog sloja OSI modela, ali ispod aplikacionog okruženja, odnosno apliacionog API nivoa“. U ovom specifičnom smislu, middleware povezuje klijent i server, odnosno dva peer-a u peer-to-peer vezi. Uključuje veb servere, aplikacione servere, CMS i slične alate koji podržavaju razvoj i funkcionisanje aplikacija. Takođe se definiše i kao softverski sloj koji je između operativnog sistema i aplikacija na svakoj strani distibuiranog sistema u mreži. [10]

3) Sloj podataka

Sloj za pristup podacima (Data Access Layer) je sloj računarskog progama koji obezbeđuje jednostavniji pristup podacima koji su smešteni u okviru nekog trajnog skladišta podataka, na primer u okviru relacione baze podataka ili file sistema. Ovaj sloj čine klase koje obezbeđuju podatke iz baze podataka, a koje mogu pristupati podacima pozivom procedura. DAL sakriva

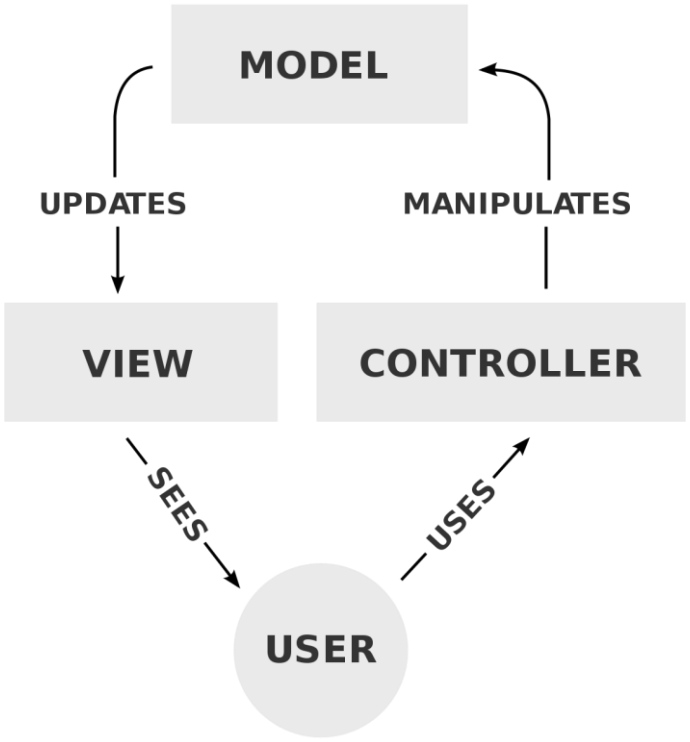
kompleksnost skladištenja podataka, a može da sadrži upite i nad više različitih izvora podataka i više baza podataka. DAL može biti zavisan od konkretnog sistema za upravljanje bazom podataka, odnosno servera baze podataka, ili nezavisan. DAL koji podržava više različitih sistema za upravljanje bazom podataka je lakši za održavanje, jer se lako izvrši izmena podrške konkretnom sistemu, dok ostali elementi ostaju nepromenjeni. Osnovna podrška sloju podataka treba da bude za CRUD (Create, Read, Update, Delete) operacije nad podacima. [10]

2.2.2. MVC (Model View Controller)

Model View Controller je softverski arhitekturni dizajn patern. Deli aplikaciju na 3 međusobno povezane komponenete kako bi se razdvojile interne prezentacije informacija od načina kako je informacija prezentovana i prihvaćena od strane korisnika. Na ovaj način, MVC dizajn patern omogućava paralelni razvoj komponenti i ponovnu iskoristivost koda. Inicijalno korišćena za desktop aplikacije, ova arhitektura je postala popularna za dizajn veb i mobilnih aplikacija. Najčešće korišćeni programski jezici Java, C#, PHP i Ruby imaju svoja popularna razvojna okruženja koja se koriste u razvoju aplikacija. [12]

Komponente su:

- Model – centralna komponenta MVC paterna. Izražava ponašanje aplikacije u smislu problemskog domena, nezavisno od korisničkog interfejsa. Upravlja podacima, programskom logikom i pravilima koja su ugrađena u aplikaciju
- View – može biti bilo koji prikaz informacija. Za iste informacije može biti više različitih prikaza
- Kontroler – preuzima ulaze i konvertuje ih u komande koje su upućene Model ili View komponenti



Slika 5. MVC komponente [12]

Podela na tri komponente definiše i njihovu međusobnu povezanost. Model čuva podatke, kontroler zadaje komande modelu za preuzimanje podataka i komande kojima se podaci prikazuju u View komponenti. View generiše novi izlaz koji je predstavljen korisniku na osnovu promena koje su se desile u modelu. Kontroler može da šalje komande modelu da promeni stanje modela i može da šalje komande view komponenti da se izmeni prikaz podataka na osnovu izmena modela ili da se prikaže druga vrsta pogleda. [11]

2.2.3. Veb servisi

Veb servis je aplikacija smeštena na nekom serveru, koja je pored osnovne namene dizajnirana da podrži interakciju između dve mašine preko mreže i omogući razmenu informacija između njih. Smatra se da je svaki servis i veb servis ako je:

- Dostupan preko interneta (ili interne mreže)
- Koristi standardizovan sistem poruka
- Prepoznatljiv je od strane mehanizma za pretragu
- Nije vezan za operativni sistem ili programski jezik

Veb servisi se objavljuju na jedinstvenoj lokaciji gde se nude kao usluge. UDDI (Universal Description, Discovery and Integration) predstavlja centralizovanu lokaciju koja obezbeđuje mehanizam za registrovanje i pronalaženje veb servisa.

Servisi mogu biti zatvoreni (privatni) ali i javno dostupni. Glavni protokol za transport podataka izmedju veb servisa i klijenta je HTTP (Hyper Text Transfer Protocol) iako je moguće koristiti i druge protokole. Format za prenos podatak je XML ili JSON (JavaScript Object Notation). [13]

SOAP veb servisi

Simple Object Access Protocol, ili skraćeno SOAP je komunikacioni protokol koji se koristi za opis poruka određenog formata, koji sam protokol definiše i propisuje, između aplikacija na različitim operativnim sistemima i različitim tehnologijama. Postoji nekoliko tipova poruka u ovom protokolu, a jedan od najpoznatijih je poziv udaljenim procedurama (tzv. Remote Procedure Call – RPC). To je tip poruka u kojem jedan čvor u mreži (klijent) šalje zahtev drugom čvoru (server) nakon čega server vraća odgovor na primljeni zahtev. SOAP poruka predstavlja XML dokument koji se sastoji iz sledećih elemenata: [13]

- Envelope element – predstavlja obavezan deo koji identifikuje XML dokument kao SOAP poruku
- Zaglavlje (header) – opcioni deo
- Telo (body) – obavezni deo koji nosi parametre zahteva i vraća odgovor veb servisa
- Fault element – opcioni deo koji pruža informacije o tome šta treba uraditi ukoliko dođe do greške prilikom pocesuiranja poruke

Nedostatak servisa baziranih na datom protokolu je kompleksnost i opširnost pri korišćenju XML-a, kao i potrošnja resursa koja je neophodna za parsiranje dokumenta. [14]

RESTful veb servisi

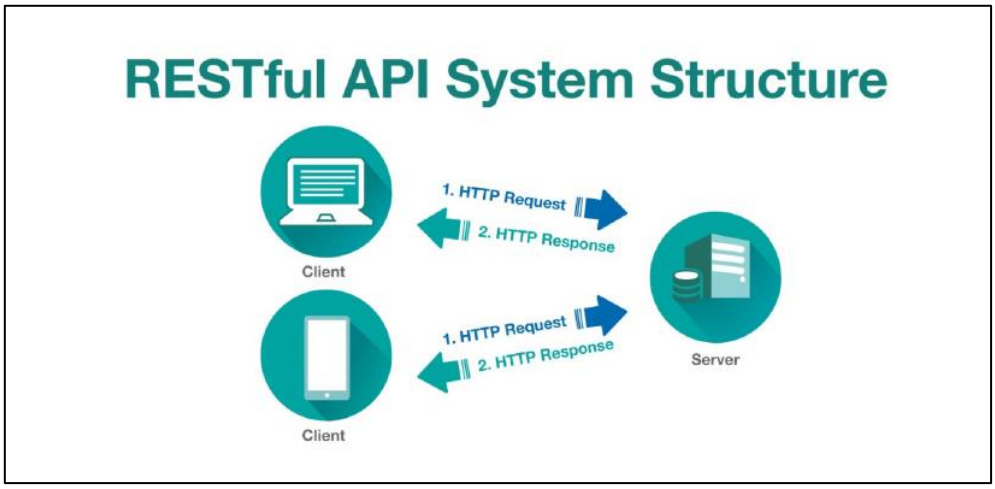
Ovi servisi su jednostavnije integrisani sa HTTP-om od SOAP servisa, ne zahtevaju XML poruke ili WSDL (Web Services Description Language) opise servisa. Danas se RESTful

izdvojio kao dominantan mrežni servis, potisnuo je SOAP i WSDL jer je značajno jednostavniji za korišćenje. [13]

Podaci se najčešće prebacuju u JSON formatu mada je dostupan i XML i YAML format. Zasniva se na **REST arhitekturi**, veoma je fleksibilan i jednostavan za razumevanje. Može biti izvršen na bilo kom klijentu ili serveru koji ima HTTP/HTTPS podršku. RESTful servisi treba da imaju sledeće osobine i karakterisitike:

- Nepostojanje stanja (Stateless)
- Mogućnost keširanja (Cacheable)
- Uniformni interfejs (Uniform interface URI)
- Izričito korišćenje HTTP metoda
- Transfer XML i/ili JSON

Kod ovog tipa servisa, resursi (npr. statičke strane, datoteke, podaci iz baze…) imaju sopstveni URL ili URI koji ih identifikuju. Pristup do resursa je definisan HTTP protokolom, gde svaki poziv čini jednu akciju (kreira, čita, menja ili briše podatke). Isti URL se koristi za sve operacije ali se menja HTTP metod koji definiše vrstu operacije. REST koristi *"CRUD like"* HTTP metode kao što su: GET, POST, PUT, DELETE, OPTIONS. [13]



Slika 6. RESTful API [15]

Odlike RESTful servisa su:

- Jednostavnost - klijenti koji pozivaju REST servise ne moraju da formatiraju zahteve po SOAP specifikaciji i ne moraju da parsiraju SOAP odgovor kako bi iz njega izvukli rezultat.
- Fleksibilnost formata vraćenih podataka - format u kome se podaci vraćaju nije unapred definisan i zavisi od samog servisa. Klijenti mogu da zatraže podatke u formatu koji im najviše odgovara, za razliku od SOAP formata koji iako je standardizovan mora da se parsira. Pa tako JavaScript može dobiti podatke u JSON formatu koji lako može da pročita, a RSS čitač u RSS-XML formatu koji može da prikaže.
- korišćenje postojeće mrežne infrastrukture
- brzo savladavanje tehnike

3. Opis primenjene tehnologije

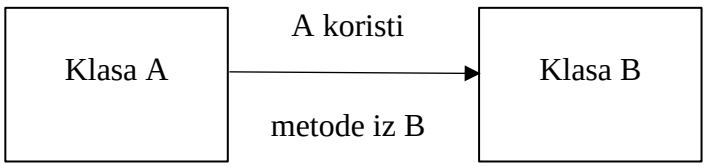
Aplikacija predškolske ustanove je spoj raznih novijih tehnologija poput Spring Boot-a za Java platformu, Angulara za front-end deo aplikacije i Hibernate-a za rad sa relacionom bazom podataka. U ovoj sekciji će svaka od korišćenih tehnologija biti ukratko opisana.

3.1.1. Spring Boot Framework

Spring Boot predstavlja razvojno okruženje za Java platformu koje pruža brži način da se podignu, konfigurišu i pokrenu veb aplikacije. To je zato što Spring Boot omogućava programeru da se fokusira na sam razvoj aplikacije, umesto na konfigurisanje i postavljanje samog projekta od početka. Pošto Spring Boot upravlja infrastrukturom projekta, on ga automatski konfiguriše i aplikacija se može pokrenuti uz pomoć samo jednog klika, što većinu programera privlači kod ovog razvojnog okruženja. [16] Njegove najbitnije karakteristike uključuju:

- **Auto-configuration**: Konfiguriše aplikaciju u zavisnosti od svog okruženja, kao i od informacija koje mu programer pruža.
- **Standalone**: Nije potreban poseban server ili posebno okruženje kako bi se aplikacija pokrenula. Jedino je potrebno pritisnuti dugme „Run“ i aplikacija će se pokrenuti.
- **Opinionated**: Ova karakteristika omogućava razvojnom okruženju da „razmišlja.“ Ukratko, to znači da ukoliko programer često koristi određene biblioteke, Spring Boot će ih na početku novog projekta automatski konfigurisati.

Ono po čemu je Spring framework najviše popularan je **Dependency Injection**. Ovaj termin se koristi za opisivanje tehnike u programiranju koja čini klasu nezavisnom tako što izoluje kreiranje objekta od njegovog korišćenja, što znači da se instanciranje klase vrši na nekom drugom mestu. [17]



Slika 7. Dijagram zavisnosti klasa [17]

Pre nego što klasa A može da koristi metode klase B, potrebno je da se kreira objekat klase B, odnosno u klasi A je potrebno instancirati klasu B. Međutim, prema S.O.L.I.D. principima objektno-orijentisanog programiranja, klase treba da se fokusiraju samo na jednu stvar, a to je odrađivanje nekog dela posla za koji su one zadužene, a ne da kreiraju objekte kako bi to ostvarile. Zbog toga je uvedena posebna klasa koja je zaslužna za instanciranje klasa, i prosleđivanja objekata (stoga „Injection“) drugim klasama.

Na ovaj način se poboljšava ponovna upotrebljivost koda i smanjuje se potreba za promenama u klasi (objekti koji klasa koristi se mogu menjati, i nema potrebe za izmenom metoda), što je inače i cilj S.O.L.I.D. principa. [18]

Postoje 3 različita načina [19]:

- **Constructor injection** – Objekti su prosleđeni preko konstruktora
- **Setter injection** – Objekti su prosleđeni preko set metode
- **Interface injection** – Mora se definisati interfejs koji koristi metodu za primanje objekata

3.1.2. Angular 4/5

Angular je platforma i razvojno okruženje za izgradnju veb aplikacija u HTML-u i TypeScript-u. Napisan je u TypeScript jeziku, i implementuje svoje funkcionalnosti kao set biblioteka koje se importuju unutar aplikacija. Osnovni gradivni elementi Angular aplikacije su NgModules, koji pružaju kontekst za komponente. Komponente definišu poglede koji predstavljaju elemente na ekrenu i one koriste servise sa specifičnim funkionalnostima koje nisu direktno povezane sa pogledima. Svaka angular aplikacija ima minimum jednu komponentu koja se naziva „root.“ [20]

Što se Angulara tiče, programeri kažu da nije najlakše razvojno okruženje za korišćenje, ali je zato moćan alat, dobro dokumentovan i interoperabilan.

Prednosti Angular razvojnog okruženja [21]:

- Alati – Pored mnoštvo korisnih alata, angular pruža i dizajn paterne (šablone) kako bi omogućio programerima da kreiraju projekte na organizovan i lako održiv način. Kada se kreira Angular aplikacija, kod je struktuiran na organizovan način što čini njegovu izmenu i snalaženje veoma lakim
- TypeScript – Angular je kreiran uz pomoć TypeScripta koji se dosta oslanja na JavaScript ES6 (ECMAScript 6), što znači da nema potrebe za učenjem novog jezika, a opet se dobijaju nove funkcionalnosti
- Biblioteke – Sa Angularom, postoje već biblioteke i alati koji omogućavaju brže i jednostavnije kreiranje aplikacije. Na primer, forme se vrlo lako mogu napraviti uz pomoć FormControl klasa sa već ugrađenim validacijama. Takođe se mogu slati svi HTTP definisani zahtevi, i postavljanje ruta je lakše nego ikad
- Komponente – Angular je napravljen sa ciljem da odvoji logiku aplikacije od samih DOM (Document Object Model) elemenata, što omogućava jednostavniju izmenu različitih komponenti
- Dokumentacija – Na sajtu https://angular.io se nalaze sve informacije vezane za angular komponente, funkcionalnost i drugi materijali koji odgovaraju na sva pitanja vezana za upotrebu ovog moćnog okruženja

NodeJS kao i NPM su bili neophodan download pre samog početka razvoja Angular aplikacije, jer se pomoću Node Packet Manager-a bio instalirao sam Angular projekat kao i neke od zavisnih biblioteka korišćenih u okviru istog. [22]

3.1.3. HTML

HTML je skraćenica za Hyper Text Markup Language i koristi se za kreiranje veb stranica i opisivanje njene strukture. Internet pretraživači preuzimaju HTML kod i kada ga pročitaju mogu da prikažu veb stranicu onako kako je ona opisana kodom. HTML se drugačije naziva i „skelet“ veb stranice jer predstavlja samo raspored elemenata na njoj. HTML elementi mogu biti slika, tekst, forma, link, paragraf i razni drugi objekti koji se definišu otvorenim i zatvorenim uglastim zagradama.

```
<img src = “slika.jpg“>
<p> Paragraph item </p>
<div> Container </div>
```

Listing 1. Primer osnovnih HTML tagova

Neki HTML elementi mogu da imaju svoje podelemente koji takođe moraju biti definisani uglastim zagradama.

```
<div>
  <div>
    <h2> Naslov </h2>
  </div>
</div>
```

Listing 2. Primer HTML tagova sa podelementima

HTML dokument se sastoji iz **head** i **body** sekcije. Navedeni tagovi nalaze se unutar html sekcije (otvorenog i zatvorenog html taga) kojim html dokument uvek i počinje i govori veb čitaču da je resurs koji prikazuje html dokument. Tag **<head>** predstavlja zaglavlje u kom se nalaze informacije o dokumentu (naslov, opis, ključne reči, ime autora), definicije JavaScript poziva, CSS stilova itd. Browser ne prikazuje informacije koje se nalaze između tagova **<head>** i **</head>**, osim sadržaja taga **<title>**. Unutar head sekcije se takođe smeštaju meta tagovi koji nose dodatne informacije o veb stranici koji se ne prikazuju i definišu se kao uređeni parovi naziva parametra i vrednosti. Meta tagovi imaju veliku primenu sa aspekta SEO pretrage (Search Engine Optimization). Vidljivi deo HTML dokumenta, odnosno kompletan sadržaj koji browser prikazuje korisniku smešta se u telo HTML dokumenta (**<body>**). [23]

Elementima se mogu dodeljivati identifikacione oznake, ili klase, koje ih jedinstveno obeležavaju i koje CSS koristi kako bi promenili izgled tog elementa. Razlika između HTML5, HTML4 i ostalih ranijih verzija ovog jezika jeste što su uvedeni novi elementi poput **embed**, **footer**, **header**, **video**, **section**.

3.1.4. CSS

CSS predstavlja akronim za Cascading Style Sheets i koristi se za izmenu izgleda osnovnih HTML elemenata. Bez CSS-a, svi HTML elementi su se na internet stranicama prikazivali na isti način, a ukoliko bi se hteo promeniti taj osnovni izgled, elementima su se atributi dodavali direktno iz HTML-a. Međutim, CSS dozvoljava autorima da se atributi pišu u posebnom dokumentu, kao što su boje teksta, pozadine, veličina i oblik, što je rezultovalo mnogo jednostavnijem HTML-u.

Pomoću dodeljenih klasa elementima, uz pomoć CSS-a im se može u jednom koraku dodeliti isti atributi. Na ovaj način sajtovi koji se sastoje iz više stranica, mogu lako zadržati kontinuirani izgled, a da pritom nije bilo potrebe dodavati isti atribut svakom elementu posebno.

CSS sintaksa se sastoji iz dva glavna dela:

- Selektor
- Deklaracija (jedna ili više)

I piše se jednolinijski ili višelinijski. Deklaracija se deli na svojstvo (atribut, osobina, property) i vrednost i grupišu se unutar vitičastih zagrada {}, dok se svaka deklaracija završava karakterom tačka-zarez (;). [23]

```
header .navigation-bar {
  height: 80px;
  width: 80%;
  margin: 0 auto;
  justify-content: space-between;
}
```

Listing 3. Primer CSS koda

U zavisnosti od vrste selektora CSS stilovi se mogu pisati na više načina. Može biti selektor po osnovi tipa elementa koji selektuje sve elemente određenog elementa, na primer, div, p, h2 itd. Može biti selektor po osnovi identifikatora HTML elementa i selektor klase koji selektuje HTML elemente sa navedenom klasom. Postoje i takozvani pseudo-selektori koji se označavaju dvotačkom (:) i uglavnom se vezuju za neki HTML element. Primeri pseudo-selektora su **:hover**, **:active**, **:focus**, :**visited** itd. [23]

CSS stilovi se mogu pisati na 3 načina (lokacije):

- Inline (umetnuti) stil – definiše se za jedan konkretan HTML element putem style atributa. Validan je samo za element kojem je pridružen i eksplicitno definiše stil tog elementa.
- Interni stil – Navedeni stilovi su dostupni samo unutar HTML dokumenta u kojem se nalaze
- Eksterni CSS fajl – Stilovi se nalaze u eksternom fajlu i mogu da se koriste u više HTML stranica. Importuju se upotrebom <link> taga unutar <head> sekcije HTML dokumenta.

3.1.5. Bootstrap

Bootstrap je besplatni open source alat za lakše i brže dizajniranje veb stranica i aplikacija. U alat su ugrađeni šabloni za dizajn raznih elemenata poput dugmića, tabela, navigacije, boja, slika i drugih, zasnovani na HTML-u i CSS-u. Prva verzija ovog alata je objavljena 2011. godine a od tada je izašlo još 4 verzije u kojima su se uvodile razne novine poput responzivnog dizajna, podrške novih internet pretraživača itd. [24] Sa Bootstrap responzivnim dizajnom, veb stranice su mogle lako da se prilagode karakteristikama svih vrsta uređaja na kojima su se prikazivale (laptop, mobilni uređaji, tableti).

3.1.6. IntelliJ razvojno okruženje

Za backend deo ovog projekta je korišćena IntelliJ IDEA, koja je namenjena za Java programski jezik. Kompanija koja je razvila ovo okruženje se naziva JetBrains. IntelliJ je dostupna u dve verzije: The Community Edition i Ultimate Edition. Ono što IntelliJ izdvaja od drugih programskih okruženja je lakoća korišćenja, fleksibilnost i dizajn. [25]

Za frontend deo ovog projekta je korišćen Visual Studio Code. Pruža podršku za stotine jezika, a razlog zašto većina programera bira ovaj IDE za razvoj softvera je to što omogućava veću produktivnost pomoću intuitivnih prečica na tastaturi, lakoj kastomizaciji, navigaciji koda, stintaksnog bojenja koda, IntelliSense automatskog dovršavanja unete reči itd.

4. Analiza semantičke oblasti

Rad predškolske ustanove je veoma odgovoran posao koji zahteva punu pažnju i angažovanje svojih zaposlenih u cilju obezbeđivanja adekvatne sredine u kojoj deca mlađa od 7 godina mogu na zdrav način da uče i da razvijaju svoje psihofizičke, motoričke i socijalne sposobnosti. Naravno, pre polaska u predškolsku ustanovu, vrši se prijava dece za određeni program nastave i boravka, gde zatim ustanova kreira spisak primljenih učenika izabranim po Pravilniku o bližim uslovima za određivanje prioriteta pri upisu. U nastavku će pomenute sekcije biti detaljnije opisane.

3.1. Rad predškolske ustanove

Osnovni cilj predškolskih ustanova je da se pruži društvena pomoć u zbrinjavanju dece za vreme radnog angažovanja njihovih roditelja. Postoje dve osnovne vrste funkcija koje predškolska ustanova obavlja, a to su socijalna i vaspitno-obrazovna. Društvena pomoć u zbrinjavanju čini socijalnu funkciju ustanove. S druge strane, zadovoljavanje razvojnih potreba dece u koje spadaju potrebe za socijalizacijom, vaspitanjem i obrazovanjem, koje se ostvaruju programiranim sadržajima vaspitno-obrazovnog karaktera, jesu vaspitno-obrazovna funkcija. Pored ove dve funkcije predškolske ustanove moraju da vrše i preventivnu zdravstvenu funkciju jer boravak male dece u predškolskoj ustanovi nosi i određene faktore rizika koji mogu štetno delovati na njihov psihofizički i socijalni razvoj. Sve tri navedene funkcije predškolskih ustanova čine suštinu njihove delatnosti, ostvaruju se istovremeno i treba ih posmatrati u jedinstvu. [26]

U nastavku će biti opisani oblici rada s decom.

Celodnevni boravak dece (od 1 do 6,5 godina)

„Celodnevni boravak je oblik vaspitno-obrazovnog rada namenjen deci od prve godine do polaska u školu. Program celodnevnog boravka realizuju medicinske sestre - vaspitači (rad sa decom do 3 godine) i vaspitači (rad sa decom od 3 godine do polaska u školu). Programska orijentacija ovog oblika obezbeđuje pravilan psiho-fizički razvoj, socijalnu sigurnost, vaspitanje, obrazovanje i preventivnu zdravstvenu zaštitu. Vaspitni rad i nega dece uzrasta do 3 godine ostvaruje se u okviru programiranih aktivnosti (motornih, senzorno-perceptivnih, ritmičko-muzičkih, intelektualnih i jezičkih), socijalno emocionalnih odnosa i nege dece“. [26]

Osnovni cilj vaspitnog rada sa decom ovog uzrasta je stvaranje povoljne vaspitne sredine u kojoj će dete sticati iskustva po svom sopstvenom programu. Organizovanim sistemom sadržaja i metoda dete će iskustva prevoditi u saznanja, otkrivanjem novih stvari kako o svojoj okolini, tako i o sebi. Osnovni zadatak vaspitnog rada je da čuva, podstiče i oplemenjuje spontane izraze ponašanja deteta u odnosu na okolinu. U radu se mora poštovati individualnost deteta u otkrivanju sebe i sveta koji ga okružuje. [26]

Posebni zadaci realizuju se kroz tri osnovne oblasti razvoja:

- fizičko-senzorni razvoj (očuvanje fizičkog zdravlja dece, podsticanje razvoja pokreta, ovladavanje motorikom, podsticanje celovitog motornog i senzornog razvoja, razvijanje navika)
- emocionalno-socijalni razvoj (negovanje otvorenosti deteta za doživljaje, pružanje pomoći u sticanju samostalnosti, podsticanje poverenja u sopstvene sposobnosti,

pomaganje detetu u usvajanju osnovnih moralnih vrednosti, kao i podsticanje zadovoljstva i radosti kod deteta)

- saznajni razvoj (podsticanje i negovanje prirodne radoznalosti deteta u odnosu na svet koji ga okružuje, negovanje osetljivosti za utiske kao motive za postavljanje pitanja, podsticanje i bogaćenje dečjeg govora kao sredstva komunikacije i sticanja znanja, podsticanje razvoja senzomotornih i perceptivnih sposobnosti, stvaranje povoljnih uslova za formiranje početnih saznajnih pojmova kroz praktične aktivnosti)

Sestre-vaspitači u planiranju svog rada u grupi imaju u vidu karakteristike razvoja dece uzrasta do 18 meseci, od18 do 24 meseca i od 24 do 36 meseci. Grupe se formiraju u skladu sa zakonskom regulativom, potrebama roditelja i dece. [26]

Slika 8. Deca u predškolskoj ustanovi [27]

Pripremni predškolski program za decu uzrasta od 5,5 do 6,5 godina

„Vaspitno-obrazovni rad u godini pred polazak u školu je deo obaveznog devetogodišnjeg obrazovanja i vaspitanja. Pripremni predškolski program zasniva se na Pravilniku o opštim osnovama predškolskog programa. Pripremni predškolski program predstavlja sponu predškolskog i školskog obrazovanja i vaspitanja.“ [26]

Pripremni predškolski program pored navedene ima i sledeće funkcije:

- obezbeđivanje kvalitetne vaspitno-obrazovne sredine, koja uvažava prava deteta, poštuje njegove osobenosti i potrebe i podstiče njegov ukupan psihofizički razvoj
- obezbeđivanje uslova za proširivanje i sređivanje socijalnog i saznajnog iskustva čime se ublažavaju socijalno-kulturne razlike i osigurava donekle podjednak start za polazak u školu
- unapređivanje vaspitne funkcije porodice (dopuna je porodičnom vaspitanju)

„Program pripreme dece za školu, kao sistem aktivnosti i sadržaja, posebnih metodičkih postupaka kojima se postiže intelektualna, socijalna, emocionalna i motivaciona gotovost za ono što ih očekuje u školi, uz podsticanje radoznalosti i interesovanja za školski način učenja, ostvaruje se kroz Model A i Model B pripremnog predškolskog programa (u zavisnosti od opredeljenja vaspitača) razvijenim u Predškolskom programu Ustanove“. [26]

Model A	Model B
Vaspitač je okosnica vaspitnog programa, kao kreator i istraživač sopstvene prakse. Zadatak vaspitača je da na osnovu interesovanja dece dnevno planira veći broj sadržaja i aktivnosti i pruži deci mogućnost izbora. Rad sa decom organizuje se prema dečijim potrebama i interesovanjima.	Strukturu programske osnove vaspitno-obrazovnog rada čine: fizički razvoj (telesne, perceptivne i zdravstveno-higijenske aktivnosti), socio-emocionalni i duhovni razvoj (društvene, afektivne i ekološke aktivnosti), kognitivni razvoj (otkrivačke, logičke, radne i saobraćajne aktivnosti) i razvoj komunikacije i stvaralaštava (govorne, dramske, likovne, muzičke i plesne aktivnosti).

Tabela 1. Model A i Model B organizovanja vaspitnog programa [26]

Program preventivno-zdravstvene zaštite

Predškolska ustanova u skladu sa programom preventivno-zdravstvene zaštite planira mere i aktivnosti koje se u saradnji sa Zdravstvenim centrom i Zavodom za javno zdravlje sprovode u cilju očuvanja zdravlja dece u kolektivu. [26]

Neke od aktivnosti obuhvataju:

- Stvaranje optimalnih higijenskih uslova u sredini gde deca borave (održavanje opšte čistoće radnih soba, sanitarnih čvorova, garderoba, inventara i igračaka, optimalna temperatura, vlažnost vazduha, provetrenost i osvetljenost prostorija, dezinfekcija, dezinsekcija I deratarizacija prostora)
- Posebna pažnja se posvećuje održavanju distributivne kuhinje (higijenska distribucija hrane, svakodnevno pranje i dezinfekcija vozila za prevoz hrane, higijensko održavanje posuđa i inventara, kuhinjsko osoblje sanitarno pregledano)
- Redovno mesečno uzimanje briseva, kako sa ruku zaposlenih tako i sa radnih površina, podova, igračaka i ostalog
- Sprovođenje pravilnog ritma dnevnih aktivnosti i ishrane u toku radnog vremena, poštujući fiziološke potrebe dece

4.1. Zakon o predškolskom vaspitanju

Ciljevi predškolskog vaspitanja i obrazovanja su podrška:

1) celovitom razvoju i dobrobiti deteta predškolskog uzrasta, pružanjem uslova i podsticaja da razvija svoje kapacitete, proširuje iskustva i izgrađuje saznanja o sebi, drugim ljudima i svetu; [28]

2) vaspitnoj funkciji porodice;

3) daljem vaspitanju i obrazovanju i uključivanju u društvenu zajednicu;

4) razvijanju potencijala deteta kao pretpostavke za dalji razvoj društva i njegov napredak.

Principi predškolskog vaspitanja i obrazovanja su:

1) dostupnost: jednako pravo i dostupnost svih oblika predškolskog vaspitanja i obrazovanja, bez diskriminacije i izdvajanja po osnovu pola, socijalne, kulturne, etničke, religijske ili druge pripadnosti, mestu boravka, odnosno prebivališta, materijalnog ili zdravstvenog stanja, teškoća i smetnji u razvoju i invaliditeta, kao i po drugim osnovama, u skladu sa zakonom;

2) demokratičnost: uvažavanje potreba i prava dece i porodice, uključujući pravo na uvažavanje mišljenja, aktivno učešće, odlučivanje i preuzimanje odgovornosti;

3) otvorenost: građenje odnosa sa porodicom, drugim delovima u sistemu obrazovanja (škola), zajednicom (institucijama kulture, zdravstva, socijalne zaštite), lokalnom samoupravom i širom društvenom zajednicom;

4) autentičnost: celovit pristup detetu, uvažavanje razvojnih specifičnosti predškolskog uzrasta, različitosti i posebnosti, negovanje igre kao autentičnog načina izražavanja i učenja predškolskog deteta, oslanjanje na kulturne specifičnosti; 5) razvojnost: razvijanje različitih oblika i programa u okviru predškolske delatnosti u skladu sa potrebama dece i porodice i mogućnostima lokalne zajednice, kontinuirano unapređivanje kroz vrednovanje i samovrednovanje, otvorenost za pedagoške inovacije. [28]

4.2. Upis dece u predškolsku ustanovu

Upis dece u predškolsku ustanovu vrši se u skladu sa Zakonom. Prilikom upisa dece u predškolsku ustanovu, čiji je osnivač Republika Srbija, autonomna pokrajina ili jedinica lokalne samouprave, prioritet za upis imaju deca iz osetljivih grupa. Način i postupak upisa dece u predškolsku ustanovu bliže se uređuju statutom, uz saglasnost osnivača.

4.2.1. Pravilnik o bližim uslovima za utvrđivanje prioriteta za upis

Predškolska ustanova u skladu sa svojim mogućnostima i iskazanim potrebama porodica za različitim programima predškolskog vaspitanja i obrazovanja, na zahtev roditelja, odnosno staratelja, vrši upis dece predškolskog uzrasta, prema sledećim kriterijumima za utvrđivanje prioriteta za upis:

1) Deca iz društveno osetljivih grupa
   a) Deca žrtve nasilja u porodici
   b) Deca iz porodica koja koriste neki oblik socijalne zaštite i deca bez roditeljskog staranja
   c) Deca samohranih roditelja
   d) Deca iz socijalno nestimulativnih sredina
   e) Deca sa smetanjama u psihofizičkom razvoju
   f) Deca iz porodice u kojoj je dete teško obolelo ili ima smetnje u razvoju
   g) Deca teško obolelih roditelja
   h) Deca čiji su roditelji ratni vojni invalidi ili imaju status raseljenog ili prognanog lica
   i) Deca predložena od strane centra za socijalni rad
   j) Deca iz sredina u kojima je usled porodičnih i drugih životnih okolnosti ugroženo zdravlje, bezbednost i razvoj

2) Deca zaposlenih roditelja i redovnih studenata
3) Deca koja imaju status trećeg i svakog narednog deteta u primarnoj porodici

4) Deca čija su braća ili sestre upisani u istu predškolsku ustanovu
5) Ostala deca

Deca se upisuju u Ustanovu po redosledu koji se određuje na osnovu broja bodova na sledeći način:

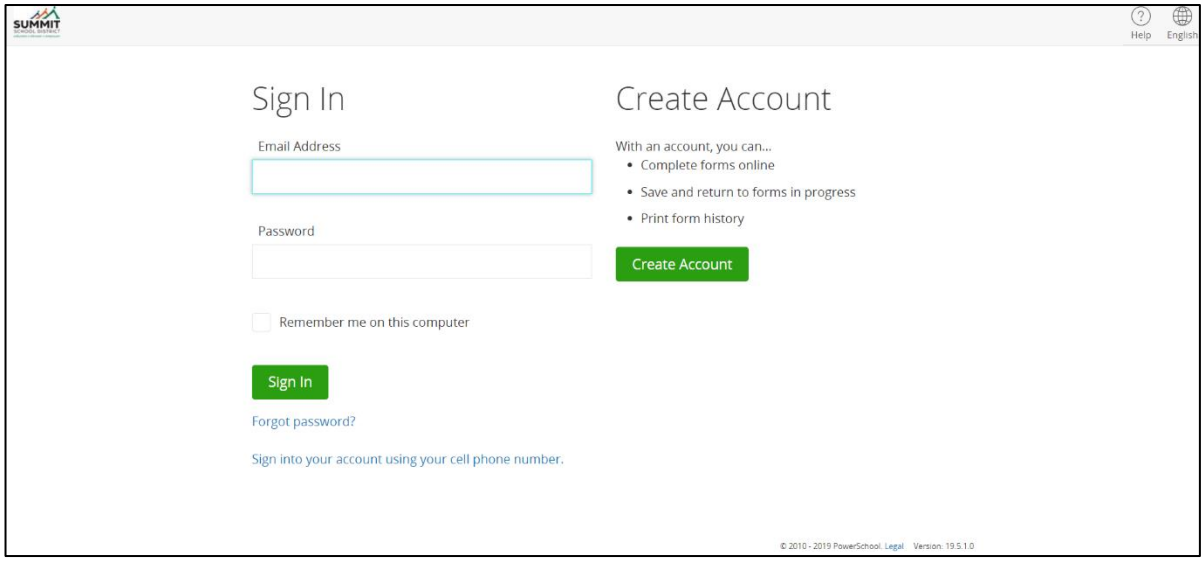
- Deca čija su oba roditelja, staratelja ili usvojitelja zaposlena i deca redovnih studenata – 20 bodova
- Deca koja imaju status trećeg i svakog narednog deteta u primarnoj porodici – 10 bodova
- Deca čija su braća i sestre upisani u istu Predškolsku ustanovu (prilikom upisa ako se radi o blizancima oba deteta dobijaju po 5 bodova) – 5 bodova

U slučaju kada dvoje ili više dece imaju isti broj bodova, prioritet pri prijemu utvrđuje se primenom sledećih kriterijuma, po sledećem redosledu:

1. Deca zaposlenih u Predškolskoj Ustanovi
2. Prema dužini čekanja za prijem u Ustanovu, ukoliko budu formirane liste čekanja
3. Prema većem broju dece u porodici
4. Ostala deca

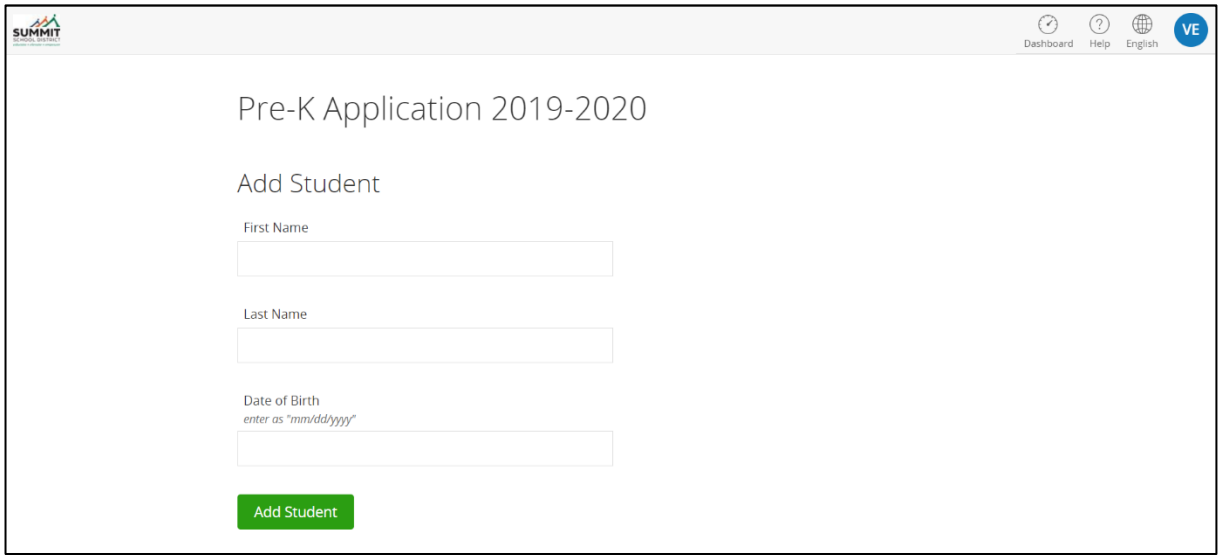
5. Postojeća rešenja

Strana škola „Summit School District“ ima svoju veb aplikaciju za slanje prijava pred upis u novu školsku godinu. Najpre, roditelji moraju da naprave nalog na sajtu koristeći svoju email adresu. (slika 9)



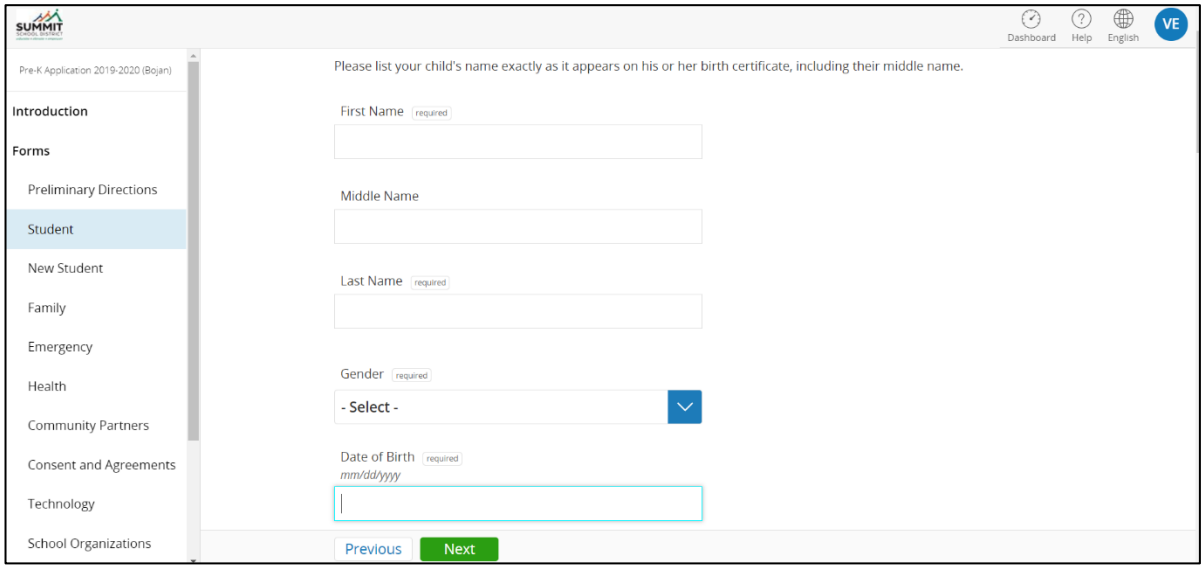
Slika 9. Početna stranica online aplikacije „Summit School District“[29]

Nakon što se nalog napravi i verifikuje preko email verifikacionog linka, omogućena je opcija popunjavanja obrasca za prijavu deteta. Prvo se upisuju osnovne lične informacije poput imena, prezimena i datuma rođenja. (slika 10)

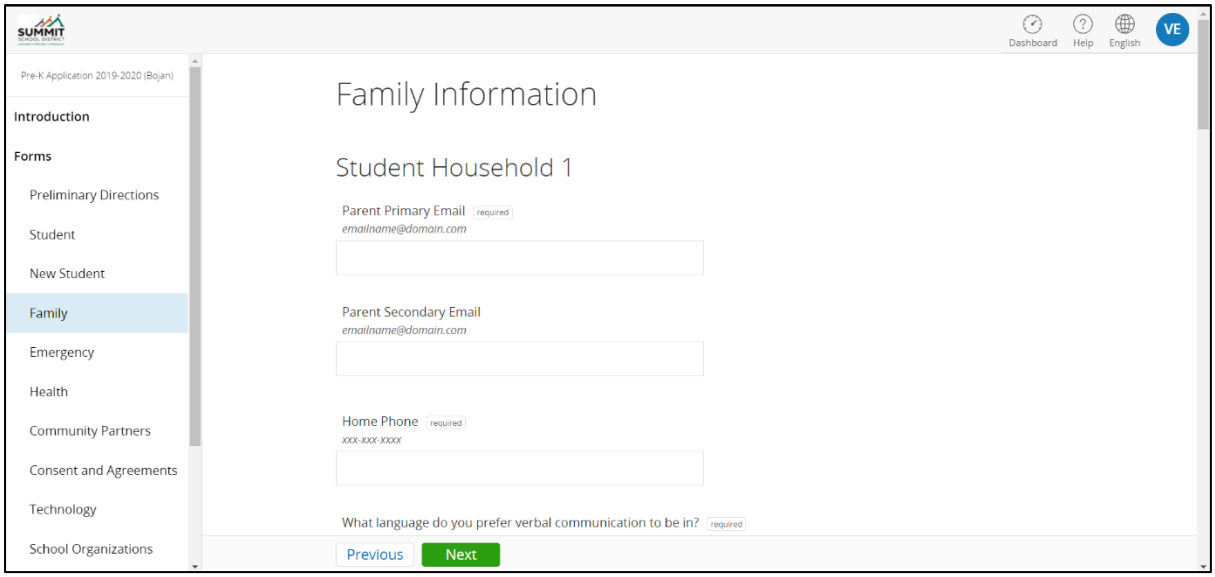


Slika 10. Forma za popunjavanje osnovnih informacija o detetu [29]

Zatim se upisuju i podaci o polu i srednjem imenu. Pored obrasca za popunjavanje, s leve strane veb stranice mogu se videti sekcije po kojima su obrasci podeljeni. (slika 11 i 12) Redosled pripremljenih obrazaca se kreću od osnovnih ličnih informacija, pa se zatim traže informacije o roditeljima, njihovim email adresama i telefonima. Jedna od najvažnijih sekcija na listi jeste sekcija vezana za zdravlje. Tu se definišu moguće alergije, vrste dijeta koje dete treba da se pridržava ili terapije neophodne za suzdržavanje ili uklanjanje simptoma bolesti od koje dete boluje.



Slika 11. Proširene osnovne informacije o detetu [29]

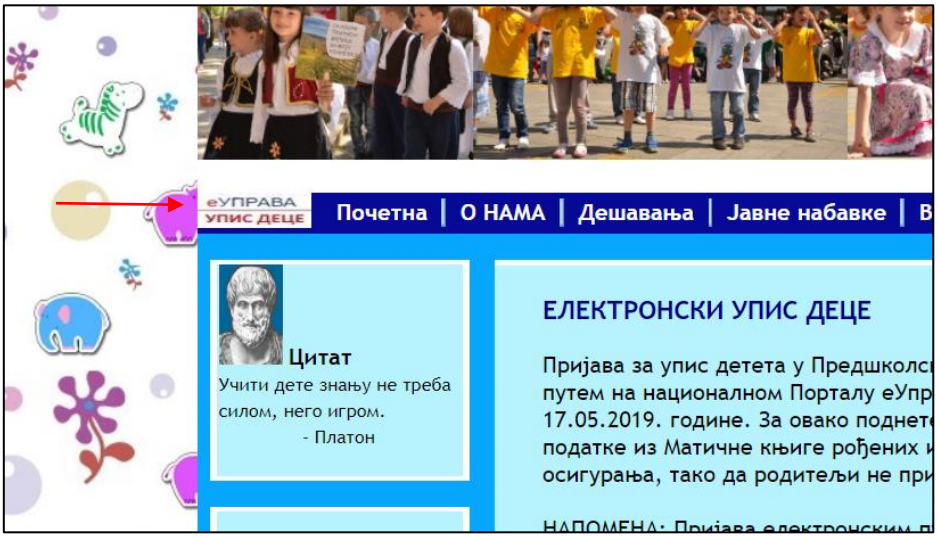


Slika 12. Forma za unos informacija o roditeljima [29]

U Srbiji je, međutim, nedavno uvedena aplikacija za podnošenje zahteva za upis dece u Predškolsku upravu koju je Ministarstvo zahtevalo. Prijava za upis deteta u Predškolsku ustanovu može se podneti elektronskim putem na nacionalnom Portalu eUprava Republike Srbije od 03.05.2019. godine. Za ovako podnete prijave sistem automatski pribavlja podatke iz

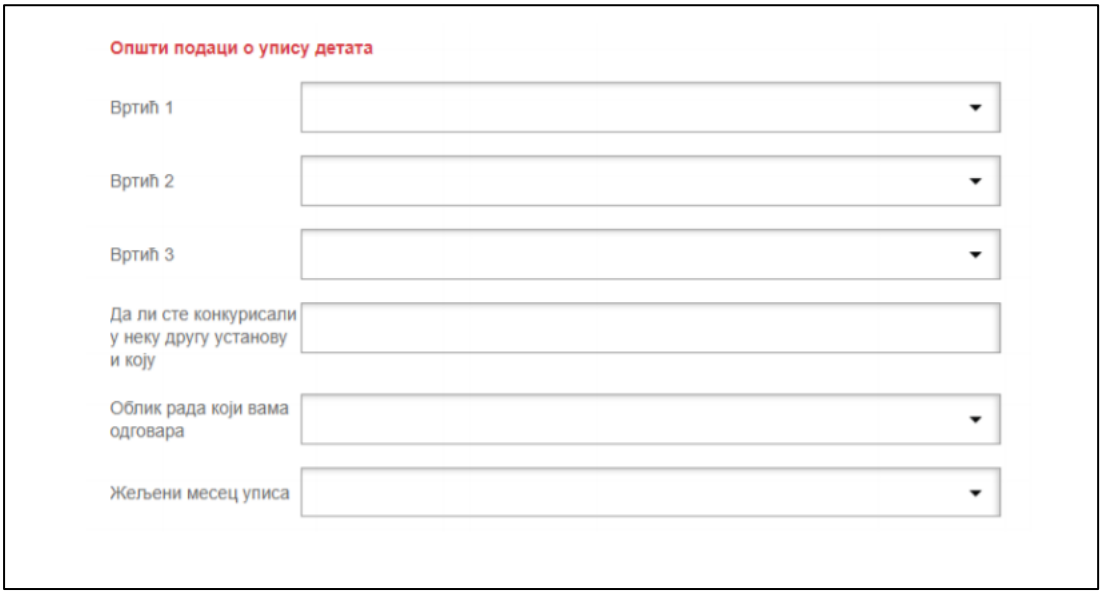
Matične knjige rođenih i Centralnog registra obaveznog socijalnog osiguranja, tako da roditelji ne pribavljaju ova dokumenta lično.

Link do ovog portala se nalazi i na stranici zrenjaninske Predškolske ustanove. [30] (slika 13)



Slika 13. Link do portala sa stranice Predškolske ustanove [30]

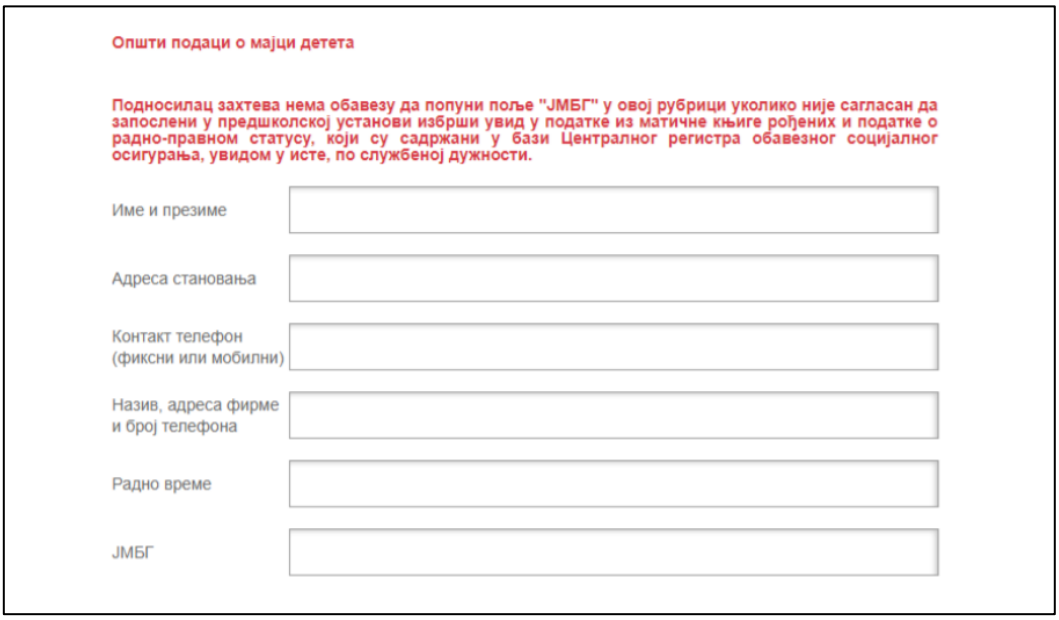
Stranica takođe sadrži uputstvo za roditelje, kako popuniti obrasce i uspešno prijaviti dete za upis. Da bi se moglo pristupiti obrascima za popunjavanje, mora se napraviti nalog na eUpravi Republike Srbije (slično kao i kod prethodnog rešenja strane škole) i ostaviti svoj jedinstveni matični broj. Nakon napravljenog naloga, najpre se izabere grad u kome se dete prijavljuje, a zatim popuni obrazac za biranje vrtića. (slika 14)



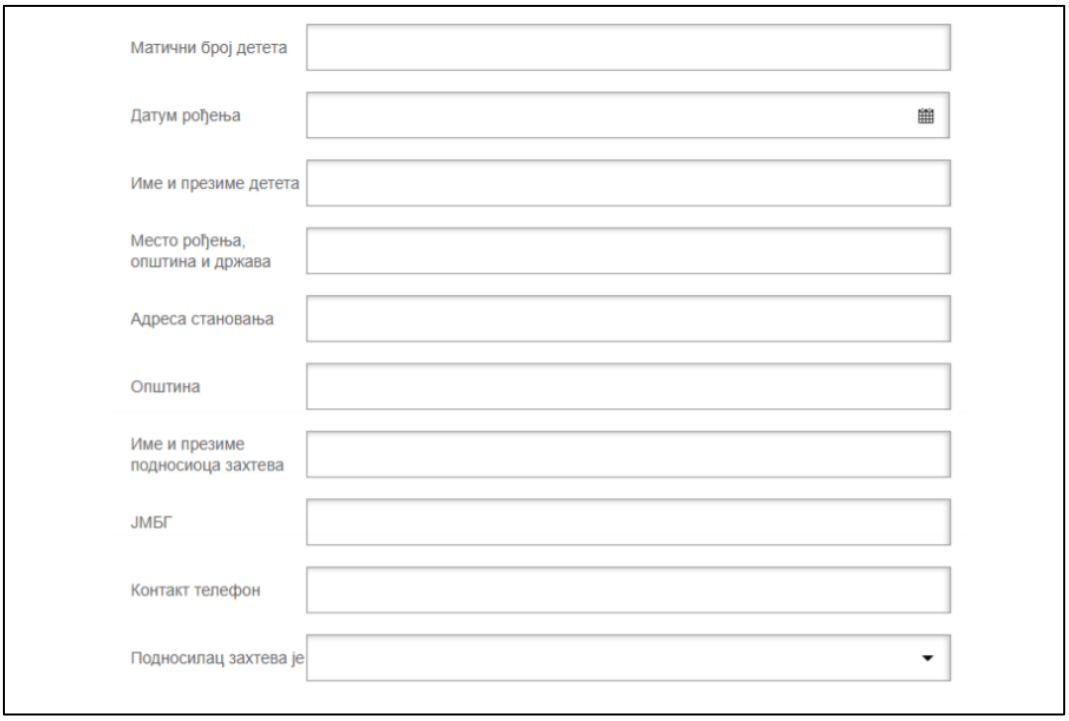
Slika 14. Opšti podaci o upisu deteta [31]

Nakon toga potrebno je označiti radni status roditelja. Ovi podaci se nakon popunjenog obrasca automatski proveravaju u Centralnom registru obaveznog socijalnog osiguranja.

Zatim se popunjavaju obrasci o ličnim podacima deteta, kao i lični podaci oca i majke odvojeno. (slika 15) i (slika 16)

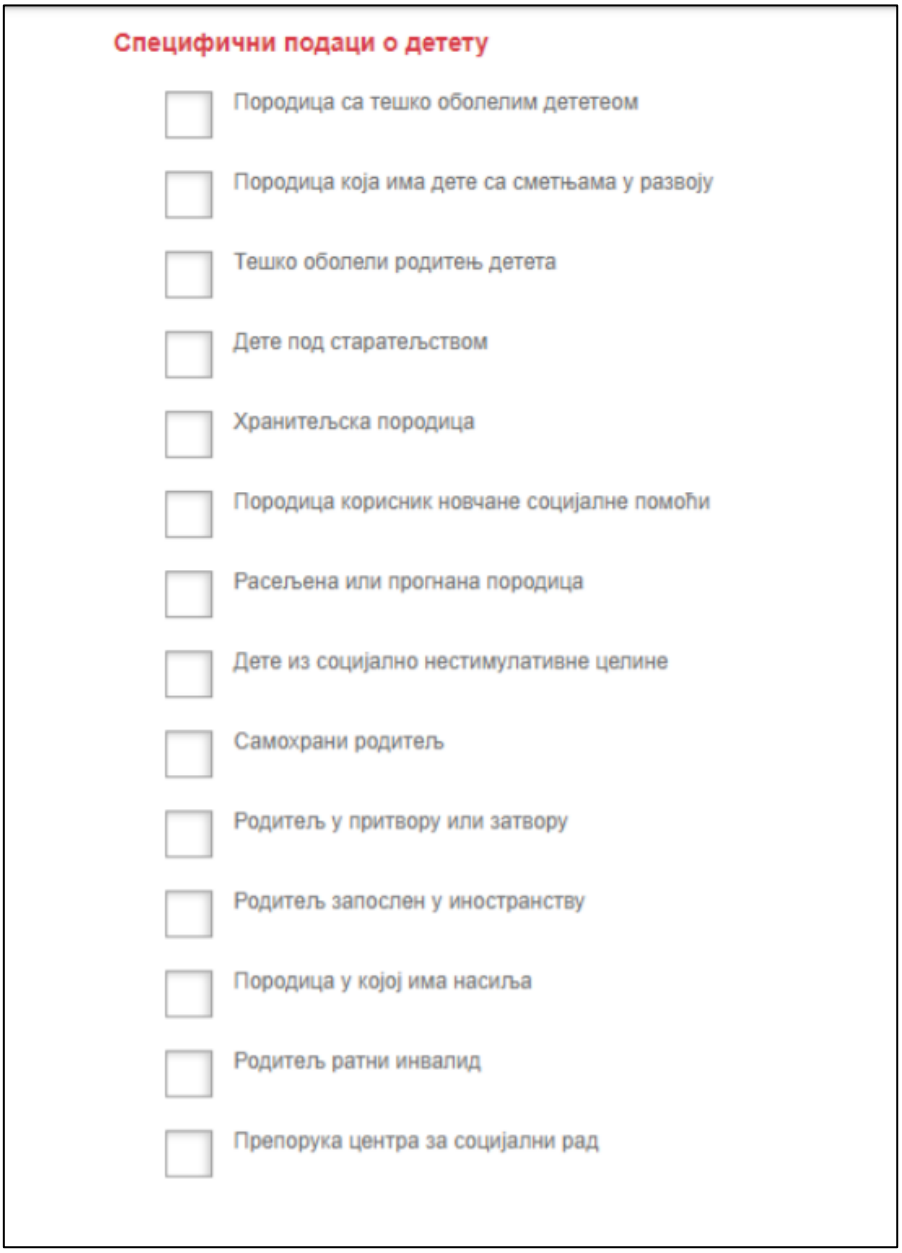


Slika 15. Lični podaci majke [31]



Slika 16. Lični podaci deteta [31]

Pred kraj se unose zdravstveni podaci o detetu, ukoliko ima zdravstvenih problema, specifičan način hranjenja, dijeta, alergija ili bilo kakvih smetnja u razvoju, a na samom kraju se nalazi checkbox lista za odabir opcije načina života i porodice u kojoj se dete nalazi. (slika 17) To su deca iz osetljive društvene grupe, i one imaju najveći prioritet pri upisu u Predškolsku ustanovu.



Slika 17. Specifični podaci o detetu [31]

6. Realizovan primer

Realizovano rešenje predstavlja prototip veb aplikacije za slanje prijava za upis u predškolsku ustanovu. Primer na kojem je zasnovana specifikacija zahteva i snimak stanja jeste Predškolska ustanova u Zrenjaninu. Cilj aplikacije je da omogući roditeljima da od kuće prijave svoje dete za vrtić, ali takođe da olakša zaposlenima ustanove sortiranje, bodovanje i prikaz prijavljenih učenika.

6.1. Prijava projekta

U tabeli 2 je prikazan tok poslovnih procesa koje je bilo potrebno automatizovati i prevesti u softverske funkcije, kao i opis procesa izrade rešenja kako tehnoloških apsekata tako i sa aspekta upravljanja projektom.

TEMA:	**Elektronska prijava za upis dece u Predškolsku ustanovu**
TIP:	**pojedinacni** / timski
KATEGORIJA:	unapredjenje / **nov**
VREME IZRADE:	(datum od, datum do): 21.10.2018. – 31.05. 2019.
UCESNICI:	a) RUKOVODILAC ... Viktorija Ekreš b) UCESNIK ... Viktorija Ekreš c) ZAINTERESOVANA STRANA ... Predškolska ustanova d) KORISNIK ... Roditelji sa malom decom
OPIS POSLA (tok poslovnih procesa):	Prilikom prijave deteta u predškolsku ustanovu, roditelj popunjava formular koji sadrži osnovne informacije o detetu, majci i ocu, njihovom porodičnom statusu (samohrani roditelj, teško oboleli roditelji, ratni invalidi itd..) Uz prijavu se prilaže potpisana izjava pod materijalnom i krivičnom odgovornošću da su upisani podaci tačni, kao i izjava o saglasnosti da se dati podaci dalje obrađuju. Nakon prijema prijave od roditelja, kreira se lista prioriteta dece za upis na osnovu podataka koji se nalaze u prijavi. Stvaraju se grupe dece, i raspoređuju po vrtićima u gradu.
SNIMAK STANJA:	
ANALIZA DOKUMENTACIJE:	**Pravna regulativa:** – Pravilnik o kriterijumima prijema dece u predškolsku ustanovu – Pravilnik o bližim uslovima za utvrđivanje prioriteta za upis dece **Obrasci dokumenata:** – Prijava za prijem dece u Predškolsku ustanovu Zrenjanin – Prijavna lista za upis dece u pripremni predškolski program – Izjava korisnika sredstava dečije zaštite – Pristanak za obradu podataka

	Standardi: – Međunarodni standardi PMBOK I SVEBOK
ANALIZA POSTOJEĆIH REŠENJA U SVETU U OVOJ OBLASTI:	/
ZAHTEVI KORISNIKA:	– validacija unosa podataka – tabelarni prikaz svih prijava – sortiranje dece po broju bodova – filtriranje dece po vrsti boravka – filtriranje dece po prezimenu
POTREBE KORISNIKA:	Online prijavljivanje deteta u Predškolsku ustanovu
INTERESI ZAINTERESOVANIH STRANA:	Jednostavnost i ekonomičnost procesa prijavljivanja deteta u Predškolsku ustanovu, lakše i brže sortiranje i raspoređivanje dece
PREDNOSTI OVOG RESENJA U ODNOSU NA RANIJA RESENJA:	Lakša realizacija upisivanja dece
OCEKIVANI REZULTATI	1. modeli – model funkcija, model podataka 2. baza podataka – SQL skript 3. program – korisnički interfejs i izvorni kod 4. dokumentacija – opis posla, snimak stanja, specifikacija zahteva korisnika, korisničko uputstvo
PLANIRANI OBUHVAT FUNKCIJA I PROFILI KORISNIKA:	Profil korisnika – admin: • Pregled prijava • Brisanje • Tabelarni prikaz • Sortiranje i filtriranje Profil korisnika – neimenovani korisnik sajta: • Mogućnost prijave deteta (unos podataka)
OGRANICENJA PROJEKTA I PROCENA RIZIKA:	- vreme izrade – kratko vreme izrade (RIZIK) - broj ljudi na projektu – samo jedan učesnik (RIZIK) - raspolozivi alati – na raspolaganju su, nema rizika - raspolozivi finansijski resursi – nema, ali nema potrebe ni rizika
INTEGRACIJA SA POSTOJECIM I NOVIM SISTEMOM:	Trenutno nema postojeće veb aplikacije.
METRIKE KVALITETA PROJEKTA:	karakteristike projekta: • Troskovi – nema troskova • Kvalitet – kvalitet softvera (baze podataka, programa, dokumentacije) • Vreme – realizovan projekat na vreme

	karakteristike programa: • Usklađenost rešenja sa strateškim ciljevima organizacije, potrebama i zahtevima korisnika i interesima zainteresovanih strana • Dizajn korisničkog interfejsa – funkcionalni (intuitivni, jednostavan za korišćenje, minimizacija napora i broja akcija za zeljeni cilj, interaktivan sa obaveštenjima, automatizmi kojima se ubrzava i skraćuje rad (uz korišćenje transakcija), vizualni (pregledan, prijatne boje) • Performanse – brzina rada • Pouzdanost – da ne puca, testiran na greške, ima kontrolu grešaka • Bezbednost podataka – bekap, kriptografija • Pogodnost za integraciju u postojeći sistem, pogodnost za održavanje (složenost rešenja, raspoloživost alata, otvoreni kod, raspoloživost osoblja za uslugu održavanja)
NACIN REALIZACIJE PROJEKTA:	
Nacin komunikacije ucesnika	Email
Rizici, SWOT analiza	**SWOT:** - Strength (snaga): prethodno predznanje u radu sa različitim tehnologijama - Weakness (slabost): samo 1 učesnik - Opportunity: dobra referenca u biografiji, iskustvo **RIZICI REŠENJA** Rizik ovog rešenja nastaje kasnijom upotrebom, jer veća količina fajlova koji bi se razmenjivali u okviru realizacije projekata može da utiče na performanse sajta Predškolske ustanove u celini.
Potrebna sredstva (alati, materijal)	Operativni sistem: Windows 10, Home Edition DBMS: MYSQL Jezik programiranja: TypeScript, Java Alat za programiranje: Visual Studio Code, IntelliJ IDEA Potrošni materijal: 1. papir 2. CD
Troskovi (alat, materijal, transport)	Sopstvena sredstva

Aktivnosti (ko, kad, sta radi)	Sve aktivnosti radi Viktorija Ekreš. Aktivnosti uključuju: - Formiranje baze podataka - Dizajn aplikacije - Kreiranje forme za unos podataka - Validacija podataka koji se unose - Postavljanje kriterijuma soritiranja i filtriranja - Postavljanje ograničenja na koji način se dobijaju bodovi

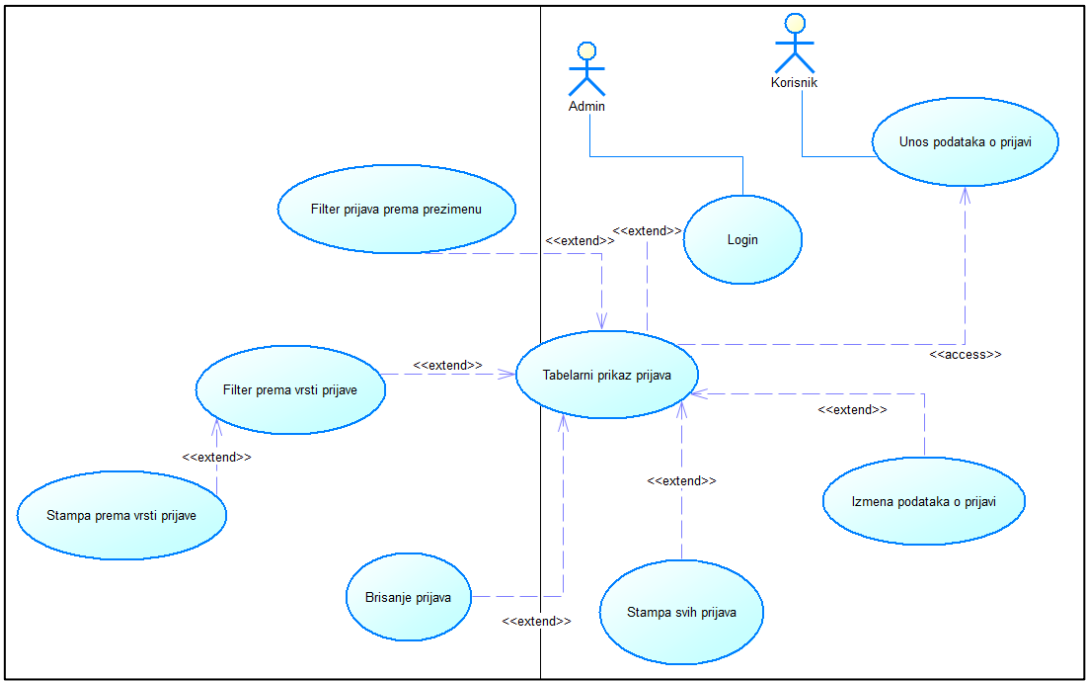
Tabela 2. Prijava projekta

6.2. UML dijagrami

Kompleksni sistemi i softver se ne mogu razumeti bez odgovarajućeg modelovanja. Danas, softver postaje veoma kompleksan i stoga da bi se mogao lakše razumeti, koriste se UML (Unified Modeling Language) dijagrami. Što je veći softver ili sistem, time modelovanje ima veće značenje. Kako nijedan model nije dovoljan da u celosti prikaže sve delove softvera i njegove funkcije, koristi se mali skup modela koji daju različite poglede na softver. [32]

Use-case dijagram

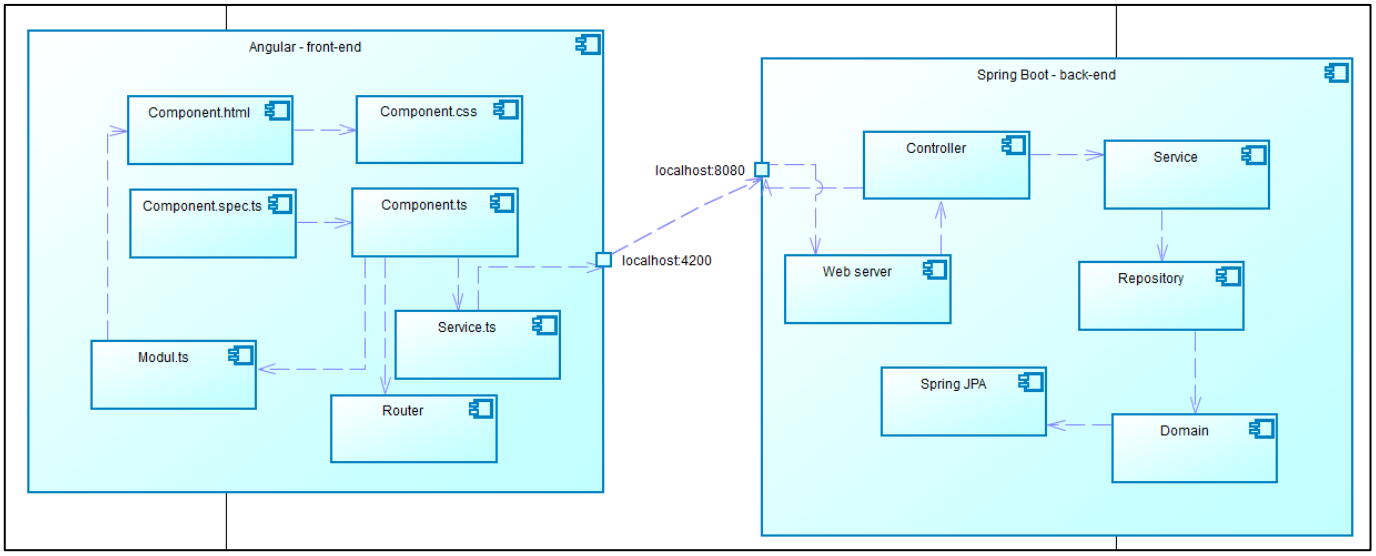
Kod use-case dijagrama definišu se slučajevi upotrebe koji predstavljaju različite funkcionalnosti sistema, i entiteti koji učestvuju u tim slučajevima upotrebe, odnosno akteri koji interaktuju sa sistemom. Na slici 18 je prikazan dijagram na kojem elipsasti oblici predstavljaju softverske funkcije koje aplikacija sadrži, a dva aktera predstavljaju korisnike aplikacije povezane sa funkcijama kojima mogu da pristupe.



Slika 18. Use case dijagram aplikacije predškolske ustanove

Dijagram komponenti

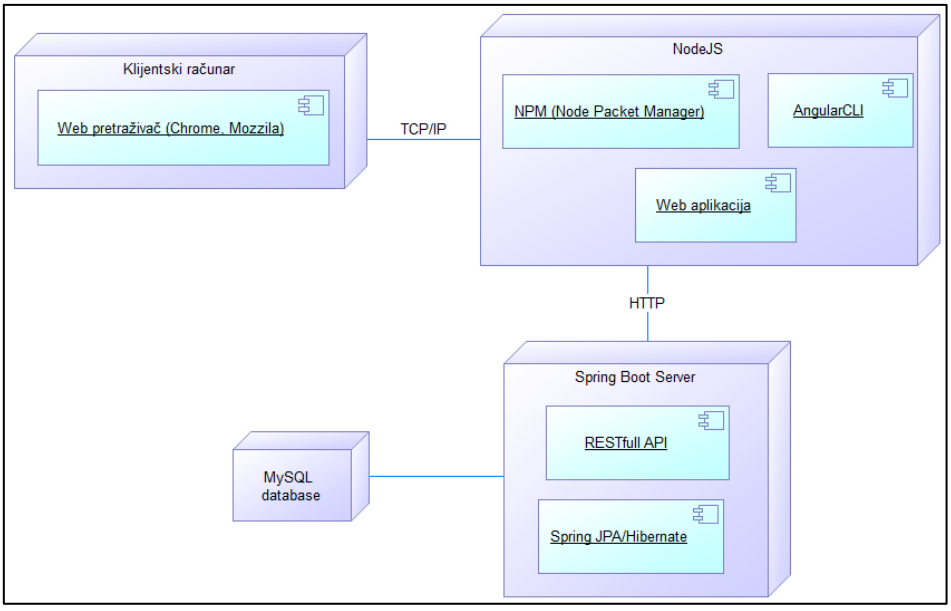
Na dijagramu komponenti se prikazuje organizacija i struktura modularnih delova sistema koji predstavljaju jednu fizičku celinu (komponente sistema, biblioteke, datoteke) i zavisnost između njih. Na slici 19 se može videti da je aplikacija podeljena na dve veće izolovane komponente, povezane preko 2 endpoint-a, koje u sebi obuhvataju manje komponente zadužene za implementaciju softverskih funkcija.



Slika 19. Prikaz dijagrama komponenti

Dijagram razmeštaja

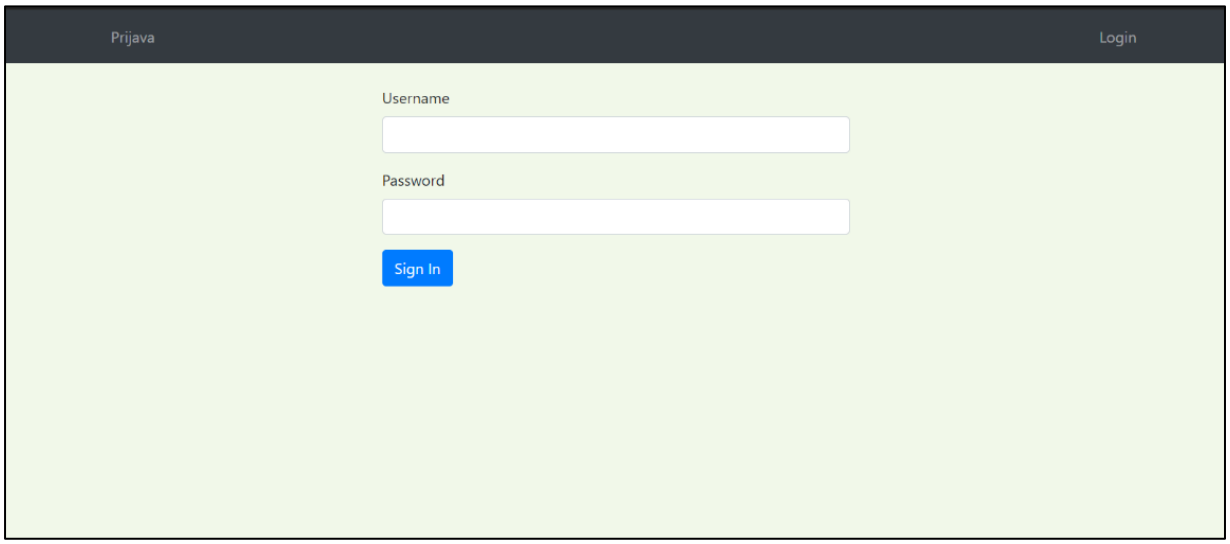
Dijagram razmeštaja služi za opisivanje veza između računarskih sistema (čvorova) koji učestvuju pri radu aplikacije i pokazuje tačno koji delovi softvera se nalaze na kom fizičkom delu sistema.



Slika 20. Prikaz dijagrama razmeštaja

6.3. Korisničko uputstvo

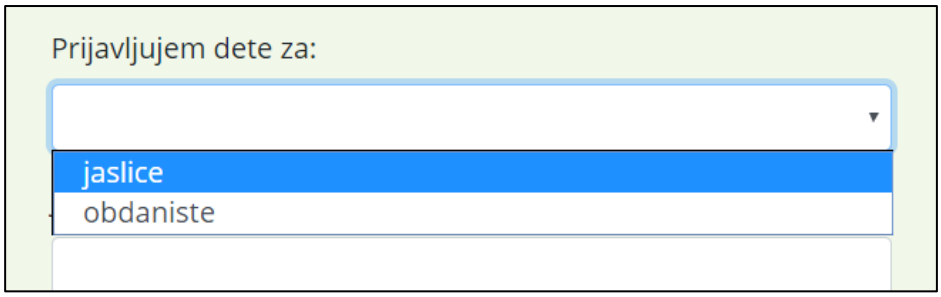
Pri pokretanju aplikacije, prva stranica koja se otvara je stranica za prijavljivanje admina na administrativni panel koja se može videti na slici 21.



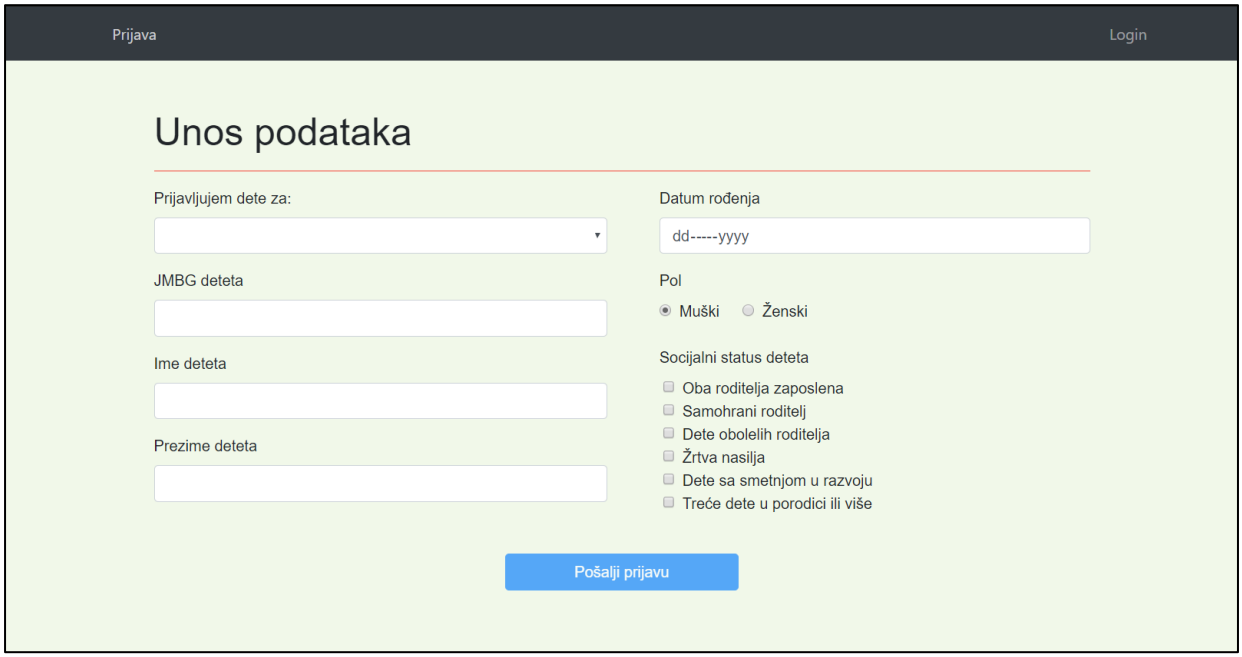
Slika 21. Prikaz početne stranice aplikacije

Neregistrovanim korisnicima nije dozvoljen pristup panelu, međutim, omogućena je opcija prijavljivanja deteta koja se nalazi u gornjem levom uglu na traci sa menijem.

Klikom na opciju „Prijava“, za prijavljivanje novog deteta u predškolsku ustanovu, otvara se nova stranica. Kao što se može videti sa slike 22, pojavljuje se forma za unos osnovnih informacija o detetu, kao što su JMBG, ime, prezime, datum rođenja, pol i vrsta boravka za koju se dete prijavljuje (slika 23). Vrste boravka koje postoje u ustanovi se ne upisuju već se biraju iz dropdown menija prikazanim na slici 22. Dodatna sekcija pod imenom „Socijalni status deteta“ je dodat kako bi deca koja su na neki način ugrožena imala prioritet pri upisu u predškolsku ustanovu po Pravilniku o bližim uslovima za određivanje prioriteta pri upisu. Dugme „Pošalji prijavu“ je onemogućeno sve dok korisnik ne potpuni celu formu.

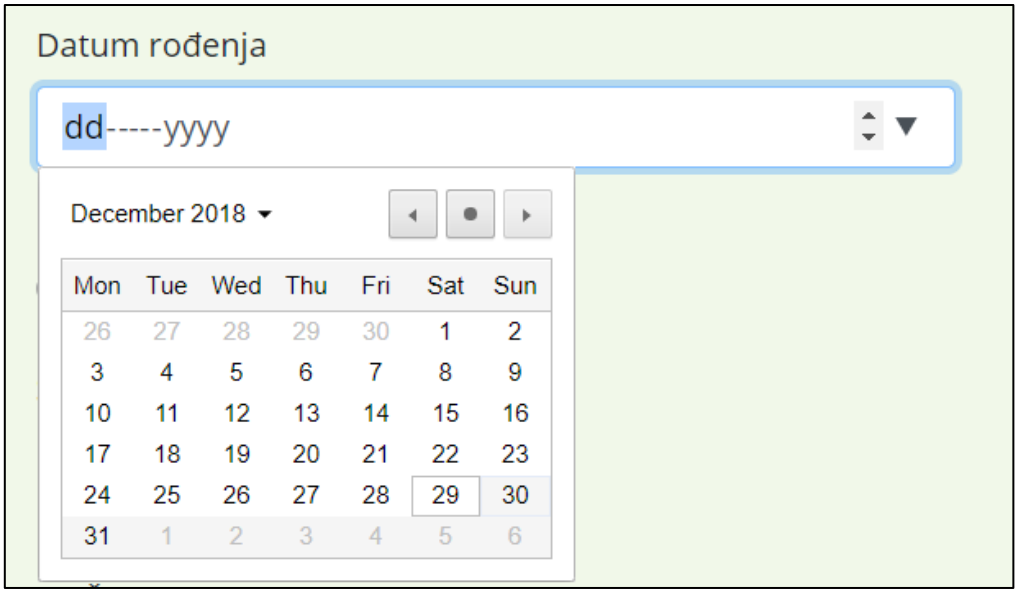


Slika 22. Prikaz dropdown opcija



Slika 23. Forma za unos podataka o detetu

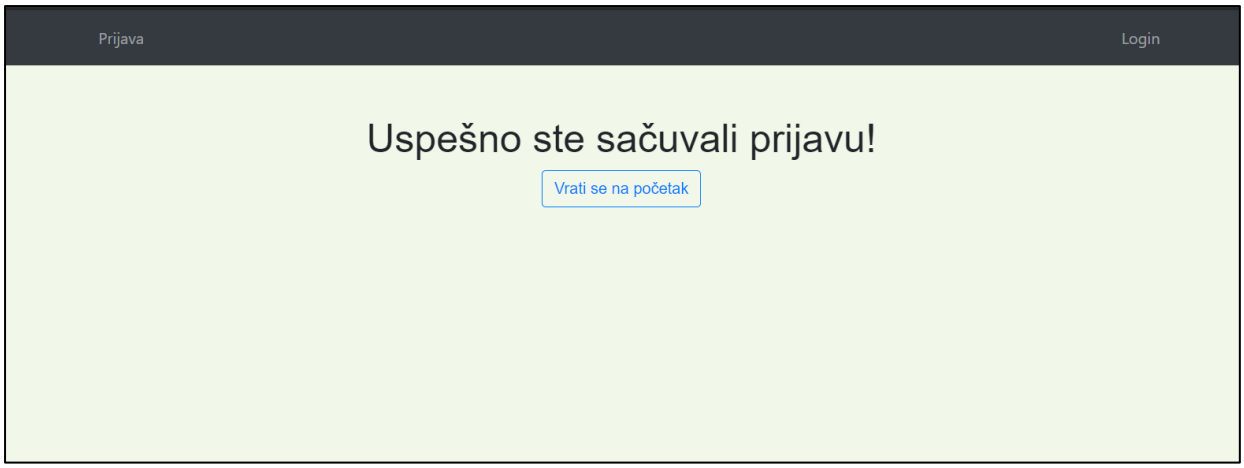
Klikom na polje za unos datuma rođenja, otvara se mali kalendar (slika 24) i bira se odgovarajući mesec, dan i godina. Kalendar je implementiran tako da se ne može izabrati datum koji je u budućnosti.



Slika 24. Kalendar za biranje datuma rođenja

Nakon što se popuni forma, automatski će dugme „Pošalji prijavu“ biti omogućeno za klik. Ukoliko je sve korektno popunjeno, odnosno upisani podaci su ispravni u smislu tipa odgovora koji se traži u datom polju, prijava će biti snimljena i korisnik će biti poslat na stranicu koja ga obaveštava da je prijava uspešno sačuvana, i nudi opciju povratka na početak tj. prvu stranicu

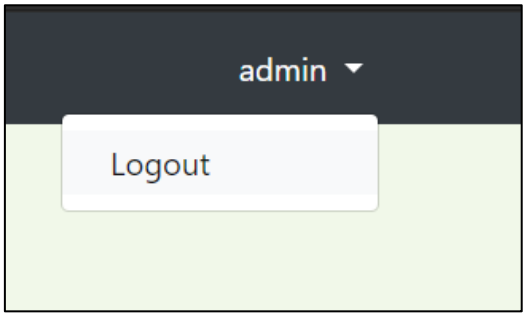
aplikacije. (slika 25) Međutim, ukoliko unet JMBG broj već postoji u bazi podataka, program će izbaciti grešku, dakle ne može se isto dete prijaviti više puta. (slika 26)



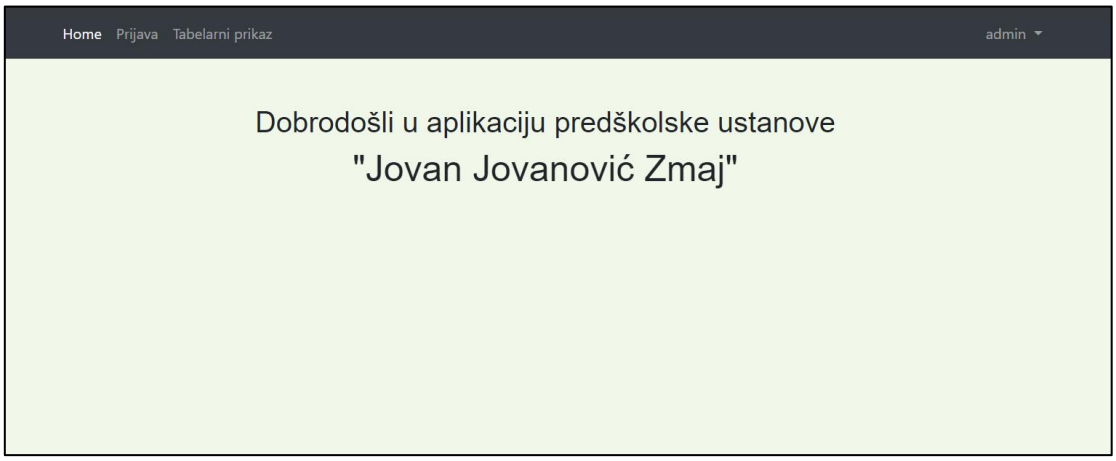
Slika 25. Prikaz poruke uspešnog snimanja

Slika 26. Greška prilikom unosa postojećeg JMBG-a

Nakon unosa, podaci koji su uneti se mogu videti u tabelarnom prikazu. Međutim, tabelarni prikaz kao i brisanje i izmena podataka je jedino dostupna administratorima aplikacije. Klikom na login opciju i unošenjem odgovarajućeg korisničkog imena i šifre, otvoriće se stranica prikazana na slici 28. Ono što se razlikuje od prikaza aplikacije neregistrovanim korisnicima jesu dodate stavke menija. Umesto „Login“ teksta, u gornjem desnom uglu se može videti ime korisnika koji je trenutno prijavljen na sistem. Ukoliko se klikne strelica pored imena korisnika, pojavljuje se opcija za odjavljivanje sa aplikacije. (slika 27) Klikom na opciju menija „Tabelarni prikaz“ otvara se stranica na slici 29.

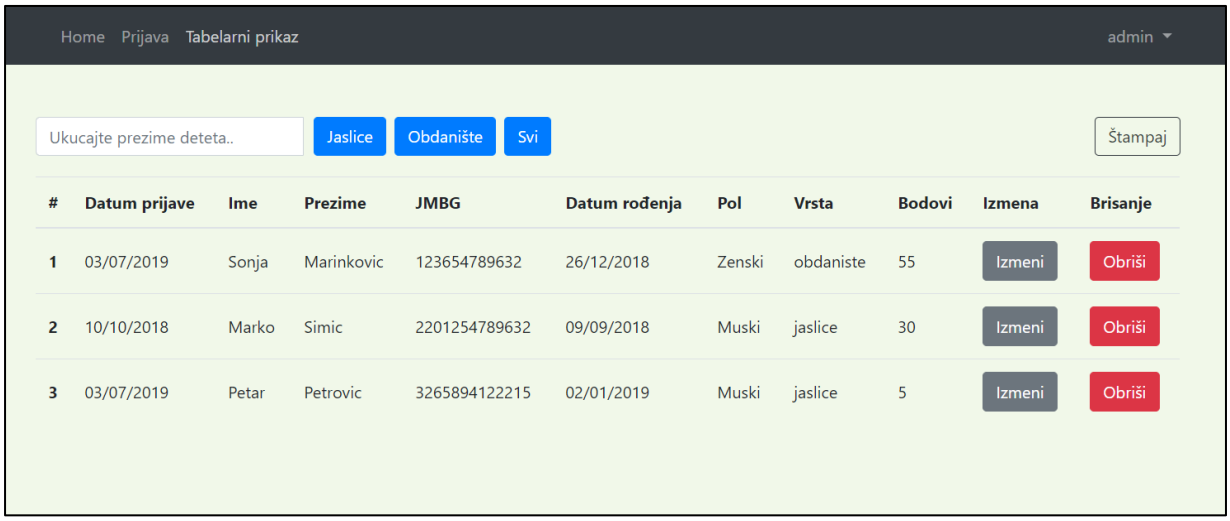


Slika 27. Opcija za logout

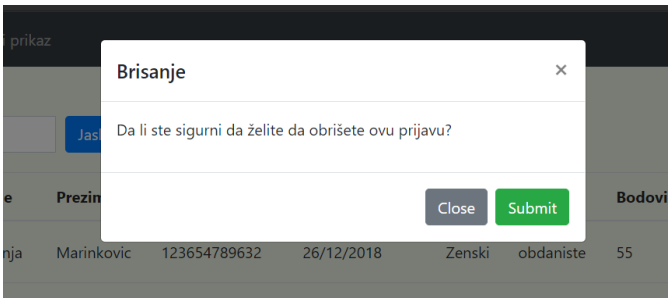


Slika 28. Početna stranica nakon logovanja

U tabeli se nalaze informacije o svim prijavama koje su uspešno sačuvane, automatski sortirane po broju bodova. Za svaku prijavu su prikazani: datum prijave, ime i prezime deteta, JMBG, datum rođenja, pol, vrsta prijave i dodeljeni bodovi. Na kraju svakog reda se nalaze dva dugmića „Izmeni“ i „Obriši“ koja služe za brisanje date prijave ili otvaranje forme za izmenu iste. Klikom na dugme obriši pojavljuje se modal koji proverava da li je korisnik siguran da želi da obriše datu prijavu ili je dugme bilo pritisnuto greškom. (slika 30)

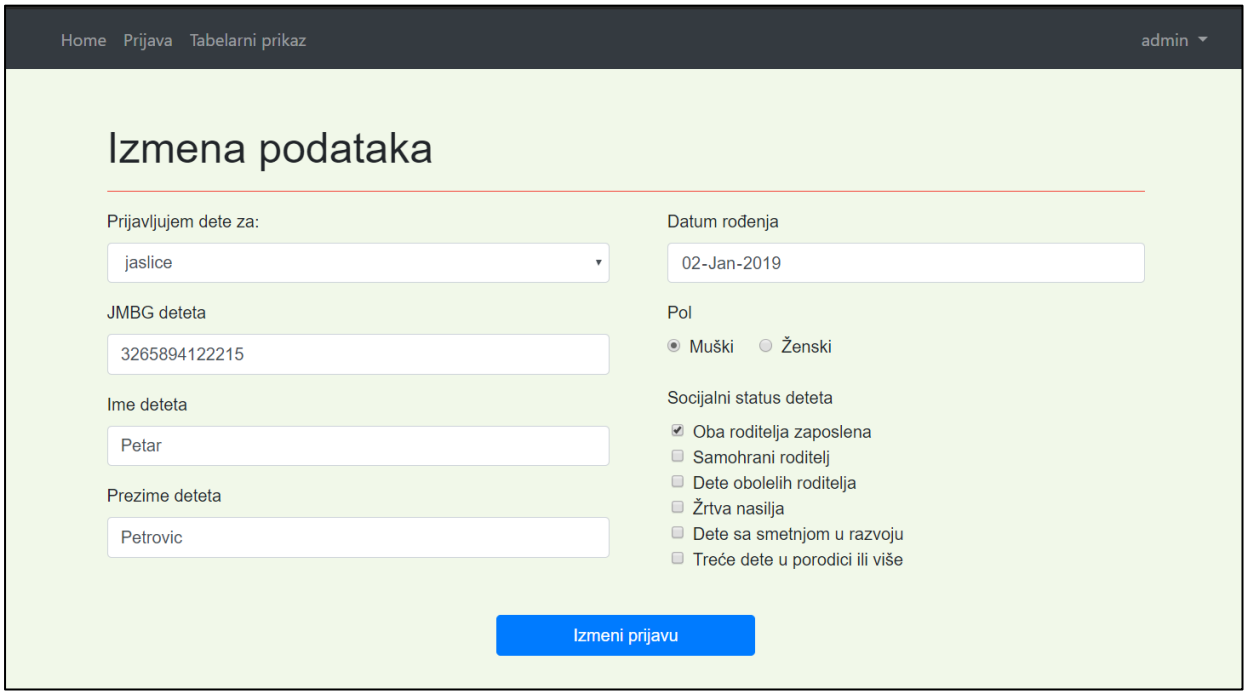


Slika 29. Tabelarni prikaz prijava



Slika 30. Modal za brisanje prijava

Klikom na zeleno dugme „Submit“, odabrana prijava se briše iz baze podataka, a klikom na sivo dugme „Close“ modal se zatvara i odustaje od brisanja. Klikom na dugme „Izmena“ otvara se nova forma sa popunjenim poljima podacima koji se trenutno nalaze u bazi. (slika 31)



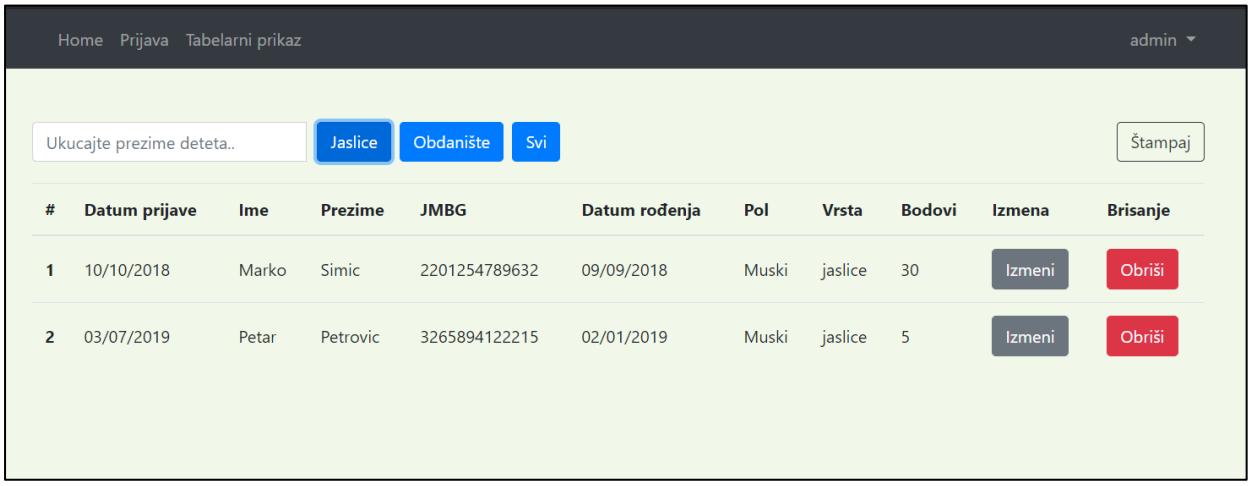
Slika 31. Forma za izmenu podataka

Moguće je izmeniti sve podatke osim datuma prijave koji se automatski definiše kada je dugme za čuvanje ili izmenu podataka pritisnuto. Nakon izvršene izmene, pojaviće se ista stranica za uspešno snimljenu prijavu kao i nakon upisa koja je bila prikazana ranije na slici 23.

Iznad tabele sa podacima se nalazi jedno polje koje služi za pretragu prijava po prezimenu deteta. Pretraga se vrši automatski u realnom vremenu čim počne da se kuca prezime. (slika 32) Brisanjem teksta iz polja, ponovo se pojavljuju sve postojeće prijave.

Slika 32. Prikaz pretrage prijava po prezimenu deteta

Pored polja za pretragu nalaze se 3 dugmića koja filtriraju prijave po njenoj vrsti. Klikom na dugme „Jaslice“ prikazuju se samo prijave za takvu vrstu boravka (slika 33), a klikom na dugme „Obdanište“ prikazuju se prijave samo za obdanište.

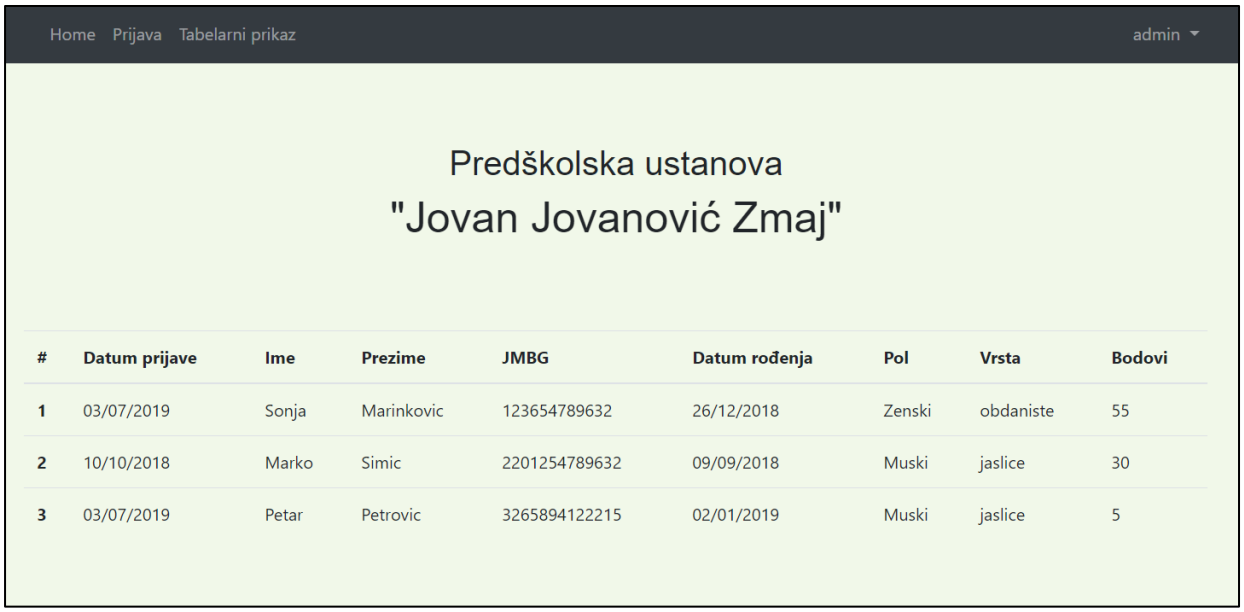


Slika 33. Filtrirani podaci po vrsti boravka - jaslice

Tako filtrirani podaci mogu i da se štampaju. Klikom na skroz desno dugme „Štampaj“ otvara se stranica za štampanje sa imenom predškolske ustanove i tabelom prijava, bez dugmića za brisanje i izmenu. U prikazanom slučaju je izabrana parametarska štampa sa vrstom prijave: jaslice. (slika 34)

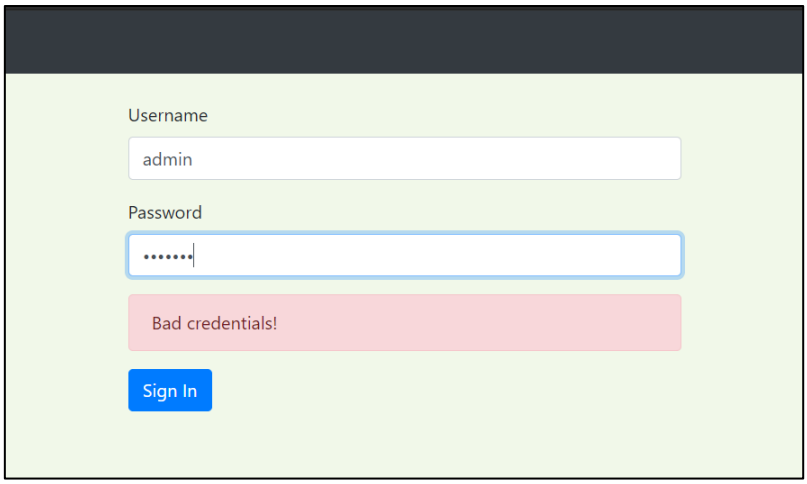
Slika 34. Parametarska štampa (vrsta prijave: jaslice)

Ukoliko je na stranici tabelarnog prikaza odabran filter „Svi“ i kliknuto dugme „Štampaj“, pojaviće se pripremna stranica za štampu sa svim prijavama. (slika 35)



Slika 35. Štampa svih prijava

Ukoliko se pri logovanju upišu pogrešni podaci za korisničko ime i šifru, aplikacija će izbaciti poruku sa greškom „Bad credentials!“ i odbiće pristup administrativnom panelu.



Slika 36. Greška prilikom logovanja

6.4. Implementacija

Izrađena aplikacija prikazuje upotrebu modernih veb tehnologija poput Angular razvojnog okruženja zasnovan na komponentama koji odvajaju prikaz stranice od njene funkcionalnosti, i Spring Boot okruženja za upravljanje podacima baze podataka.

6.4.1. Baza podataka

Spring Boot prilikom pokretanja projekta pokušava automatski da konfiguriše bazu podataka tako što pristupa podacima vezanim za bazu u dokumentu „application.properties.“ Međutim, da bi to bilo moguće, u datoteci pod nazivom „build.gradle“ se moraju upisati odgovarajuće biblioteke koje su potrebne za rad sa JPA (Java Persistence API) i MySQL. Svaki put kada se aplikacija pokrene, „build.gradle“ datoteka se takođe pokreće i sa sobom povlači sve neophodne biblioteke upisane unutar nje. One se moraju pisati pod komandom „dependencies.“

```
dependencies {
    compile('org.springframework.boot:spring-boot-starter-veb');
    compile('org.springframework.boot:spring-boot-starter-data-jpa');
    compile('org.springframework.boot:spring-boot-starter-security');
    compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.44'
}
```

Listing 4. Biblioteke potrebne za rad sa JPA i MySQL

Nakon ubacivanja odgovarajućih biblioteka, unutar dokumenta „application.properties“ se upisuju parametri za konekciju sa bazom podataka.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.url=jdbc:mysql://localhost:3306/predskolska?useSSL=false
spring.jpa.show-sql=true
```

Listing 5. Parametri konekcije sa bazom podataka

Hibernate predstavlja popularan ORM (Object Relational Mapping) framework, i služi za mapiranje Java objekata u tabele baze podataka i obrnuto. Na ovaj način, programeri ne moraju da pišu optimizovan SQL kod, već objekti koji su generisani to rade umesto njih. Hibernate implementuje JPA specifikaciju, odnosno set standarda za upravljanje relacionom bazom podataka. Unutar specifikacije su definisane metode opisivanja entiteta, mapiranja atributa i mapiranja veza između entiteta koje Hibernate razume i na osnovu kojih osigurava čuvanje i čitanje podataka iz baze. [33]

Prva naredba u listingu 5 se odnosi na inicijalizaciju baze, i umesto „update“ ona može biti: create, create-drop i validate. „Update“ znači da će se tabele automatski kreirati u bazi nakon kreiranja domenskog modela i polja unutar tog modela će biti mapirani sa odgovarajućim kolonama u tabeli. Bilo koja promena u modelu, rezultovaće promeni i na mapiranoj tabeli baze.

Jedan domenski model treba da predstavlja jednu tabelu u bazi podataka. Unutar modela se pišu anotacije kako bi se definisale tabele, kolone i druge osobine. Najbitnije anotacije koje se koriste su [33]:

- **@Entity**, koja se mora napisati na početku svakog domenskog modela jer označava Java klasu.
- **@Table**, koristi za navođenje detalja tabele koja se mapira u bazi. U zagradama se piše ime tabele.
- **@Column**, koristi se za navođenje detalja o koloni tabele, a unutar zagrade se mogu takođe i dodati druge osobine kao što je dužina, da li je dozvoljeno da polje bude null itd.
- **@Id**, anotacija koja služi za definisanje primarnog ključa
- **@GeneratedValue**, anotacija koja definiše kako se primarni ključ generiše (auto increment na primer)
- **@JoinColumn**, koristi se za spajanje dve tabele. U paru sa ovom anotacijom se koriste se anotacije koje označavaju kardinalnost @ManyToOne, @OneToOne, @ManyToMany.
- **@NotNull**, koje označava da polje ne sme biti prazno.

U izrađenoj aplikaciji za predškolsku ustanovu napravljena je jedna posebna apstraktna klasa pod imenom „BaseEntity“ koja sadrži samo id atribut, i sve ostale klase je nasleđuju. Uz pomoć anotacije @MappedSuperclass neće se kreirati tabela u bazi za ovu klasu. Kako je sve ostale klase nasleđuju, unutar njih se ne moraju definisati primarni ključevi jer taj atribut preuzimaju iz bazne klase.

```
package predskolska.Domain;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;

@MappedSuperclass
public abstract class BaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

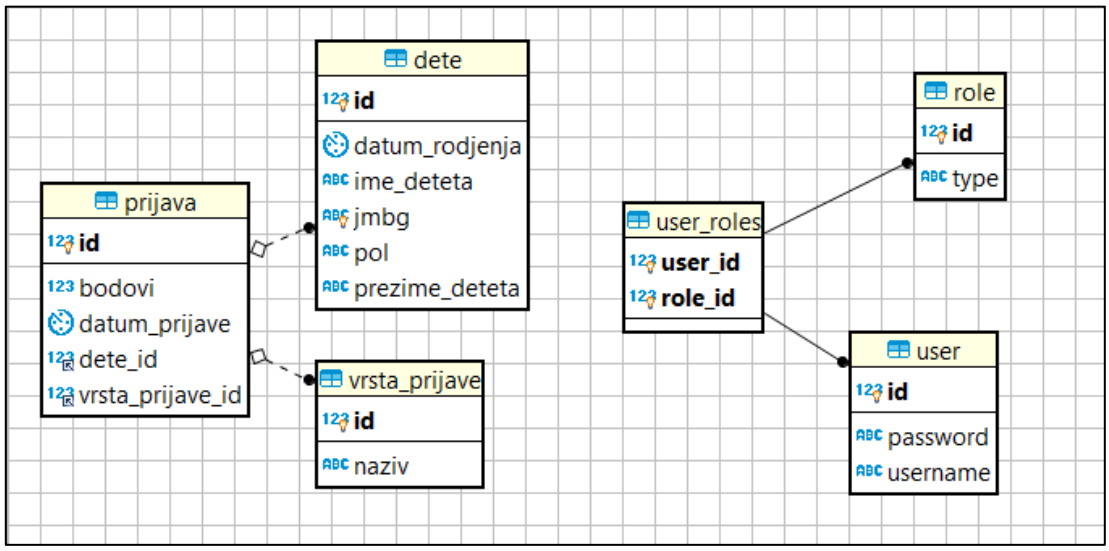
Listing 6. Apstraktna klasa BaseEntity

Primer domenskog modela za prijavu se može videti u listingu 7.

```java
package predskolska.Domain;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.io.Serializable;
import java.util.Date;

@Entity
@Table(name = "prijava")
public class Prijava extends BaseEntity implements Serializable {

    @Column
    private Date datumPrijave;

    @Column
    private Long bodovi;

    @NotNull
    @ManyToOne(targetEntity = VrstaPrijave.class)
    @JoinColumn
    private VrstaPrijave vrstaPrijave;

    @NotNull
    @OneToOne(targetEntity = Dete.class)
    @JoinColumn
    private Dete dete;

    public Dete getDete() {
        return dete;
    }

    public void setDete(Dete dete) {
        this.dete = dete;
    }

    public Date getDatumPrijave() {
        return datumPrijave;
    }

    public void setDatumPrijave(Date datumPrijave) {
        this.datumPrijave = datumPrijave;
    }

    public Long getBodovi() {
        return bodovi;
    }

    //ostale get i set metode
}
```

Listing 7. Primer domenskog modela koji kreira tabelu „dete“ u bazi podataka

Nakon kreiranja svih domenskih modela, automatski se generiše baza podataka čije veze se mogu videti na slici 37.



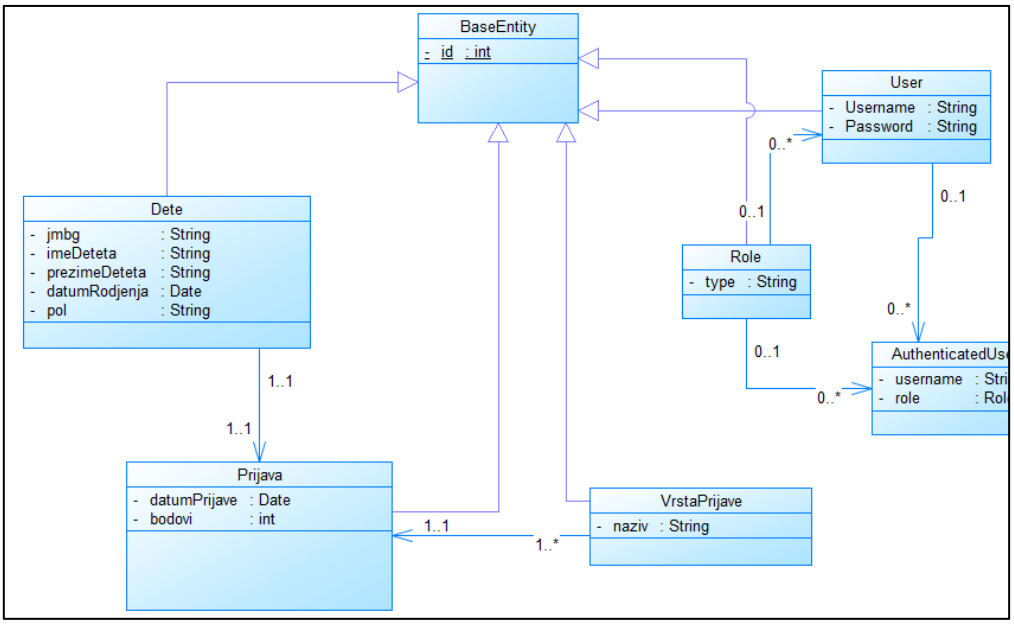
Slika 37. Šema baze podataka

6.4.2. UML modeli za opis implementacije

U nastavku će biti opisane i prikazane dve vrste UML dijagrama (dijagram klasa i dijagram sekvenci) koji služe za vizuelizaciju unutrašnje strukture aplikacije, odnosno način na koji je implementirano dato softversko rešenje.

Dijagram klasa

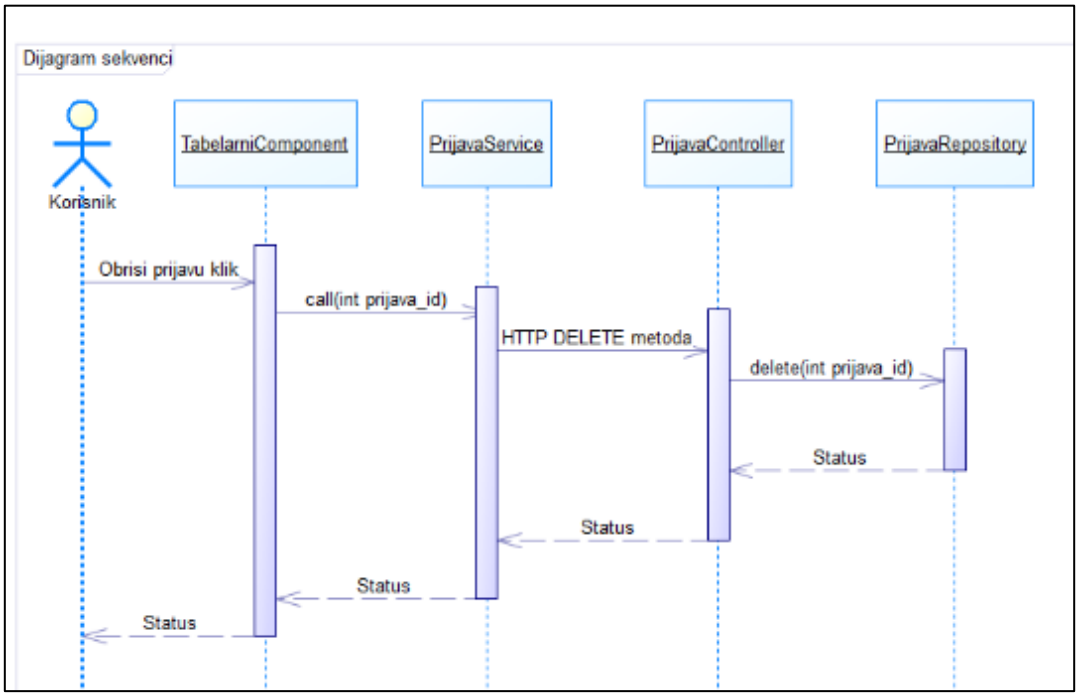
Uzimajući u obzir da su objektno-orijentisani sistemi zasnovani na klasama, dijagram klasa je pogodan za vizuelizaciju i uvid u klase koje aplikacija sadrži, prikaz njenih metoda i atributa. Klase mogu da budu apstraktne, da nasleđuju jedna drugu, ili da zavise jedna od drugih. Dijagram klasa takođe omogućava i uvid u veze između klasa i način na koji zavise jedna od druge.



Slika 38. Prikaz dijagrama klasa

Dijagram sekvenci

Dijagram sekvenci predstavlja dijagram interakcije koji opisuje ponašanje sistema definišući vremenski sled događaja unutar jednog slučaja korišćenja, dakle, kako i kojim redosledom funckionišu objekti u sistemu. Koristi sistem poruka za komunikaciju između objekata. Na slici 39 je prikazan dijagram sekvenci za slučaj brisanja prijave iz baze podataka.



Slika 39. Prikaz dijagrama sekvenci

6.4.3. Tabelarni prikaz višeslojne arhitekture softvera

Realizovano rešenje je višeslojna veb aplikacija izrađena prema MVC dizajn paternu. U narednoj tabeli se može videti tehnološka implementacija različitih slojeva aplikacije, sa referencama koje vode na određene listinge i sekcije gde se ta implementacija može videti u samom kodu.

GLAVNI SLOJ	**MVC dizajn patern**	**PODSLOJ**	**TEHNOLOŠKA IMPLEMENTACIJA**
PREZENTACIONI SLOJ	VIEW	Korisnički interfejs	Angular
	CONTROL	Klase prezentacione logike	Angular komponenta Primer prijava.component.ts Listing 13.
SERVISNI SLOJ		Veb servis	Spring Controller koji uslužuje HTTP zahteve od Angulara

			Primer: Sekcija 5.5.3.
		Klase za mapiranje slojeva	Angular komponenta Primer prijava.service.ts - Listing 12.
SLOJ POSLOVNE LOGIKE		Radni tokovi	
	MODEL	Poslovni objekti	Domenski modeli odnosno klase koji predstavljaju poslovne objekte (entitete pojmova realnog sveta, dokumente) koji se po nazivu razlikuju od naziva tabela iz baze podataka. Primer dete.java Listing 4.
		Poslovna pravila	Klasa sa algoritmom za proveru poslovnog pravila poslovnaLogika.service.ts (Listing 14)
SLOJ ZA RAD SA PODACIMA		Rad sa relacionom bazom podataka - Klase modela I repository za rad sa podacima iz tabela relacione baze podataka - DBMS sa bazom podataka	JPA/Hibernate ORM framework Relaciona baza podataka sa tabelama i relacijama
		Rad sa drugim formatima podataka XML, XLS, JSON	

Tabela 3. Prikaz višeslojne arhitekture softvera

6.4.4. Ključni delovi programskog koda

Ključni delovi koda podrazumevaju najbitnije sekcije koje predstavljaju srž aplikacije i omogućavaju njenu osnovnu funkcionalnost. Iz ove sekcije izostavljeni su delovi koda koji se ponavljaju, u smislu da se ne prikazuju klase koje su po svojoj strukturi iste (na primer, kontroler za dva različita domenska modela, jer se kontroler definiše na standardni način, i nema potrebe prikazivati ga ponovo sa drugim podacima).

6.4.4.1. Repozitorijum

Kako je u sekciji 5.2. u listingu 7 prikazan domenski model za klasu prijava, u ovoj sekciji će biti prikazan primer repozitorijuma za istu klasu koji služi za pristup podacima iz baze. Uz pomoć JpaRepository interfejsa, osnovne CRUD operacije nad klasom su već definisane. Sve što je potrebno u Repository sloju je da se poveže sa JpaRepository interfejsom. Takođe, mogu se dodavati i nove operacije kao što je prikazano u listingu 8, gde se može videti definisana operacija findAllByVrstaPrijave pomoću koje se filtriraju podaci prema vrsti prijave.

```
package predskolska.Repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import predskolska.Domain.Prijava;
import predskolska.Domain.VrstaPrijave;

import java.util.List;

@Repository
public interface PrijavaRepository extends JpaRepository<Prijava, Long> {

    List<Prijava> findAllByVrstaPrijave(VrstaPrijave vrstaPrijave);
}
```

Listing 8. Primer repozitorijuma za klasu Prijava

6.4.4.2. Servis

Nakon toga definiše se sloj servisa, u kojem se postavlja anotacija @Transactional koja naglašava da se operacije izvršavaju tako što se prvo jedna završi pre nego što druga može da počne.

```
package predskolska.Service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import predskolska.Domain.Prijava;
import predskolska.Domain.VrstaPrijave;
import predskolska.Repository.PrijavaRepository;

import javax.transaction.Transactional;
import java.util.List;
```

```
@Transactional
@Service
public class PrijavaService {
    private PrijavaRepository prijavaRepository;

    @Autowired
    public PrijavaService(PrijavaRepository prijavaRepository) {
        this.prijavaRepository = prijavaRepository;
    }

    public Prijava save(Prijava prijava) {
        return prijavaRepository.save(prijava);
    }

    public List<Prijava> findAll() {
        return this.prijavaRepository.findAll();
    }

    public List<Prijava> findAllByVrstaPrijave(VrstaPrijave vrstaPrijave)  {
        return this.prijavaRepository.findAllByVrstaPrijave(vrstaPrijave);
    }

    public Prijava findOne(Long id) {
        return prijavaRepository.findOne(id);
    }

    public void delete(Long id) {
        prijavaRepository.delete(id);
    }
}
```

Listing 9. Prikaz servisa za klasu Prijava

6.4.4.3. Kontroler

Kontroler se obeležava sa anotacijom @RestController i u njemu se definišu sve metode koje primaju HTTP zahteve za kreiranje, brisanje, izmenu i čitanje podataka. Pod anotacijom @RequestMapping se definiše početak URL-a za sve metode koje se nalaze unutar tog kontrolera.

```
@RestController
@RequestMapping("/prijava")
```

Listing 10. Anotacije za definisanje kontrolera i URL-a

Za GET metodu koristi se anotacija @GetMapping. Na sledećem primeru se može videti metoda za prikaz jedne prijave koristeći njen identifikator. Metoda vraća ResponseEntity, odnosno HTTP odgovor korisniku, u kojem se definiše da li je zahtev uspešno izvršen.

```
@GetMapping(path = "/{id}")
    public ResponseEntity findOne(@PathVariable("id") Long id) {
        Prijava prijava = prijavaService.findOne(id);
        if(prijava == null) {
            return new ResponseEntity(HttpStatus.NOT_FOUND);
        }
```

```
        return new ResponseEntity(prijava, HttpStatus.OK);
    }
```

Listing 11. Prikaz GET metode

Za POST metodu koristi se anotacija @PostMapping. U listingu 12 je navedena metoda za čuvanje prijave u bazu podataka.

```
@PostMapping
    public Prijava save(@RequestBody Prijava prijava) {
        return prijavaService.save(prijava);
    }
```

Listing 12. Prikaz POST metode

Za PUT metodu koristi se anotacija @PutMapping. PUT metoda se može koristiti i za kreiranje i za izmenu podataka, međutim najbolje je PUT metodu koristiti za izmenu, a POST za kreiranje iz tog razloga što ukoliko se pokuša poslati isti zahtev više puta, PUT će svaki put menjati isti resurs, dok kod POST metode se može dogoditi da se kreira isti resurs više puta. Sledeći primer prikazuje metodu za izmenu podataka o prijavi.

```
@PutMapping
    public Prijava update(@RequestBody Prijava prijava) {
       return prijavaService.save(prijava);
    }
```

Listing 13. Prikaz PUT metode

Za brisanje se koristi metoda DELETE, i označava se anotacijom @DeleteMapping. Primer se može videti u listingu 14.

```
@DeleteMapping(path = "/{id}")
    public ResponseEntity delete(@PathVariable("id") Long id) {
        prijavaService.delete(id);
        return new ResponseEntity(HttpStatus.OK);
    }
```

Listing 14. Prikaz DELETE metode

6.4.4.4. Component service

U sledećem listingu je dat primer za servis podataka koji uz pomoć HTTP protokola komunicira sa Spring servisom. Najpre je naveden URL veb servera koji radi na localhost:8080 i ruta koja je definisana u odgovarajućem kontroleru. U ovom slučaju je to http://localhost:8080/prijava.

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { Prijava } from "./prijava.model";
import { Observable } from "rxjs/Observable";
import { VrstaPrijave } from "./vrstaPrijave.model";

@Injectable()
export class PrijavaService {

    URLprijava = "http://localhost:8080/prijava"
```

```
    constructor(private httpClient: HttpClient) {}

    dajSvePrijave(): Observable<Prijava[]> {
        return this.httpClient.get<Prijava[]>(this.URLprijava);
    }

    dajPrijavu(id: number): Observable<any> {
        return this.httpClient.get(this.URLprijava + '/' + (id));
    }

    savePrijava(prijava: Prijava) {
        return this.httpClient.post(this.URLprijava, prijava);
    }

    updatePrijava(prijava: Prijava) {
        return this.httpClient.put(this.URLprijava, prijava);
    }

    deletePrijava(id: number) {
        return this.httpClient.delete(this.URLprijava + '/' + (id));
    }

    dajPrijavuPoVrsti(id: number): Observable<Prijava[]> {
        return this.httpClient.get<Prijava[]>(this.URLprijava + '/lista/' +
(id));
    }

}
```

Listing 15. Primer servisa za klasu Prijava

Metode iz servisa su pozivane od strane datoteke prijava.component.ts. Na sledećem primeru je prikazana metoda za čuvanje prijave koja poziva servis podataka sa listinga 15, i njegovu metodu „savePrijava.“

```
onSubmit(form: NgForm) {
    const jmbg = form.value.jmbg;
    const ime = form.value.imeDeteta;
    const prezime = form.value.prezimeDeteta;
    const datum = form.value.datumRodjenja;
    const pol = form.value.pol;
    this.dete = new Dete(null, jmbg, ime, prezime, datum, pol);
    this.deteService.saveDete(this.dete).subscribe(
      (data) => {
        this.handlePrijavaSuccess(data);
      },
      (error) => {
        this.handlePrijavaError(error);
      }
    );
  }

handlePrijavaSuccess(data) {
    this.novaPrijava = new Prijava(null, new Date(), this.brojBodova,
this.vrsta, data);
    this.prijavaService.savePrijava(this.novaPrijava).subscribe(
```

```
      (data) => {
        this.route.navigate(['/success-page']);
      }
    );
  }

handlePrijavaError(error) {
    this.error = error;
    this.error.message = "Taj JMBG već postoji u bazi!"
  }
```

Listing 16. Deo koda iz fajla prijava.component.ts za čuvanje prijave

6.4.4.5. Login

Kako bi se ostvarila mogućnost prijavljivanja na aplikaciji, potrebno je napraviti jednu klasu za korisnika sa atributima za korisničko ime i šifru, drugu klasu za vrstu nadležnosti koju ima (admin, običan korisnik) i jednu klasu za autentifikovanog korisnika.

```
@Entity
public class User extends BaseEntity {

    @Column(nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @ManyToMany
    @JoinTable(joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles;
```

Listing 17. Prikaz dela klase korisnik

Na listingu broj 18 je prikazana klasa za ulogu korisnika.

```
package predskolska.Domain;

import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;

@Entity
public class Role extends BaseEntity {

    @Enumerated(EnumType.STRING)
    private RoleType type;

    public RoleType getType() {
        return type;
    }

    public void setType(RoleType type) {
        this.type = type;
```

```
    }

    public enum RoleType {
        ROLE_USER, ROLE_ADMIN
    }
}
```

Listing 18. Klasa Role

Koristeći već postojeći WebSecurity Spring dependency koji je unet na početku ovog projekta, unutar UserService servisa koji nasleđuje postojeću klasu UserDetailsService iz navedenog dependency-ja, menja se njegova funkcija loadUserByUsername (što se iz priloženog listinga može videti uz pomoć anotacije @Override, gde se zatim dodeljuje autoritet korisnicima. (listing 19)

```
@Transactional
@Service
public class UserService  implements UserDetailsService {

    private UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        try {
            User user = userRepository.findByUsername(username);
            if(user == null) {
                return null;
            }
            return new
org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), getAuthorities(user));
        } catch (Exception e) {
            throw new UsernameNotFoundException("User not found.");
        }
    }

    private Set<GrantedAuthority> getAuthorities(User user) {
        Set<GrantedAuthority> authorities = new HashSet<>();
        for(Role role: user.getRoles()) {
            GrantedAuthority grantedAuthority = new
SimpleGrantedAuthority(role.getType().toString());
            authorities.add(grantedAuthority);
        }
        return authorities;
    }
}
```

Listing 19. Klasa UserService

Na frontend strani, definisan je servis koji vrši autentifikaciju korisnika prikazan na listingu 20.

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpHeaders, HttpClient } from '@angular/common/http';
import { tap } from 'rxjs/operators';

export interface User {
    username: string;
    roles: string[];
}

@Injectable()
export class AuthService {
user: User;
private authenticated = false;
private headers;

constructor(private httpClient: HttpClient, private router: Router) {}

login(username: string, password: string) {
    const base64Credential = btoa(username + ':' + password);
    const headers = new HttpHeaders({
    authorization: 'Basic ' + base64Credential
    });
    return this.httpClient
    .get<User>('http://localhost:8080/user', { headers: headers })
    .pipe(
        tap(user => {
        this.user = user;
        this.headers = headers;
        this.authenticated = true;
    }));
}
```

Listing 20. Klasa AuthService

Login komponenta, odnosno *.ts datoteka login komponente koristi AuthServise da proveri unete podatke u input poljima koji se nalaze u *.html datoteci login komponente. (listing 21)

```
export class LoginComponent implements OnInit {

  error: Error;

  constructor(private authService: AuthService, private router: Router) { }

  ngOnInit() {
  }

  onLogin(form: NgForm) {
    const username = form.value.username;
    const password = form.value.password;
    this.authService.login(username, password)
```

```
      .subscribe(
        () => this.router.navigate(['/home']),
        (error) => {
          this.error = error;
          //console.error(error);
        }
      );
  }
}
```

Listing 21. Klasa LoginComponent

Takođe, potrebna je jedna klasa koja određuje koji korisnik može da pristupi kojim stranicama. Nazvana je AuthGuard, i ukoliko osoba koja nema autentifikaciju za određenu stranicu pokuša da se prijavi na aplikaciju, odbiće joj pristup i automatski će je prebaciti na /home stranicu. (listing 22)

```
import { Injectable } from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot }
from '@angular/router';
import { AuthService } from './login/auth.service';

@Injectable()
export class AuthGuard implements CanActivate {

    constructor(private authService: AuthService, private router: Router) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const expectedRole = route.data.expectedRole;

        if (this.authService.isAuthenticated()) {
            if (expectedRole === 'user' || (expectedRole === 'admin' &&
this.authService.hasRoleAdmin())) {
                return true;
            }
            this.router.navigate(['/home']);
            return false;
        }
        this.router.navigate(['/login']);
        return false;
    }
}
```

Listing 22. Klasa AuthGuard

6.4.4.6. Poslovna logika

Klasa zaslužna za izračunavanje bodova prijave se nalazi u datoteci poslovnaLogika.service.ts i kod koji ona sadrži je prikazan na sledećem listingu.

```
import { Injectable } from "@angular/core";
import { ActivatedRoute } from "@angular/router";

@Injectable()
export class PoslovnaLogikaService {
```

```
    selectedValue: number;
    CheckboxesList = [
      {type: 'Oba roditelja zaposlena', value: 5},
      {type: 'Samohrani roditelj', value: 10},
      {type: 'Dete obolelih roditelja', value: 20},
      {type: 'Žrtva nasilja', value: 35},
      {type: 'Dete sa smetnjom u razvoju', value: 35},
      {type: 'Treće dete u porodici ili više', value: 25},
    ];

    constructor() {
        this.selectedValue = 0;

    }

    izracunajBodove(event, checkedElement){

        if(event.target.checked){
          this.selectedValue = this.selectedValue + checkedElement.value;
        }
        else {
          this.selectedValue = this.selectedValue - checkedElement.value;
        }
        return this.selectedValue;
      }
}
```

Listing 23. Primer poslovne logike

6.4.4.7. Filter pipe

Pipes predstavljaju posebnu vrstu Angular klase koja se koristi da preuzima neku vrednost i transformiše je u odgovarajući izlaz. Postoje neke ugrađene klase na primer za transformisanje datuma, ali mogu se praviti i nove. Tako je u ovom slučaju napravljen novi *Pipe* za pretragu prijava po prezimenu dece. Klasa preuzima pojam koji se pretražuje i prikazuje odgovarajuću prijavu u tabeli u realnom vremenu.

```
import { PipeTransform, Pipe } from "@angular/core";
import { Prijava } from "./prijava.model";

@Pipe({
    name: 'prijavaFilter'
})
export class PrijavaFilterPipe implements PipeTransform {
    transform(prijave: Prijava[], searchTerm: string): Prijava[] {
        if(!prijave || !searchTerm) {
            return prijave;
        }
        return prijave.filter(prijave =>
prijave.dete.prezimeDeteta.toLowerCase().indexOf(searchTerm.toLowerCase()) !==
-1);
    }
}
```

Listing 24. Prikaz koda za pretragu prijava prema prezimenu

6.4.4.8. Tabelarni prikaz

Na sledećem primeru je prikazana komponenta tabelarni.component.ts, odnosno kod koji stoji iza nje. U spomenutoj datoteci se nalaze metode za brisanje i izmenu podataka, kao i filter prema određenoj vrsti prijave. Takođe se nalazi metoda koja vodi na komponentu „Štampa“ i u zavisnosti od odgovarajućeg filtera prikazuje novu tabelu za štampanje.

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { PrijavaService } from '../shared/prijava.service';
import { Prijava } from '../shared/prijava.model';
import { Router } from '@angular/router';
import { Dete } from '../shared/dete.model';

@Component({
  selector: 'app-tabelarni',
  templateUrl: './tabelarni.component.html',
  styleUrls: ['./tabelarni.component.css']
})
export class TabelarniComponent implements OnInit {

  prijave$: Observable<Prijava[]>
  selektovanaPrijava: Prijava = new Prijava(null, null, null, null, null);
  searchTerm: string;
  selektovanaVrsta: number;

  constructor(private prijavaService: PrijavaService, private router: Router)
{ }

  ngOnInit() {
    this.prijave$ = this.prijavaService.dajSvePrijave();
    this.selektovanaVrsta = 0;
  }

  onBrisanje(prijava: Prijava) {
    this.selektovanaPrijava = prijava;
  }

  onBrisanjeSubmit() {
    this.prijavaService.deletePrijava(this.selektovanaPrijava.id).subscribe(
      () => {
        this.prijave$ = this.prijavaService.dajSvePrijave();
        this.selektovanaPrijava = new Prijava(null, null, null, null, null);
      },
      (error) => {
        console.log(error);
      }
    )
  }

  onIzmena(prijava: Prijava) {
    this.prijavaService.idPrijave = prijava.id;
    this.router.navigate(['/izmeni/' + prijava.id]);
  }

  onJasliceClick() {
```

```
    this.selektovanaVrsta = 1;
    this.prijave$ = this.filterPrijavePoVrsti(this.selektovanaVrsta);
  }

  onObdanisteClick() {
    this.selektovanaVrsta = 2;
    this.prijave$ = this.filterPrijavePoVrsti(this.selektovanaVrsta);
  }

  onSviClick() {
    this.selektovanaVrsta = 0;
    this.prijave$ = this.prijavaService.dajSvePrijave();
  }

  onStampajClick() {
    this.router.navigate(['/stampa/', this.selektovanaVrsta]);
  }

  filterPrijavePoVrsti(vrsta: number) {
    return this.prijavaService.dajPrijavuPoVrsti(vrsta);
  }
}
```

Listing 25. Prikaz koda komponente za tabelarni prikaz

6.4.4.9. Rutiranje

Sve prikazane komponente su unutar angular komponente rutirane, i te rute se definišu u datoteci app-routing.module.ts. One rute koje imaju osobinu expectedRole: 'admin' predstavljaju rute kojima samo admin može pristupiti.

```
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent},
  { path: 'prijava', component: PrijavaComponent },
  { path: 'tabela', component: TabelarniComponent, canActivate: [AuthGuard],
data: { expectedRole: 'admin'}},
  { path: 'izmeni/:id', component: IzmenaComponent, canActivate: [AuthGuard],
data: { expectedRole: 'admin'}},
  { path: 'stampa/:num', component: StampaComponent, canActivate: [AuthGuard],
data: { expectedRole: 'admin'}},
  { path: 'success-page', component: SuccessPageComponent},
  { path: 'login', component: LoginComponent }
];

@NgModule({
  imports: [CommonModule, RouterModule.forRoot(routes, { useHash: true })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listing 26. Rute komponenti

7. Zaključak

U ovom radu je izrađena veb aplikacija za evidenciju i prijavu dece za Predškolsku ustanovu koja pokriva sve osnovne CRUD operacije, i neke dodatne funkcionalnosti poput pretrage, filtriranja, sortiranja i štampe. Za izradu su se koristile moderne tehnologije za kreiranje brzih i fleksibilnih aplikacija poput Spring Boot i Angular razvojnih okruženja. Posebno se daje naglasak na višeslojnost veb aplikacije i na REST arhitekturu koja odvaja front-end od back-end dela, time obezbeđujući lako održavanje i izmenu koda kao i ponovnu upotrebljivost istog.

Aplikacija takođe otvara mnogo mogućnosti za unapređenje, što funkcionalnosti, što izgleda i dizajna. U narednoj iteraciji poboljšavanja softvera, planira se proširivanje liste osetljivih grupa koja bi se nalazila u bazi podataka kao i dve različite liste prijavljenih učenika, jedna za tek prijavljene, a druga za primljene učenike. Podrazumeva se i proširenje same forme za unos osnovnih informacija, u smislu dodavanja polja za lične podatke o roditeljima, zdravlju, posebnim potrebama itd.

8. Literatura

[1] https://www.flatironschool.com/blog/front-end-vs-back-end-development

[2] http://www.differencebetween.net/technology/difference-between-frontend-and-backend/

[3] https://www.lifewire.com/what-is-a-web-application-3486637

[4] https://blog.stackpath.com/web-application/

[5] https://surpass.com.ph/web-application.php

[6] https://en.yeeply.com/blog/web-app-development-website-accessible-mobile/

[7] https://dzone.com/articles/types-of-web-applications-from-a-static-web-page-t

[8]https://www.honeywebsolutions.com/blog/10-best-e-commerce-website-design-features/

[9]http://www.geekmantra.com/subsection.php?section=J2EE+Intro&subSection=J2EE+Overview&permalink=34

[10] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair: "Distributed Systems: Concepts and Design", Addison Wesley, 2012.

[11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: “Design Patterns”, Addison-Wesley, 1995.

[12] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[13] https://www.webprogramiranje.org/web-servisi-osnove/

[14] https://www.javatpoint.com/soap-web-services

[15] Moxa Inc, Using a RESTful API to Connect to Remote I/Os, 2017

[16] http://www.springboottutorial.com/hibernate-jpa-tutorial-with-spring-boot-starter-jpa

[17]https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/

[18] https://stackify.com/dependency-injection/

[19] https://martinfowler.com/articles/injection.html

[20] https://angular.io/

[21] https://www.sitepoint.com/angular-introduction/

[22] Hrćan Jan, Veb aplikacija za evidentiranje softverskih zahteva, diplomski rad, Tehnički fakultet „Mihajlo Pupin“ Zrenjanin, 2018.

[23] Predrag Pecev, Veb dizajn – materijal sa vežbi, Tehnički fakultet „Mihajlo Pupin“ Zrenjanin, 2019.

[24] https://getbootstrap.com/

[25] https://www.techopedia.com/definition/7755/intellij-idea

[26] http://www.nasedete.org/boravak_u_vrticu.php?page=programi_rada_sa_decom

[27]https://www.essentialeducation.my/preschool/Choosing-A-Preschool-Simple-Tips-For-Parents

[28] http://www.nsprv.org/zakon_o_predskolskom_vaspitanju_i_obrazovanju.pdf

https://www.vps.ns.ac.rs/Materijal/mat22111.pdf

[29] https://www.summitk12.org/school-eb45c0f4/apply-for-preschool-a125cb92

[30] http://www.predskolskazr.edu.rs/eUpis.php

[31]https://www.euprava.gov.rs/eusluge?service=servicesForTemplate&serviceTemplateId=3042

[32] https://www.vps.ns.ac.rs/Materijal/mat22111.pdf

http://www.geekmantra.com/subsection.php?section=J2EE+Intro&subSection=J2EE+Overview&permalink=34

[33] https://www.callicoder.com/spring-boot-rest-api-tutorial-with-mysql-jpa-hibernate/