

JavaScript biblioteka za rukovanje prikazom i ažuriranjem podataka

Šimunović, Ivan

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:144:440487>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-12-21**



Repository / Repozitorij:

[Repository of Bjelovar University of Applied Sciences - Institutional Repository](#)



VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

**JAVASCRIPT BIBLIOTEKA ZA RUKOVANJE
PRIKAZOM I AŽURIRANJEM PODATAKA**

Završni rad br. 05/RAČ/2021.

Ivan Šimunović

Bjelovar, 09. 2021.



Veleučilište u Bjelovaru
Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Šimunović Ivan** Datum: 22.07.2021. Matični broj: 001993
Kolegij: **WEB PROGRAMIRANJE 1** JMBAG: 0314019831
Naslov rada (tema): **JavaScript biblioteka za rukovanje prikazom i ažuriranjem podataka**
Područje: **Tehničke znanosti** Polje: **Računarstvo**
Grana: **Programsko inženjerstvo**
Mentor: **Tomislav Adamović, mag.ing.el.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. Krunoslav Husak, dipl. ing. rač., predsjednik
2. Tomislav Adamović, mag.ing.el., mentor
3. Ivan Sekovanić, mag.ing.inf.et comm.techn., član

2. ZADATAK ZAVRŠNOG RADA BROJ: 05/RAČ/2021

U radu je potrebno izraditi JavaScript biblioteku za povezivanje čelnog sloja sa servisima za upravljenje evidencijom poslovanja koristeći koncept Aplikacija u jednoj stranici. Potrebno je koristiti postojeću biblioteku DataTables.js za prikaz podataka u tablicama te izraditi ljusku koja omogućuje jednostavnije upravljanje elementima prikaza podataka. Osim navedene biblioteke potrebno je izraditi biblioteku za generiranje korisničkog sučelja za izmjenu i validaciju podataka.

Zadatak uručen: 22.07.2021.

Mentor: **Tomislav Adamović, mag.ing.el.**



Sadržaj

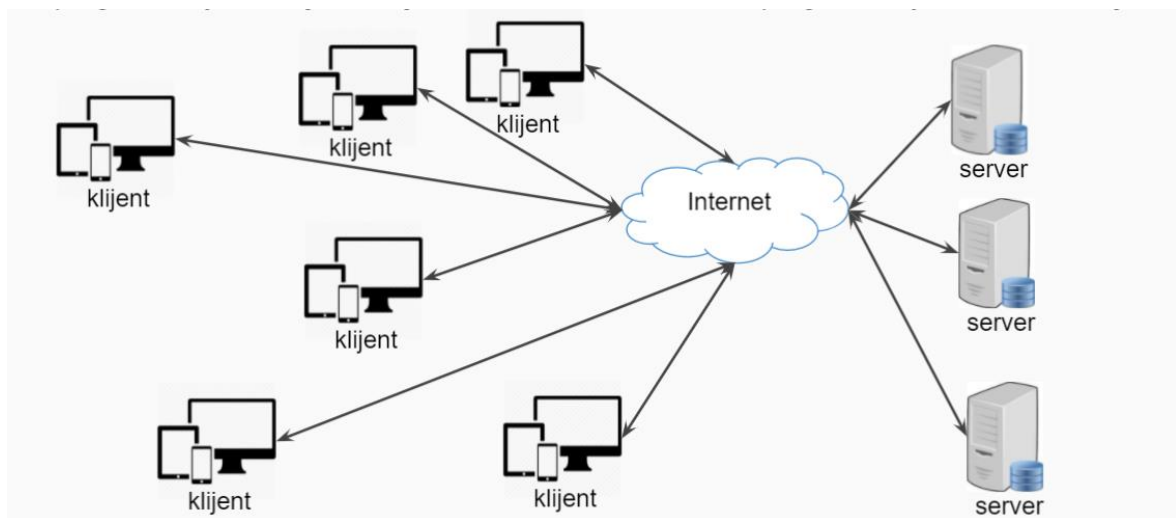
1. UVOD.....	1
2. ARHITEKTURA SUSTAVA I OBLIK PODATAKA.....	2
3. AJAX.....	4
4. JEDNOSTRANIČNE APLIKACIJE	5
5. DATATABLES.JS BIBLIOTEKA	6
5.1 Rad s podacima	6
5.2 Primjena CSS stilova na tablice	9
5.3 Dodatne funkcionalnosti tablica.....	9
5.4 Događaji na tablici.....	10
5.5 Dodatni elementi na tablici	11
6. CONFIG DATOTEKA	12
6.1 Definiranje različitih tipova podataka	14
6.2 Definiranje HTML tablice	16
7. COMMON DATOTEKA	17
8. VALIDACIJA PODATAKA.....	21
9. PROŠIRENJE FUNKCIONALNOSTI FORMI	25
10. KAKO NAPRAVITI NOVU TABLICU	28
11. ZAKLJUČAK.....	30
12. LITERATURA	31
13. OZNAKE I KRATICE	32
14. SAŽETAK	33
15. ABSTRACT.....	34

1. UVOD

Cilj ovog završnog rada je omogućiti korisnicima jednostavno i brzo razvijanje tablica i formi za različite sustave kojima je potreban jednostavan prikaz podataka. Izrada tablica i formi na klasičan način kroz HTML može biti dugotrajan i mukotrpan postupak, to se izrazito može primijetiti na sustavima koji moraju prikazivati velik broj tablica. Također, uz većinu tablica potrebna je i popratna forma kroz koju se korisniku omogućuje upravljanje podacima koji se odnose na tu tablicu. Forme se mogu sastojati od mnoštva različitih elemenata kao što su radio gumb, potvrdni okvir, više tipova polja za unos teksta, izbornik s vrijednostima i još mnogo drugih elemenata. Svaki element predstavlja jednu vrijednost koja se sprema u odgovarajući stupac u tablici. Zbog toga se na održavanje takvih sustava može izgubiti mnogo vremena. Ako nisu dobro projektirani i jednostavne promjene u tablici kao što je dodavanje jednog retka ili brisanje jednog retka može dovesti do velikih problema. Ime biblioteke koja se izrađuje je Wrapper.js kako bi je razlikovali od ostalih biblioteka koje se pojavljuju tijekom ovog rada. Wrapper.js biblioteka rješava sve probleme održavanja tablica vrlo jednostavno uz promjenu samo nekoliko redova koda. Način na koji Wrapper.js radi je sljedeći, pritiskom na određenu tipku na ekranu ili elementu izbornika moguć je prikaz određene tablice. Nakon što je tablica otvorena moguć je pregled svih podataka koji se nalaze unutar te tablice i moguće su osnovne CRUD operacije. Wrapper.js se sastoji od tri datoteke od kojih svaka ima svoju ulogu. Ovaj rad se sastoji od nekoliko glavnih poglavlja, drugo poglavlje ukratko objašnjava način na koji sustav radi i u kojem su obliku podaci koji se koriste. Treće poglavlje govori o AJAX komunikaciji koja se koristi u Wrapper.js biblioteci, a četvrto poglavlje o jednostraničnim aplikacijama. U petom poglavlju je opisana biblioteka koja se koristi za generiranje tablica pod nazivom DataTables.js. Šesto poglavlje govori o config.js datoteci koja sadrži podatke iz kojih se generiraju tablice i forme, te o različitim elementima koji se mogu prikazati na formi i različitim funkcionalnostima koje se mogu primijeniti na određeni stupac u tablici. Sedmo poglavlje prikazuje način na koji se integriraju config podaci i DataTables.js datoteka. Osmo poglavlje opisuje način na koji je provedena validacija nad podacima koji se šalju s forme. Deveto poglavlje opisuje način na koji se mogu proširiti funkcionalnosti formi. Deseto poglavlje prikazuje način na koji se može stvoriti nova tablica i forma. Na kraju se nalazi zaključak i u njemu su opisane ideje koje su nastale prilikom izrade Wrapper.js biblioteke.

2. ARHITEKTURA SUSTAVA I OBLIK PODATAKA

Arhitektura sustava je izvedena tako da više klijenata istovremeno može pristupiti sustavu preko interneta, a sustav se nalazi na udaljenom serveru. Svi podaci se šalju i dohvaćaju preko PHP servisa. Servisi omogućuju komunikaciju s bazom podataka kako bi dobili podatke koje želimo prikazati u tablicama i slanje podataka kako bi ih mogli pohraniti za daljnje korištenje. Na slici 2.1 je prikazan oblik na koji je provedena arhitektura sustava.



Slika 2.1: Arhitektura sustava

Podaci se šalju s Wrapper.js biblioteke na PHP servis gdje se mogu dodatno obraditi i pretvaraju se u JSON oblik koji se dalje prosljeđuje do baze podataka ako je sve u redu. Također, podaci s baze podataka se šalju u JSON obliku [1] i potrebno ih je pretvoriti u objekt kako bi JavaScript mogao raditi s podacima. Pretvaranje iz JSON oblika podataka u JavaScript objekt se može napraviti s `JSON.parse()` funkcijom, a obrnuto pretvaranje je moguće s `JSON.stringify()` funkcijom. JSON oblik podataka je vrlo dobar i jednostavan za korištenje zbog toga što su JSON objekti vrlo slični JavaScript objektima i pretvorba iz jednog oblika u drugi je brza i jednostavna. Na slici 2.2 je prikazan primjer način na koji je potrebno strukturirati podatke za rad s Wrapper.js-om.

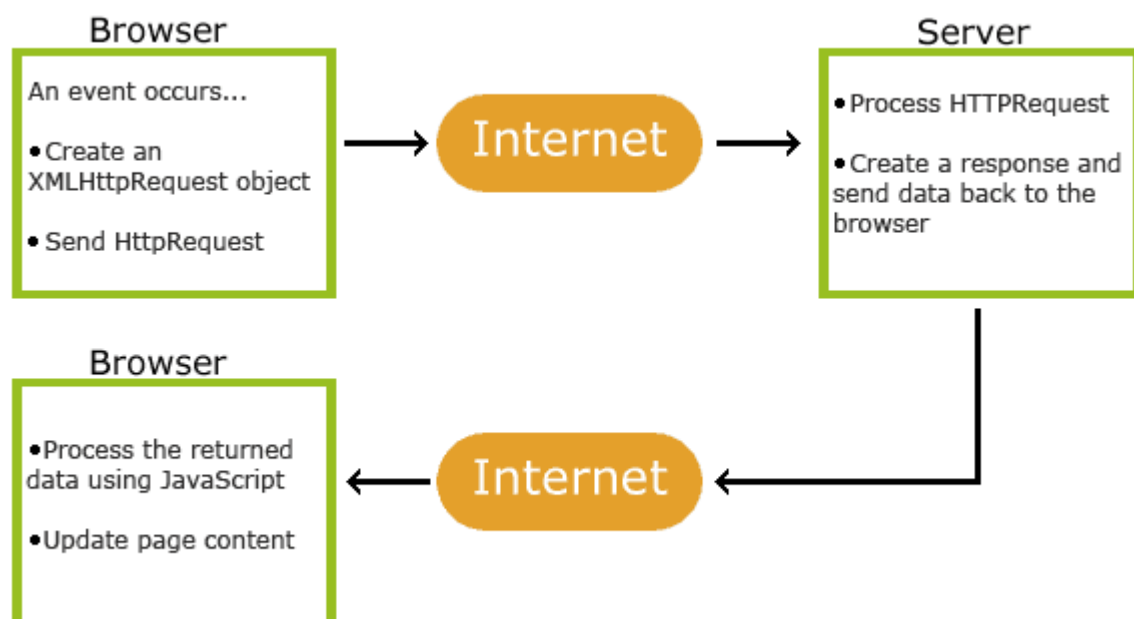
```
{
  "data": [
    {
      "ID": 1,
      "IDKORISNIKA": 2,
      "IME": "Pero",
      "PREZIME": "Perić",
      "ADRESA": "adresa",
      "STRUKA": "Viša",
      "RADNO_MJESTO": "Tvrtka",
      "OPIS": "neki",
      "SATNICA": 30
    }
  ]
}
```

Slika 2.2: JSON oblik podataka

Kao što se sa slike 2.2 vidi vrijednosti u JSON strukturi mogu biti jednostavni oblici podataka, ali i složeniji oblici podataka kao polja i objekti. Podaci su strukturirani tako da se s lijeve strane dvotočke nalazi ime svojstva, a s desne strane se nalazi vrijednost. Programski jezici koji u sebi imaju integriranu podršku za JSON, vrijednost iz JSON-a dohvaćaju preko naziva svojstva. Za korištenje ove biblioteke potrebno je imati izrađeni PHP servis za komunikaciju čelnog sloja sa bazom podataka. Također, potrebno je strukturirati podatke na način koji je opisan u kasnijim poglavljima ovog rada. Jezgra Wrapper.js biblioteke je DataTables.js biblioteka koja služi za stvaranje tablica, a sama Wrapper.js biblioteka sadrži dodatne funkcionalnosti koje proširuju funkcionalnost cijelog sustava.

3. AJAX

Značenje kratice AJAX je Asinkroni JavaScript i XML. AJAX nije posebna tehnologija nego se koriste već postojeće tehnologije zajedno poput HTML-a, CSS-a, JavaScripta, XML-a, DOM-a i drugih tehnologija. Kombinacijom svih tih tehnologija, web aplikacije se mogu izrađivati vrlo brzo i moguće je raditi promjene na korisničkom sučelju bez ponovnog učitavanja cijele stranice. Na taj način aplikacije su brže i bolje reagiraju na unose korisnika. Iako znak x u AJAX kratlici označava XML, u današnje vrijeme sve više se koristi JSON zbog jednostavnosti i dio je JavaScripta. Pomoću AJAX komunikacije moguće je poslati i preuzeti podatke u različitim formatima kao JSON, XML, HTML, ali moguće je prenositi i datoteke. Glavna karakteristika AJAX komunikacije je asinkroni način rada, što omogućuje komunikaciju sa serverom i promjene na aplikaciji bez ponovnog učitavanja aplikacije. Na slici 3.1 je prikazan način na koji AJAX komunikacija radi. Prije same komunikacije potreban je neki događaj na aplikaciji kao što je učitavanje aplikacije, pritisak gumba, unos teksta. Nakon toga JavaScript izrađuje XMLHttpRequest objekt. Taj objekt se koristi za komunikaciju sa serverom i dohvaćanje podataka. Kada je objekt izrađen on šalje zahtjev na server i server ga obrađuje. Nakon obrade server vraća odgovor aplikaciji i odgovor se obrađuje u JavaScriptu. Na kraju izvršava se neki događaj kao što su promjena podataka u tablici, promjena boje, otvaranje forme.



Slika 3.1: Način rada AJAX komunikacije

4. JEDNOSTRANIČNE APLIKACIJE

Jednostranična aplikacija je aplikacija kod koje je većina korisničkog sučelja cijelo vrijeme isto dok se mijenjaju samo određeni dijelovi. Neki od dijelova aplikacija koji ostaju cijelo vrijeme isti mogu biti izbornik, bočna traka s gumbovima, zaglavlje i podnožje. Aplikacija tijekom korisničkog rada mijenja svoje podatke i prikazuje samo ono što je potrebno za razliku od standardnih aplikacija gdje se svakim klikom cijela stranica učitava iznova. Kako bi ostvarili takav način rada koristi se AJAX komunikacija i na taj način aplikacije su puno brže i olakšava rad servera. Neki primjeri jednostraničnih aplikacija su: Gmail, Facebook, PayPal, Netflix. Postoji nekoliko nedostataka kod ovog tipa aplikacija. Neki od nedostataka su korištenje mnogo resursa web preglednika i problem prikaza pronalaska aplikacije prilikom pretraživanja putem Google-a ili ostalih pretražitelja. Problem pronalaska aplikacije se javlja zbog toga što aplikacija sadrži mali broj stranica s URL-ovima, zaglavlja i ključnim riječima što spušta aplikaciju na popisu prilikom pretraživanja. Također, jedan od problema koji se nalazi kod nekih jednostraničnih aplikacija je što ne postoji gumb unazad prilikom korištenja aplikacije. Na taj način kada bi korisnik pritisnuo gumb za vraćanje web preglednik će izbaciti korisnika iz aplikacije. Uz to postoje i sigurnosni problemi, koji dovode do skriptnih napada. Zbog korištenja XSS-a hakeri jednostavno mogu ubaciti skripte u aplikaciju.

5. DATATABLES.JS BIBLIOTEKA

DataTables.js [2] je besplatna JavaScript biblioteka čija je glavna svrha brzo generiranje tablica na web stranicama poput tablice prikazane na slici 5.1.



IME	PREZIME	ADRESA	STRUČNA SPREMA	NAZIV RADNOG MJESTA	OPIS RADNOG MJESTA	BRUTO SATNICA	ACTION
Pero	Perić	Adresa	Viša	Tvrtka	Opis	50.00	 
Ivo	Ivić	Adresa	Viša	Tvrtka	Opis	50.00	 

Slika 5.1: Tablica dobivena DataTables.js bibliotekom

Kako bi mogli koristiti DataTables.js biblioteku u vlastitom projektu potrebno je uključiti biblioteku jQuery inačice 1.7 ili novije i potrebno je uključiti dvije linije koda kako bi uključili potrebne stilove i Javascript datoteke za rad DataTables.js biblioteke. Kod koji je potrebno uključiti u HTML datoteku je prikazan na slici 5.2.

```
<link rel="stylesheet" href="https://cdn.datatables.net/1.10.22/css/jquery.dataTables.min.css">  
<script src="https://cdn.datatables.net/1.10.22/js/jquery.dataTables.min.js"></script>
```

Slika 5.2: Datoteke potrebne za rad DataTables.js biblioteke

Kod za uključivanje jQuery biblioteke je moguće pronaći na stranici [3]. Jedna od opcija koje pruža DataTables.js biblioteka za rad nad tablicama je paginacija koja omogućuje ograničenje prikaza određenog broja redaka u isto vrijeme. Svi podaci koji neće moći biti prikazani se mogu prikazati klikom na gumb stranice, *next* ili *previous* gumbove ispod tablice. Također, moguće je uključiti pretraživanje nad podacima u tablici kako bi lakše pronašli potrebne podatke. Dodatno je omogućen i poredak vrijednosti od najmanje prema najvećoj i obrnuto.

5.1 Rad s podacima

Podaci koji se prikazuju na tablici moraju uvijek biti strukturirani kao polje vrijednosti i svaki element u tom polju predstavlja jedan redak koji će se prikazati u tablici. Prema [4] elementi unutar tog polja mogu biti polja ili objekti. Ako se koriste polja kao podaci to omogućuje jednostavan rad DataTables.js biblioteci tako da prvo polje predstavlja prvi redak u tablici, drugo polje drugi redak i tako do kraja podataka. Na slici 5.1.1 je prikazan oblik podataka ako se koriste polja kao podaci za prikaz na tablici.

```

var data = [
  [
    "Tiger Nixon",
    "System Architect",
    "Edinburgh",
    "5421",
    "2011/04/25",
    "$3,120"
  ],
  [
    "Garrett Winters",
    "Director",
    "Edinburgh",
    "8422",
    "2011/07/25",
    "$5,300"
  ]
]

```

Slika 5.1.1: Polja unutar polja

Na slici 5.1.2 su prikazani podaci sa slike 5.1.1 unutar tablice.

Name	Position	Office	Extn.	Start date	Salary
Tiger Nixon	System Architect	Edinburgh	5421	2011/04/25	\$3,120
Garrett Winters	Director	Edinburgh	8422	2011/07/25	\$5,300

Slika 5.1.2: Prikaz podataka sa slike 5.1.1 unutar tablice

Kod za inicijalizaciju tablice se nalazi na slici 5.1.3, *example* je *id* prazne tablice na kojoj želimo prikazati podatke. Način na koji je potrebno definirati tablicu je prikazan na slici 5.1.4. Tablica sa slike 5.1.4 je definirana unutar varijable kako bi se kasnije na više mjesta mogla koristiti i može biti definirana unutar same HTML stranice.

```

$('#example').DataTable( {
  data: data
} );

```

Slika 5.1.3: Inicijalizacija tablice kod korištenja polja

```

var dataTablica = "<table id=\"example\" class=\"display\" style=\"width:100%\"><thead><tr>

```

Slika 5.1.4: Definiranje prazne tablice

Drugi način na koji podaci mogu biti strukturirani su objekti unutar polja. Ovaj način rada s podacima omogućuje vrlo jednostavan prikaz samo određenih podataka koji želimo.

Potreban je jedino naziv svojstva unutar objekta, a u slučaju polja vrlo važno je znati na kojoj poziciji unutar polja su ti podaci. Osim toga objekti mogu sadržavati i više podataka nego što je potrebno u prikazu, što može olakšati rad nad podacima. Na primjer primarni ključ koji se ne smije prikazati krajnjem korisniku. Na slici 5.1.4 su prikazani podaci gdje se koriste objekti unutar polja.

```
[
  {
    "name": "Tiger Nixon",
    "position": "System Architect",
    "salary": "$3,120",
    "start_date": "2011/04/25",
    "office": "Edinburgh",
    "extn": "5421"
  },
  {
    "name": "Garrett Winters",
    "position": "Director",
    "salary": "$5,300",
    "start_date": "2011/07/25",
    "office": "Edinburgh",
    "extn": "8422"
  }
]
```

Slika 5.1.4: Objekti unutar polja

Nedostatak korištenja objekata je što je potrebno točno specificirati DataTables.js biblioteci koje svojstvo objekta treba prikazati u kojem stupcu. Inicijalizacija tablice se provodi na drugačiji način u odnosu kada se koriste polja. Kod potreban za inicijalizaciju tablice se nalazi na slici 5.1.5.

```
$('#example').DataTable( {
  data: data,
  columns: [
    { data: 'name' },
    { data: 'position' },
    { data: 'salary' },
    { data: 'office' }
  ]
} );
```

Slika 5.1.5: Inicijalizacija tablice kod korištenja objekata

Na slici 5.1.5 se vidi novo svojstvo *columns* koje govori DataTables.js biblioteci na kojem mjestu će se nalaziti koji podaci u tablici. U slučaju sa slike *name* će biti prikazano na prvom stupcu, *position* na drugom stupcu, *salary* na trećem, a *office* na četvrtom stupcu.

Start_date i *extn* u ovom slučaju nije korišteno (nije potrebno prikazati sve podatke), ti podaci se mogu koristiti za različite provjere koje su nam potrebne na stranici ili na neki drugi način. Još jedna prednost kod korištenja objekata je što se jednostavno mogu preoblikovati tablice. Moguće je vrlo jednostavno dodati ili ukloniti stupac iz tablice, a u slučaju da se mora promijeniti poredak stupaca to se isto može napraviti brzo i jednostavno tako da se promijene pozicije objekata unutar svojstva *columns*.

5.2 Primjena CSS stilova na tablice

DataTables.js ima mogućnost uključivanja gotovo svake biblioteke za promjenu stilova HTML elemenata. Prema [5] već postoji nekoliko gotovih primjera klasa koje se mogu koristiti za mijenjanje izgleda tablica. Promjena izgleda se može jednostavno napraviti tako da se promijeni naziv klase varijable *dataTablica* unutar *common.js* datoteke, također različiti stilovi se mogu primijeniti i na element u kojem se nalazi tablica i njegov *id* je *container*.

Neke od gotovih klasa koje se mogu primijeniti nad vašim tablicama su:

- *cell-border*: dodaje obrub oko tablice,
- *compact*: uklanja nepotreban prazan prostor koji se nalazi unutar tablice,
- *hover*: kada se postavi pokazivač miša na redak boja se mijenja kako bi se isticao od ostatka tablice,
- *stripe*: svaki drugi redak u tablici obojan je drugačijom bojom od bijele.

Osim gotovih klasa moguće je uključiti i *Bootstrap* 3, 4 i 5. Kako bi to napravili potrebno je preuzeti odgovarajuće CSS datoteke sa stranice DataTables.js biblioteke i uključiti ih u projekt. Na stranici [6] je opisano način na koji je moguće uključiti različite *Bootstrap* inačice u vlastiti projekt.

5.3 Dodatne funkcionalnosti tablica

Osim prikaza podataka moguće je uključiti neke dodatne funkcionalnosti tablica kao što su filtriranje podataka, podjela tablice na stranice, pomicanje redaka i stupaca. Neke od funkcionalnosti koje se mogu upotrijebiti na gotovo svim tablicama prema [7] su:

- *autoWidth*,
- *lengthChange*,
- *ordering*,

- *paging*,
- *searching*,
- *colreorder*.

Sve ove nabrojane funkcionalnosti su automatski uključene osim *colreored*, moguće ih je isključiti ako želite napraviti vlastite funkcionalnosti nad tablicama. *AutoWidth* služi za pametno postavljanje širine stupaca unutar tablica kako ne bi gubili potreban prostor. Opcije *paging* i *lengthChange* služe za podjelu tablice po stranicama, *lengthChange* nije dostupan u slučaju da je *paging* funkcionalnost isključena. *LengthChange* omogućuje odabir koliko redova želimo prikazati po stranici. Slika 5.3.1 prikazuje izgled tablice u slučaju korištenja *paging* i *lengthChange* funkcionalnosti. Na slici element na kojem se odabire broj redaka u tablici ima mogućnost odabira 10, 25, 50 ili 100 redaka prikaza unutar tablice.

Show entries Search:

Name	Position	Office	Age	Start date	Salary
Airi Satou	Accountant	Tokyo	33	2008/11/28	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	\$86,000
Bradley Greer	Software Engineer	London	41	2012/10/13	\$132,000
Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	\$206,850
Brielle Williamson	Integration Specialist	New York	61	2012/12/02	\$372,000
Bruno Nash	Software Engineer	London	38	2011/05/03	\$163,500
Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	\$106,450
Cara Stevens	Sales Assistant	New York	46	2011/12/06	\$145,600
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	\$433,060

Showing 1 to 10 of 57 entries First Previous 2 3 4 5 6 Next Last

Slika 5.3.1: Tablica s uključenom funkcionalnosti *paging*

5.4 Događaji na tablici

Prema [8] DataTables.js šalje DOM događaje koje se može uhvatiti i ovisno o događaju može se izvršiti određeni dio koda. Događaje se može uhvatiti pomoću DataTables.js funkcije *on()* ili pomoću jQuery funkcije *on()*. Događaji koje šalje DataTables.js tablica rade na isti način kao i standardni DOM događaji i popis svih događaja koji postoje se mogu pronaći unutar DataTables.js dokumentacije. Ako se koristi jQuery funkcija *on()* potrebno je dodati *.dt* iza imena događaja unutar funkcije, dok

DataTables.js funkcija automatski dodaje umjesto vas. Na slici 5.4.1 je prikazano kako uključiti slušanje na određeni događaj putem DataTables.js funkcije *on()*, a na slici 5.4.2 je prikazano kako uključiti praćenje određenog događaja putem jQuery funkcije.

```
table.on( 'draw', function () {  
    alert( 'Table redrawn' );  
} );
```

Slika 5.4.1: Slušanje na događaj putem DataTables funkcije

```
$('#example').on( 'draw.dt', function () {  
    alert( 'Table redrawn' );  
} );
```

Slika 5.4.2: Slušanje na događaj putem jQuery funkcije

5.5 Dodatni elementi na tablici

Osim prikaza podataka moguće je dodati i različite HTML elemente unutar stupaca tablice kao što su različite tipke, ikonice, linkovi i ostali. Na slici 5.5.1 je prikazano kako dodati ikone unutar tablice.

```
columns.push({ "data": "ACTION" });  
columnDefs.push({  
    "targets": -1,  
    "data": null,  
    "className": 'center',  
    "defaultContent": "<a class = \"edit\"><i class='fas fa-edit fa-lg' style='color:green'></i></a>"  
});
```

Slika 5.5.1: Dodavanje HTML elemenata unutar tablice

U *columns* varijabli moramo stvoriti novi objekt, a vrijednost *ACTION* označava da se taj element ne stvara unutar forme. Kroz svojstvo *className* možemo primijeniti određene CSS klase, a u *defaultContent* postavljamo HTML kod za stvaranje željenog elementa, u ovom slučaju je to ikonica.

6. CONFIG DATOTEKA

Ova datoteka je glavni dio Wrapper.js biblioteke i koristi se za definiranje tablica i održavanje sustava. U njoj se nalaze dvije glavne funkcije pod nazivima *getColumnsConfig()* i *getTableConfig()*. U funkciji *getColumnsConfig()* se definiraju stupci svih tablica koje se prikazuju. Unutar te funkcije provjerava se vrijednost globalne varijable *tablica* kroz *switch* naredbu. *Switch* naredba se koristi u ovom slučaju kako bi dobili pregledniji kod, kod bi bio dosta nepregledan zbog velikog broja *if* uvjeta. U varijablu *tablica* je potrebno spremi naziv tablice prilikom obrade događaja pritiska gumba za prikaz tablice. Svaka tablica je definirana putem dva polja *columns* i *columnDefs*, oba polja su popunjena objektima i svaki objekt predstavlja jedan stupac u tablici. Unutar polja *columns* definirano je zaglavlje tablice, a unutar polja *columnDefs* definiramo kakav je tip podatak koji ćemo prikazati u tom stupcu i da li želimo da taj stupac bude skriven ili ne. Vrijednost svojstva *data* mora odgovarati imenu svojstva podataka koje želimo prikazati u tom stupcu, a u slučaju da želimo različiti naziv stupca može se koristiti svojstvo *title* u kojemu se definira novi naziv stupca. Na slikama 6.1 i 6.2 je prikazan način kako se definira tablica unutar *getColumnsConfig()* funkcije.

```
case 'SKLADISTA_V':
  columns = [
    { "data": "ID" },
    { "data": "IDKORISNIKA" },
    { "title": "NAZIV SKLADIŠTA", "data": "NAZIV" },
    { "title": "ADRESA SKLADIŠTA", "data": "ADRESA" },
  ];
```

Slika 6.1: Definiranje tablice unutar *config* datoteke

```

columnDefs = [
  {
    "targets": [0],
    "visible": false,
    "tip": 'number'
  },
  {
    "targets": [1],
    "visible": false,
    "tip": 'number'
  },
  {
    "targets": [2],
    "visible": true,
    "orderable": false,
    "tip": 'text'
  },
  {
    "targets": [3],
    "visible": true,
    "orderable": false,
    "tip": 'text',
  }
];
break;

```

Slika 6.2: Definiranje tablice unutar config datoteke

Također, vrlo je važno da tip podatka bude točan jer je bitan kako bi se mogao generirati odgovarajući element forme za unos i promjenu podataka. Postoji nekoliko tipova podataka koji se mogu koristiti, a to su:

- *number*,
- *text*,
- *hidden*,
- *listOfValues*,
- *dateTime*,
- *cbox*,
- *radioButton*,
- *readOnlyNumber*,
- *tableHiddenNumber*.

Svaki od tih tipova generira svoj element unutar forme i nebitan je kod prikaza podataka unutar tablice. Tip *number* se koristi u slučaju kada se koriste brojevi u tom stupcu i generira polje za unos brojeva, tip *text* se koristi kada se koristi običan tekst i generira standardno polje za unos. Tip *hidden* se koristi kada je potrebno podatak prikazati u tablici, ali ne želimo da ih korisnik može mijenjati, kao na primjer datum i vrijeme koje se

generira prilikom unosa podatka u bazu ili slično. Tip *listOfValues* se koristi kada je potreban izbornik s listom vrijednosti koju odabire korisnik prilikom unosa, tip *dateTime* generira standardan element za odabir datuma i vremena. Tip *cbox* je običan potvrdni okvir koji korisnik označava prilikom unosa podataka, a može ostati i prazan. Tip *radioButton* generira dva gumba i samo jedan može biti pritisnut u isto vrijeme. *ReadOnlyNumber* je tip koji generira polje za unos teksta, ali nije moguće unositi tekst unutar njega. Jedna od mogućih uporaba ovakvog tipa je kada su potrebna virtualna polja za različite proračune koje želimo samo prikazati, a nije potreban unos u bazu podataka. Zadnji tip je *tableHiddenNumber* i on se koristi u slučaju kada nije potreban prikaz tog podatka na tablici ili kako bi smanjili broj redaka tablice, ali generira odgovarajući element unutar forme za unos i mijenjanje tog podatka.

6.1 Definiranje različitih tipova podataka

U ovom potpoglavlju bit će objašnjeno način na koji se definiraju različiti tipovi podataka unutar *getColumnConfig()* funkcije. Tipovi poput: *number*, *text*, *hidden*, *dateTime*, *cbox*, *readOnlyNumber* i *tableHiddenNumber* mogu se koristiti na isti način kao na slici 6.2, potrebno je samo postaviti vrijednost od svojstva *tip* na željenu vrijednost. Ako se koristi podatak tipa *text* moguće je postaviti svojstvo *maxLength* koje će ograničiti podatak na određenu duljinu prilikom unosa unutar forme. Ako se koristi podatak tipa *number* moguće je postaviti svojstvo *decimal* s vrijednošću *true* kako bi vrijednost unutar tablice uvijek bila prikazana i zaokružena na dvije decimale. Tipovi *listOfValues* i *radioButton* se koriste na drugačiji način od osnovnih tipova i potrebno je definirati neka dodatna svojstva kako bi se mogli koristiti. Kod tipa *radioButton* potrebno je definirati svojstva *label1* i *label2* s odgovarajućim vrijednostima koje se prikazuju ispred tipaka unutar forme. Vrijednost prvog gumba je 1 dok je vrijednost drugog gumba 0. Primjer definiranja tipa *radioButton* je prikazan na slici 6.1.1.

```
{
  "targets": [8],
  "visible": true,
  "orderable": false,
  "tip": 'radioButton',
  "label1": 'Ulazna pošta',
  "label2": 'Izlazna pošta'
},
```

Slika 6.1.1: Definiranje tipa *radioButton*

Kada smo ispravno definirali *radioButton* tip generira se rezultat prikazan na slici 6.1.2.

Ulazna pošta	<input type="radio"/>
Izlazna pošta	<input type="radio"/>

Slika 6.1.2: Rezultat tipa *radioButton*

Na sličan način se može koristiti i tip *listOfValues*, potrebno je samo definirati nekoliko dodatnih svojstava. Primjer definiranja tipa *listOfValues* je prikazan na slici 6.1.3.

```
{
  "targets": [3],
  "visible": true,
  "orderable": false,
  "tip": 'listOfValues',
  "lov_table": 'LOV_PARTNERI_V',
  "insert_column": "IDPARTNERA",
  "label": 'PARTNER'
},
```

Slika 6.1.3: Definiranje tipa *listOfValues*

Kao što se na slici 6.1.3 vidi potrebno je definirati *lov_table*, *insert_column* i *label*. Svojstvo *lov_table* govori Wrapper.js biblioteci iz koje tablice na bazi se dohvaća lista vrijednosti, *insert_column* označava u koji stupac se sprema vrijednost prilikom unosa forme i svojstvo *label* označava tekst koji se nalazi ispred elementa na formi. Rezultat ispravnog definiranja je prikazan na slici 6.1.4.

PARTNER	<input type="text"/>
Stalni	<input type="text"/>

partner234▼

partner▼

Slika 6.1.4: Rezultat tipa *listOfValues*

Podatke koje želimo prikazati kao listu vrijednosti potrebno je formatirati kao na slici 6.1.5.

```

{
  "data": [
    {
      "ID": 1,
      "PARTNER": "partner234"
    },
    {
      "ID": 22,
      "PARTNER": "partner"
    }
  ]
}

```

Slika 6.1.5: Oblik podataka koje želimo prikazati kao lista vrijednosti

6.2 Definiranje HTML tablice

Svi podaci potrebni za stvaranje tablice se moraju definirati u funkciji `getTableConfig()`. Kao i u funkciji `getColumnsConfig()` podaci se dohvaćaju preko `switch` naredbe ovisno o imenu tablice. Za stvaranje tablice potrebni su podaci, nazivi stupaca, svojstva stupaca i određene funkcionalnosti koje se mogu uključiti uz tablicu, a opisane su u potpoglavlju 5.3. Neke funkcionalnosti su automatski omogućene, dok se neke moraju uključiti, a u slučaju da nije potrebna određena funkcionalnost koja je automatski uključena moguće ju je isključiti tako da se postavi na vrijednost `false`. Primjer način na koji je potrebno definirati tablicu prikazan je na slici 6.2.1. Nakon što je tablica definirana ti podaci se dalje predaju `DataTables.js` funkciji koja stvara tablicu od tih podataka.

```

case 'RADNICI_V':
  postavke = {
    data: podaci,
    "searching": true,
    "paging": true,
    "lengthChange": true,
    "info": false,
    "autoWidth": false,
    "columns": columns,
    "columnDefs": columnDefs
  }
  break;

```

Slika 6.2.1: Definiranje tablice unutar funkcije `getTableConfig()`

7. COMMON DATOTEKA

Common datoteka obavlja većinski dio logike koji je potreban za rad Wrapper.js biblioteke, ona služi kao poveziivač DataTables.js biblioteke s *config* datotekom u kojoj se nalaze svi podaci o tablicama. U ovoj datoteci se nalaze sve funkcije potrebne za prikaz tablica, dohvaćanje podataka, generiranje odgovarajuće forme uz tablicu, ispunjavanje forme dobivenim podacima i spremanje i brisanje podataka. Prije samog korištenja Wrapper.js biblioteke potrebno je podesiti podatke za povezivanje na bazu podataka. Za komunikaciju s bazom korišten je PHP servis, a podaci se prosljeđuju do servisa pomoću AJAX komunikacije. Na samom početku datoteke nalaze se globalne varijable *url* i *projekt* koje se koriste za spajanje na bazu i unutar funkcija za dohvaćanje, spremanje i brisanje podataka nalazi se varijabla *procedura* koju također treba podesiti. Moguće je promijeniti sve podatke koji su potrebni za komunikaciju s bazom, a moguće je koristiti i neki drugi način komunikacije. Kako bi mogli prikazati tablice potreban nam je `<div>` element na HTML stranici, a vrijednost *id-a* bi trebala odgovarati vrijednosti container. Moguće je koristiti i *breadcrumbs*, također potreban je `<div>` element i vrijednost *id-a* mora biti *breadcrumbs*. Uz generiranje tablica, Wrapper.js generira i odgovarajuće gumbe za unos, promjenu i brisanje podataka. Za stvaranje tablica koristi se funkcija *showTable()*. Parametri koje je potrebno predati ovoj funkciji su ime tablice kao prvi parametar i *Breadcrumbs*, ukoliko ih želite koristiti, kao drugi parametar. Prvo što se radi u toj funkciji je preuzimanje potrebnih podataka iz *config* datoteke kako bi mogli prikazati tablicu. U ovu svrhu se koriste funkcije *getColumnConfig()*, a *getTableConfig()* funkciju pozivamo nakon dohvaćanja podataka koje želimo prikazati iz baze ili neke datoteke. Nakon što dohvatimo potrebne podatke koji opisuju stupce u tablici, potrebno je pozvati funkciju *getPodaci()*, a kao parametar se koristi ime tablice koju želimo prikazati. Vrlo je bitno da se nazivi tablica podudaraju s nazivima tablica na bazi jer se u tom obliku šalju podaci na bazu, ukoliko se ne podudaraju potrebno je provesti dodatnu logiku za provjeru imena tablica ili na bazi ili prije poziva prema bazi za dohvaćanje podataka. Sada kada imamo sve potrebne podatke poziva se funkcija od DataTables.js biblioteke kako bi stvorili tablicu i poslije toga možemo dodati neke od CSS klase kako bi promijenili izgled tablice. Na kraju funkcije su definirani događaji koji se izvršavaju prilikom klika na ikone za promjenu podataka i brisanje podataka. Na slici 7.1 je prikazano način na koji je izvršen taj dio koda i što se događa kada se klikne na ikonu za promjenu podataka, a na slici 7.2 je prikazano

što se izvršava prilikom klika na ikonu za brisanje podataka. Funkcije *showRecord()* i *delRecord()* sa slika 7.1 i 7.2 se koriste za prikaz i brisanje podataka.

```
$('#example tbody').on('click', '.edit', function () {  
    var data = table.row($(this).parents('tr')).data();  
    showRecord(data.ID, data.IDKORISNIKA);  
});
```

Slika 7.1: Događaj prilikom klika na ikonu za promjenu podataka

```
$('#example tbody').on('click', '.delete', function () {  
    var data = table.row($(this).parents('tr')).data();  
    delRecord(data.ID, data.IDKORISNIKA);  
});
```

Slika 7.2: Događaj prilikom klika na ikonu za brisanje podataka

Sljedeća funkcija koja se nalazi u common datoteci je funkcija pod nazivom *getPodaci()*. Ona služi jedino za dohvaćanje podataka iz baze podataka, dohvaća cijelu tablicu, sprema ih u polje i prosljeđuje za daljnje korištenje. Svaki element unutar polja odgovara jednom retku u tablici. Komunikacija je izvedena putem AJAX-a, a moguće je podesiti komunikacijske parametre kako bi odgovarali strukturi vašeg projekta. Funkcija *insertForma()* služi za generiranje svih formi koje su potrebne za upravljanje podataka, a to su forme za unos i forme za promjenu podataka. Svaka forma koja se generira ovim putem odgovara konfiguraciji tablice koja je specificirana u *config* datoteci. Glavna logika ove funkcije se izvršava u *each* funkciji jQuery biblioteke, a kao parametar se koristi globalna varijabla *columns*. U toj funkciji kod svakog elementa unutar polja *columnDefs* se provjerava vrijednost atributa *tip* i generira odgovarajući element forme. Nakon toga dolazi funkcija *showRecord()*. Kao parametri se koriste *ID* kako bi dohvatili samo podatke odgovarajućeg retka, a ne cijelu tablicu. Prvo se dohvaćaju podaci iz baze podataka, isto je moguće podesiti komunikacijske parametre kao i u funkciji *getPodaci()*. Ukoliko su podaci uspješno preuzeti poziva se funkcija *insertForma()* koja je prije bila objašnjena i popunjava se forma s postojećim podacima kako bi ih korisnici mogli mijenjati. Popunjavanje elemenata forme s vrijednostima se izvršava kroz *each* funkciju jQuery biblioteke i ponovno se provjerava atribut *tip* unutar *columnDefs* polja. Svaki tip popunjava vrijednosti na različite načine i zbog toga se koriste *if-else* uvjeti. Na slici 7.3 prikazan je način na koji se popunjavaju vrijednosti elemenata.

```

$.each(jsonBody.data, function (k, v) {
    Object.keys(v).forEach(function (key) {
        for (i = 0; i < columns.length; i++) {
            if (columns[i].data == key) {
                //-----postavljanje vrijednosti na elemente forme-----
                if (columnDefs[i].tip == "cbox") {
                    setCheckBox(columns[i].data, v[key]);
                }
                else if (columnDefs[i].tip == "radioButton") {
                    setRadioButton(columns[i].data, v[key]);
                }
                else if (columnDefs[i].tip == "dateTime") {
                    $('#' + columns[i].data).val(v[key].replace(" ", "T"));
                }
                else if (columnDefs[i].tip == "listOfValues") {
                    selectValue = v[key];
                }
                else if (columnDefs[i].decimal == true) {
                    $('#' + columns[i].data).val(number_to_decimal(v[key], 2));
                }
                else {
                    $('#' + columns[i].data).val(v[key]);
                }
                break;
            }
        }
    });
});

```

Slika 7.3: Popunjavanje elemenata forme vrijednostima

Funkcije *setCheckBox()* i *setRadioButton()* će biti kasnije opisane, a sada će biti objašnjeni ostali *if* uvjeti. Ukoliko atribut *tip* odgovara vrijednosti *dateTime*, element *datetime-local* se popunjava s preuzetim podacima. Potrebno je napraviti *replace* naredbu nad vrijednošću kako bi uklonili oznaku koju generira ovaj element. U slučaju da vrijednost atributa *tip* odgovara vrijednosti *listOfValues* tada vrijednost spremamo u globalnu varijablu koju poslije koristimo u funkciji *nazivDropDown()*. U slučaju da je broj decimalnog oblika koristi se funkcija *number_to_decimal()* s prvim parametrom vrijednošću i drugim parametrom brojem decimala koliko želimo da broj ima. Važno je naglasiti da ta funkcija vraća vrijednost u obliku teksta, a ne broja, jer se prateće nule gube ukoliko je vrijednost tipa broj. Na primjer, ukoliko želimo da broj bude prikazan s dvije decimale, a korisnik unese broj 2,50 zadnja nula se gubi prilikom prikaza vrijednosti i zbog toga vrijednost mora biti pretvorena u tekst. Zadnji uvjet se ispunjava u slučaju kada je tip tekst ili broj i on samo postavlja vrijednost na element, nije potrebno nikakvo dodatno formatiranje vrijednosti. Nakon funkcije za popunjavanje forme, dolaze funkcije koje služe za spremanje i brisanje podataka. Spremanje podataka se izvršava kada se pritisne gumb s *id-om* *spremiRecord*, poziva se *control()* funkcija u kojoj se provodi validacija forme, a kao parametar se predaje ime tablice. U slučaju da validacija prođe bez problema na bazu podataka se šalju podaci s forme i spremaju ukoliko je sve u redu. Brisanje podataka se

može napraviti tako da se klikne ikona kante za smeće u retku koji se želi obrisati, prikazuje se obavijest na koju treba pritisnuti potvrdi kako bi se izbrisali podaci. Izvedeno je na taj način kako korisnici ne bi slučajno izbrisali podatke. Funkcije za postavljanje vrijednosti na *checkbox* element i na *radio* gumb rade tako da provjeravaju da li je vrijednost parametra koji je predan funkciji jednak vrijednosti 1. Ukoliko je vrijednost 1, *checkbox* element će biti označen, a kod *radio* gumbova prvi gumb će biti označen. Ukoliko je vrijednost 0, *checkbox* element neće biti označen i kod radio gumbova drugi gumb će biti označen. Sljedeća funkcija za postavljanje vrijednosti na elemente forme je *nazivDropDown()*. Unutar te funkcije preuzimaju se podaci iz baze podataka koje želimo prikazati kao listu vrijednosti, postavljaju se kao opcije unutar elementa za odabir i ukoliko se mijenjaju podaci postavlja se odgovarajuća vrijednost kao početna. Način na koji se postavljaju opcije liste vrijednosti je prikazan na slici 7.4.

```
$.each(jsonBody.data, function (k, v) {  
    output += '<option id="' + v.ID + '" value="' + v.NAZIV + '">' + v.NAZIV + '</option>'  
});
```

Slika 7.4: Postavljanje opcija u element za odabir vrijednosti

Podaci bi trebali biti strukturirani na način da se sastoje od svojstva *ID* i *NAZIV*, vrijednost svojstva *ID* će biti postavljena kao *id* opcije za odabir kako bi lakše kasnije unijeli vrijednosti u bazu podataka, a vrijednost svojstva *NAZIV* se koristi kao tekst koji će biti ponuđen korisniku na odabir. Bilo bi dobro na ovakav način strukturirati podatke na bazi podataka, a ako to nije moguće može se modificirati ovaj dio koda kako bi odgovarao vašem projektu. To su bile najvažnije funkcije common datoteke i način na koji one funkcioniraju i kako se mogu koristiti.

8. VALIDACIJA PODATAKA

Validacija podataka se izvršava unutar *control()* funkcije u *control* datoteci. Za dohvaćanje svih elemenata forme koriste se dvije *for* petlje i različiti *if* uvjeti kako bi se utvrdilo koji element je trenutno u petlji. Prva *for* petlja služi za dohvaćanje svih redaka forme, svaki redak u formi se sastoji od *<th>* HTML elementa i u njemu se nalazi tekst koji opisuje element koji se nalazi u *<td>* HTML elementu koji sadrži element za unos ili odabir podataka. U drugoj *for* petlji se dohvaćaju svi elementi forme koji sadrže podatke koje je korisnik unio. Koriste se različiti *if* uvjeti kako bi se utvrdilo koji element je trenutno u petlji i ovisno o elementu izvršava se različiti dio koda. Dalje se spremaju podaci u objekt tako da ime svojstva objekta odgovara *id-u* elementa forme, a vrijednosti svojstva odgovara vrijednosti trenutnog elementa forme u petlji. Prvo se radi provjera da li je element lista s podacima za odabir, provjera se radi na način da uspoređujemo svojstvo *tagName* s vrijednošću *SELECT*. Ako je rezultat *if* uvjeta *true*, vrijednost varijable *id* postavljamo na vrijednost koja odgovara svojstvu *insert_column* koja je definirana u *config* datoteci i označava ime stupca u tablici u bazi podataka u koji ćemo spremati ovu vrijednost. Ukoliko vrijednost nije odabrana, na korisničkom sučelju se prikazuje poruka koja obavještava korisnika da je potrebno ispuniti to polje. Ako je odabrana vrijednost preuzimamo vrijednost koju sadrži trenutno označena opcija u listi vrijednosti i sprema se u varijablu *value*. Slika 8.1 sadrži programski kod gdje je prikazano kako se izvršava provjera i spremaju vrijednosti u varijable.

```
if (element.tagName === 'SELECT') {
  id = columnDefs[r].insert_column;
  label = t.rows[r].firstChild.textContent;
  if (element.options[element.selectedIndex] == null || element.options[element.selectedIndex] == "") {
    Swal.fire('Molimo ispunite polje: ', label);
    return;
  }
  else {
    value = element.options[element.selectedIndex].id;
    label = t.rows[r].firstChild.textContent;
  }
}
```

Slika 8.1: Validacija i priprema podataka u slučaju kada je element lista vrijednosti

Sljedeće provjere koje se provode u ovoj funkciji su provjere kojoj vrijednosti odgovara vrijednost svojstva *type*, provjera se radi za vrijednosti *checkbox*, *radio* i *datetime-local*. Kada vrijednost svojstva *type* odgovara vrijednosti *checkbox* tada se izvršava dio programskog koda u kojem spremamo *id* elementa u varijablu *id* i provjeravamo da li je

element na formi korisnik označio ili ne. Ako je element označen vrijednost koju spremamo u varijablu `value` je 1, a ako element nije označen tada spremamo vrijednost 0. Element na formi može biti označen, a može ostati i prazan i zbog toga nije potrebna dodatna provjera i poruka za korisnika ako vrijednost nije postavljena. Na slici 8.2 je prikazan programski kod koji se izvršava ukoliko je tip elementa `checkbox`.

```
else if (element.type === 'checkbox') {
  id = element.id;
  value = 0;
  checked = element.checked;
  label = t.rows[r].firstChild.textContent;
  if (checked == true) {
    value = 1;
  }
  else {
    value = 0;
  }
}
```

Slika 8.2: Priprema podataka za spremanje ukoliko je tip elementa `checkbox`

Kada vrijednost svojstva `type` odgovara vrijednosti `radio`, u varijablu `id` spremamo `id` koji odgovara `radio` gumbu. `Radio` gumbi su uvijek u paru ili ih je više od dva i zbog toga ne možemo imati isti `id` na svim gumbima, taj problem je riješen na način da je redni broj `radio` gumba nadodan na vrijednost `id-a`. Funkcija `slice()` se koristi kako bi se uklonio zadnji znak iz stringa i dobije čisti `id` bez rednog broja. Kako bi dobili vrijednost trenutno označenog gumba putem jQuery-a se izvršava provjera koji je označen i preuzima se njegova vrijednost i sprema u varijablu `value`. Također ako niti jedan gumb nije označen korisniku se prikazuje poruka da je potrebno odabrati gumb. Na slici 8.3 prikazan je dio programskog koda koji se izvršava i slučaju kada je element na formi `radio` gumb.

```
else if (element.type === 'radio') {
  id = element.id.slice(0, -1);
  value = $('input[name=' + element.name + ']:checked').val();
  label = t.rows[r].firstChild.textContent;
  if (value == "" || value == null) {
    Swal.fire('Molimo ispunite polje: ', label);
    return;
  }
}
```

Slika 8.3: Priprema podataka za spremanje ukoliko je tip elementa `radio` gumb

Sljedeću vrstu elemenata koju je potrebno provjeriti je element za odabir datuma i vremena. Ukoliko je uvjet ispunjen *id* elementa se sprema u varijablu *id*, a nakon toga se preuzima vrijednost i sprema u varijablu *value*. Prije spremanja vrijednosti potrebno je izvršiti *replace()* funkciju nad tekстом kako bi dobili čiste podatke bez znaka T koji generira ovaj HTML element. Potrebno je naglasiti da se vrijednost datuma i vremena šalje u obliku stringa, a ne u obliku *timestampa*. Ako se želi koristiti *timestamp* potrebno je napraviti dodatno pretvaranje u taj oblik. Ponovno kao i kod prethodnih elemenata prikazuje se poruka korisniku ako nije postavljena vrijednost na taj element. Na slici 8.4 je prikazan prethodno objašnjen dio koda vezan uz element za odabir datuma i vremena.

```
else if (element.type === 'datetime-local') {
  id = element.id;
  value = element.value.replace("T", " ");
  label = t.rows[r].firstChild.textContent;
  if (value == "" || value == null) {
    Swal.fire('Molimo ispunite polje: ', label);
    return;
  }
}
```

Slika 8.4: Priprema podataka za spremanje ukoliko je element za odabir datuma i vremena

Na kraju ostaje *else* uvjet, u taj uvjet dolaze sva polja za unos teksta i brojeva, a radi na sličan način kao i prethodni uvjeti. Prvo se preuzimaju *id* i vrijednost elementa i pohranjuje u odgovarajuće varijable. Nakon toga provjerava se da li je element ispunjen, ako nije prikazuje se korisniku poruka da je potrebno ispuniti to polje. Zatim provjeravamo da li je vrijednost elementa broj pomoću regularnog izraza i funkcije *test()*, ako je broj, vrijednost je potrebno pretvoriti u tip broj zbog toga što smo decimalni tip brojeva pretvarali u tekst kako bi se decimalni brojevi uvijek mogli prikazati s jednakim brojem decimala. Na kraju funkcije ako vrijednost nije broj provjeravamo da li je postavljeno ograničenje za maksimalni broj znakova kod unosa teksta. Ako je postavljeno svojstvo *maxLength* u *config* datoteci provjerava se da li je duljina teksta unesena u polje za unos veća od specificirane maksimalne duljine. Taj dio programskog koda je prikazan na slici 8.5.

```

else {
  id = element.id;
  value = element.value;
  label = t.rows[r].firstChild.textContent;

  if (value == "" || value == null) {
    Swal.fire('Molimo ispunite polje: ', label);
    return;
  }
  else{
    var isNumber = /^d+\.\d+$/i.test(value)
    if(isNumber == true){
      value = Number(value);
    }
    else{
      var length = value.length;
      if(columnDefs[r].maxLength < length){
        Swal.fire('Molimo smanjite broj znakova na polju: ' + label + ' na ' + columnDefs[r].maxLength + " znakova");
      }
    }
  }
}
}
}







```

Slika 8.5: Priprema podataka za spremanje ukoliko je tip elementa polje za unos

Na kraju svake iteracije druge *for* petlje vrijednosti koje su dobivene u trenutnoj iteraciji se spremaju u objekt. Ako su svi elementi na formi ispunjeni, funkcija vraća objekt koji je spreman za slanje na bazu podataka i spremanje u tablicu.


9. PROŠIRENJE FUNKCIONALNOSTI FORMI



Svaka tablica generirana putem Wrapper.js biblioteke ima odgovarajuću formu s kojom se mogu unositi i mijenjati podaci. Forme se sastoje od običnih HTML elemenata u koje korisnici unose podatke i svaki element predstavlja jedan stupac u tablici. Na slici 9.1 je prikazan primjer tablice, a na slici 9.2 odgovarajuća forma za unos podataka.

RBR	DATUM	ISPRAVA	ACTION
1	2021-07-09 15:38	123	 
3	2021-07-10 15:39	12345	 
2	2021-07-13 13:13	123	 

Previous 1 Next

Slika 9.1: Primjer tablice

RBR	<input type="text" value="1"/>
DATUM	<input type="text" value="07/09/2021 03:38 PM"/> 
ISPRAVA	<input type="text" value="123"/>

Spremi  Odustani 

Slika 9.2: Primjer forme

Uz svaku tablicu generiraju se jednostavne forme, a u slučaju da je potrebna dodatna funkcionalnost potrebno je modificirati funkciju za generiranje forme i dodati potrebne događaje na elemente. Jedan od slučajeva gdje je potrebno proširiti funkcionalnost formi je kada je potrebno automatski mijenjati podatke u formi kako korisnik mijenja vrijednost sa *SELECT* elementa. Svaki puta kada korisnik promjeni vrijednost tog elementa popunjavaju se vrijednosti elemenata koji su postavljeni u funkciji koju je potrebno dodati. Jedan od načina na koji se ta funkcionalnost može ugraditi u sustav je da se postavi funkcija u kojoj je potrebno napraviti željenu funkcionalnost kao vrijednost *onChange* događaja nad elementom u *insertForma()* funkciji u common datoteci. Primjer programskog koda je prikazan na slici 9.3.

```

else if (columnDefs[k].tip == 'listOfValues') {
  if (tablica == "KALKULACIJA_V") {
    output += '<tr><th scope="col">' + columnDefs[k].label + '</th><td><select id="' + v.data + '" onChange="promjenaArtikla()"></td></tr>';
  }
  else {
    output += '<tr><th scope="col">' + columnDefs[k].label + '</th><td><select id="' + v.data + '"></td></tr>';
  }
}
nazivDropDown(v.data, columnDefs[k].lov_table);
}

```

Slika 9.3: Proširenje funkcionalnosti SELECT HTML elementa

Kao što se na slici vidi dodan je jedan *if* uvjet koji će se dogoditi samo kada se generira forma za tablicu *KALKULACIJA_V*, a u drugim slučajevima ništa se ne mijenja. Na *onChange* događaj dodana je funkcija *promjenaArtikla()* i u njoj je potrebno definirati što želimo da se dogodi kada se ispuni *onChange* događaj. Na slici 9.4 je prikazan programski kod funkcije *promjenaArtikla()*.

```

function promjenaArtikla() {
  document.getElementById("NABAVNA_CIJENA").value = $("#ARTIKL").find(':selected').attr('data-cijena');
  kalkulacija();
}

```

Slika 9.4: Funkcija koja se poziva prilikom onChange događaja

Kada se pozove funkcija mijenjamo vrijednost elementa čija vrijednost id-a odgovara vrijednosti *NABAVNA_CIJENA* i postavljamo je na drugu vrijednost koja se preuzima s elementa koji je trenutno odabran. Funkcija *kalkulacija()* se koristi za različite proračune koju je također potrebno provesti u ovom slučaju i njena funkcionalnost bit će objašnjena malo kasnije. Na slici 9.5 su prikazani elementi na kojima je provedena prethodno objašnjena funkcionalnost.

ARTIKL	<input type="text" value="test"/> <input type="text" value="novi"/> <input type="text" value="test"/> <input type="text" value="asdf"/>
KOLICINA	<input type="text"/>
NABAVNA CIJENA	<input type="text" value="200"/>
NABAVNA VRIJEDNOST	<input type="text" value="2400.00"/>

Slika 9.5: Forma nad kojom je provedena funkcionalnost za promjenu vrijednosti

Svaki puta kada korisnik promjeni vrijednost elementa artikl vrijednost elementa nabavna vrijednost se također mijenja na vrijednost koja odgovara nabavnoj vrijednosti trenutno odabranog artikla. Sljedeći slučaj gdje se može proširiti funkcionalnost formi je kod automatskih izračuna, na primjer izračun PDV-a, ukupne plaće, ukupna cijena po količini i

još mnogo drugih slučajeva. Takvi izračuni mogu ubrzati rad korisnika jer ne moraju gubiti vrijeme na računanje već ih sustav sam generira i prikazuje. Kako bi dobili sličnu funkcionalnost potrebno je dodati *oninput* događaj na element u funkciji *insertForma()* kao i u prošlom primjeru. Na slici 9.6 je prikazan primjer kako se dodaje funkcija koja će se izvršiti kada korisnik unese neku vrijednost na element. Na isti način se ta funkcionalnost može napraviti i na drugim tipovima nije potrebno da baš bude *tablehiddenNumber* samo je potrebno naznačiti točnu tablicu kako se funkcije ne bi pozivale na svim tablicama i dovele do problema u radu sustava.

```

if (columnDefs[k].tip == 'tablehiddenNumber') {
  if (v.title != null) {
    if (tablica == "KALKULACIJA_V") {
      output += '<tr><th scope="col">' + v.title + '</th><td><input type="number" id="' + v.data + '" oninput="kalkulacija()"></td></tr>';
    }
    else if (tablica == "PLACE_V") {
      output += '<tr><th scope="col">' + v.title + '</th><td><input type="number" id="' + v.data + '" oninput="placa()"></td></tr>';
    }
    else {
      output += '<tr><th scope="col">' + v.title + '</th><td><input type="number" id="' + v.data + '"></td></tr>';
    }
  }
  else {
    if (tablica == "KALKULACIJA_V") {
      output += '<tr><th scope="col">' + v.data + '</th><td><input type="number" id="' + v.data + '" oninput="kalkulacija()"></td></tr>';
    }
    else if (tablica == "PLACE_V") {
      output += '<tr><th scope="col">' + v.data + '</th><td><input type="number" id="' + v.data + '" oninput="placa()"></td></tr>';
    }
    else {
      output += '<tr><th scope="col">' + v.data + '</th><td><input type="number" id="' + v.data + '"></td></tr>';
    }
  }
}
}

```

Slika 9.6: Proširenje funkcionalnosti prilikom unosa korisnika

U funkcijama *kalkulacija()* i *placa()* rade se razni proračuni vrijednosti koje su nam potrebne za te tablice, a vrijednosti je moguće postaviti na isti način kao što je prikazano na slici 9.4. Potrebno je dohvatiti sve elemente na kojima želimo raditi proračune i postaviti vrijednost na izračunatu vrijednost. Takva funkcionalnost je vrlo dobra za ugraditi u vlastiti sustav jer znatno smanjuje vrijeme koje je potrebno korisniku da izračuna sve potrebne vrijednosti i unese ih u sustav.

10. KAKO NAPRAVITI NOVU TABLICU

U ovom poglavlju će biti objašnjen način na koji sa Wrapper.js bibliotekom korisnik može brzo i jednostavno stvoriti nove tablice i uključiti ih u svoj projekt. Prije početka rada potrebno je uključiti skripte u projekt za rad DataTables.js biblioteke i common, config i control datoteke za ostale funkcionalnosti koje su potrebne za rad sustava. Nakon toga potrebno je napraviti `<div>` HTML element sa *id* vrijednošću *container*. U tom elementu će se prikazivati sve forme koje se generiraju, a ako je potrebno moguće je podesiti *id* vrijednost prema vašim potrebama. To je sve što je potrebno napraviti u vašem HTML kodu. Sada imamo sve potrebno za izradu tablica. Nove tablice se definiraju unutar config datoteke koja je prethodno dodana u projekt. U config datoteci se nalazi *getColumnsConfig()* funkcija i u njoj se definira kako će tablica izgledati. Primjer kako se definira jednostavna tablica je prikazan na slikama 6.1 i 6.2. U prvoj varijabli se definiraju nazivi stupaca. Vrlo je bitno da su podaci koje želimo prikazati u tablici strukturirani kao na slici 2.2 i da imena svojstava unutar objekta koji se nalazi u polju *data* odgovaraju vrijednostima svojstva *data* unutar *columns* varijable. Te vrijednosti će biti prikazane kao natpisi stupaca i podaci se prikazuju tim redom kako su definirani unutar *columns* varijable. U slučaju da je potrebno koristiti drugačiji natpis od vrijednosti koja je postavljena u *data* svojstvo, moguće je koristiti svojstvo *title* i vrijednost koja je postavljena na to svojstvo će uvijek biti prikazana. Koliko objekata se nalazi unutar *columns* varijable toliko će stupaca biti unutar tablice. To je sve što je potrebno napraviti u *columns* varijabli, a sljedeće je potrebno postaviti vrijednosti *columnDefs* varijable. Unutar te varijable se postavlja da li će stupac biti vidljiv na tablici. Moguće je postaviti CSS klase na stupac i potrebno je postaviti vrijednost svojstva *tip*. Nije potrebno da su svi stupci uvijek prikazani, moguće je koristiti skriveni stupac koji će sadržavati primarni ključ. Bez obzira da li je stupac s primarnim ključem prikazan ili ne on je potrebna u svim tablicama kako bi mogli izvršiti brisanje i mijenjanje podataka nad točno tim retkom u tablici. Također u svakom objektu je potrebno postaviti vrijednost svojstva, *tip* koji se koristi pri stvaranju forme preko koje je moguće upravljanje nad podacima. U potpoglavlju 6.1 je objašnjeno način na koji se definira određeni *tip* elementa unutar forme. Preostalo je definirati koje funkcionalnosti će imati tablica unutar funkcije *getTableConfig()*. Većina funkcionalnosti koje se mogu uključiti su objašnjena unutar potpoglavlja 5.3. Kako se to radi je prikazano na slici 6.2.1, potrebno je napraviti *case* slučaj unutar *switch* naredbe prema imenu tablice i unutar njega definirati objekt postavke. Kao vrijednost svojstva *data*

postavljamo podatke koje želimo prikazati, a u svojstva *columns* i *columnDefs* varijable *columns* i *columnDefs* koje su definirane u prethodnom koraku. To je sve što je potrebno za stvaranje tablica i formi, sada je samo potrebno pozvati funkciju *showTable()* za stvaranje tablice. Primjer kako pozvati funkciju je prikazan na slici 10.1.

```
$(document).on("click", "#skladista", function () {  
    tablica = 'SKLADISTA_V';  
    breadcrumbs = "Skladista";  
    showTable(tablica, breadcrumbs);  
});
```

Slika 10.1: Prikaz tablice

Ako je sve dobro definirano tablica će se prikazati unutar *container* elementa s gumbom za unos novih podataka. Sva komunikacija s bazom se odvija preko PHP servisa i potrebno je podesiti komunikacijske parametre unutar funkcija *getPodaci()*, *showRecord()*, *delRecord()*, *nazivDropDown()*, *refresh()* i globalnih varijabla *url* i *projekt* unutar *common* datoteke te podesiti komunikacijske parametre unutar *control()* funkcije u *control* datoteci. Ukoliko je sve dobro napravljeno cijeli sustav bi trebao biti funkcionalan i spreman za rad. Preostaje napraviti neke dodatne funkcionalnosti po želji kao što je opisano u poglavlju 9 i primjena raznih CSS stilova kako bi sustav bolje izgledao.

11. ZAKLJUČAK

Svaki dan raste broj novih tvrtki i poduzeća i njihovi vlasnici gotovo uvijek žele vlastitu internetsku stranicu za promoviranje proizvoda, praćenje poslovanja ili za neke druge aktivnosti. Moguće je da poslodavci nemaju neku osobu koja će im napraviti gotov sustav, a da bude brzo napravljen i uz to jednostavan za održavanje. Uz korištenje ove biblioteke korisnik koji radi na sustavu ili ga održava ne treba puno znanja o HTML-u i JavaScriptu kako bi izradio tablice spremne za korištenje. Uz ovu biblioteku moguće je vrlo brzo razviti cijeli sustav tablica koje se mogu koristiti na mnogo načina. Također nije potrebno raditi novu HTML stranicu jer su tablice vrlo jednostavne za uključiti u projekt. Jedna od prednosti korištenja Wrapper.js biblioteke je što se brzo mogu raditi promjene na tablicama. Na primjer, moguće je zamijeniti poredak redaka ili promijeniti natpis u zaglavlju tablice promjenom jednog redka programskog koda. Cijeli sustav ima mnogo funkcionalnosti nad tablicama koje su automatski uključene nakon stvaranja tablice i nije ih potrebno posebno izrađivati. Neke od funkcionalnosti su: sortiranje, pretraživanje podataka, paginacija i još mnogo drugih. Uz osnovne funkcionalnosti moguće je dodatno proširiti sustav s raznim događajima koji se pozivaju kako korisnik unosi podatke kako bi se olakšao njihov rad. Međutim, kako bi se provele dodatne funkcionalnosti potrebno je neko osnovno znanje o HTML-u i Javascriptu. Prilikom izrade tablice automatski se generiraju i odgovarajuće forme za unos podataka što dodatno olakšava rad osobama koje razvijaju sustav. Uz sve te funkcionalnosti sustav je vrlo brz za korištenje i novi korisnik se vrlo lako može snaći u programskom kodu.

12. LITERATURA

Navesti popis literature prema pravilima za navođenje literature koja se nalaze u Uputama za izradu završnog rada.

Primjeri:

- [1] JSON.org. Uvod u JSON [Online]. Dostupno na: <https://www.json.org/json-en.html>. (19.8.2021.)
- [2] Službena stranica DataTables.js biblioteke. Dodajte napredne interakcijske kontrole na vaše tablice na besplatan i lagan način [Online]. 2007. Dostupno na: <https://datatables.net/>. (19.8.2021.)
- [3] Službena stranica jQuery biblioteke. jQuery CDN – Najnovije stabilne verzije [Online]. Dostupno na: <https://code.jquery.com/>. (8.9.2021.)
- [4] Dokumentacija DataTables.js biblioteke. Podaci [Online]. 2007. Dostupno na: <https://datatables.net/manual/data/>. (19.8.2021.)
- [5] Dokumentacija DataTables.js biblioteke. Stilsko oblikovanje [Online]. 2007. Dostupno na: <https://datatables.net/manual/styling/classes>. (19.8.2021.)
- [6] Dokumentacija DataTables.js biblioteke. Stilsko oblikovanje [Online]. 2007. Dostupno na: <https://datatables.net/manual/styling/>. (19.8.2021.)
- [7] Dokumentacija DataTables.js biblioteke. Funkcionalnosti [Online]. 2007. Dostupno na: <https://datatables.net/manual/options>. (19.8.2021.)
- [8] Dokumentacija DataTables.js biblioteke. Funkcionalnosti [Online]. 2007. Dostupno na: <https://datatables.net/manual/events>. (19.8.2021.)

13. OZNAKE I KRATICE

HTML – HyperText Markup Language (Prezentacijski jezik za izradu web stranica)

CRUD – Create, Read, Update, Delete (Napravi, Pročitaj, Ažuriraj, Izbriši)

PHP – Hypertext Preprocessor

JSON – JavaScript Object Notation (JavaScript objektno obilježavanje)

DOM – Document Object Model (Objektni model dokumenta)

AJAX – Asynchronous JavaScript And XML (Asinkroni JavaScript i XML)

XML – Extensible Markup Language

14. SAŽETAK

JavaScript biblioteka za rukovanje prikazom i ažuriranjem podataka:

Cilj ovog rada je izrada biblioteke koja služi kao poveznica čelnog sloja sa servisima za upravljanje evidencijom poslovanja. Glavne funkcionalnosti ove biblioteke su prikaz tablica, automatska izrada formi uz tablice i CRUD upravljanje nad podacima. Tablice se izrađuju putem besplatne JavaScript biblioteke DataTables.js. U radu su opisane funkcionalnosti DataTables.js biblioteke i kako se može koristiti. Nakon toga objašnjen je način na koji se koristi izrađena biblioteka i što je sve moguće napraviti.

Ključne riječi: JavaScript, tablice, forme, DataTables.js.

15. ABSTRACT

JavaScript library for handling of display and data update:

The aim of this work is to develop library that connects front end with services for management of business records. Main functionalities of this library are table display, automatic generation of forms for tables and CRUD operations on data. Tables are generated with free JavaScript library DataTables.js. In this work main point that are described are functionalities of DataTables.js library and how to use it. After that it's explained how to use finished library and what can you do with it.

Keywords: JavaScript, tables, forms, DataTables.js.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, <u>20.09.2021.</u>	IVAN ŠIMUNOVIĆ	Ivan Šimunović

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

IVAN ŠIMUNOVIĆ

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 20.09. 2021.

Ivan Šimunović

potpis studenta/ice