JAVA WEB ALAT ZA IZRADU STATIČKE HTML STRANICE

Vukelić, Iva

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

JAVA WEB ALAT ZA IZRADU STATIČKE HTML STRANICE

Iva Vukelić

Zagreb, veljača 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 18.2.2020.

Predgovor

Zahvaljujem se mentoru prof. Aleksandru Radovanu što mi je pružio priliku da napišem ovaj rad i upoznam se s novim tehnologijama. Također se zahvaljujem svojim roditeljima, svom dečku i prijateljima koji su mi pružali podršku tokom pisanja ovog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Cilj ovog završnog rada je stvaranje alternative za platforme za izradu web stranica kao što su Wix i Squarespace gdje će korisnik biti u mogućnosti da izveze konačni rezultat u obliku HTML i CSS datoteka što nije bilo moguće s već postojećim platformama za stvaranje web stranica. Također će biti moguće da korisnik određuje raspored elemenata web stranice kroz crtež koji može biti nacrtan ručno ili kroz neki program sličan MS Paint. Prvi korak u stvaranju web stranice je stvaranje slike i njeno učitavanje. Sliku se zatim procesira kroz *backend* pisan u programskom jeziku Java uz korištenje biblioteke OpenCV. Drugi korak će biti prikaz generiranih HTML i CSS datoteka uz mogućnost da korisnik dodatno podesi raspored i veličinu elemenata.

Ključne riječi: React.js, OpenCV, Java, procesiranje slike, HTML, CSS

Main goal of this final thesis is to create an alternative solution for building web sites similar to Wix and Squarespace which would allow user to export the final result as HTML and CSS files which wasn't possible in already existing website bulder platforms. It's also going to be possible for user to arrange the layout through drawing which can be drawn manually pr by using software like MS Paint. First step in creating a web page is creating a picture and uploading it. Afterwards, the picture is being processed through backend wiritten in Java combined with OpenCV library. Second step is going to be preview of generated HTML and CSS file with option that allows user to additionally rearrange the layout.

Keywords: React.js, OpenCV, Java, image proccessing, HTML, CSS

Sadržaj

1. Uvod

Servisi poput Squarespace ili Wix korisnicima omogućuju jednostavnu izradu web stranica uz pomoć *drag and drop* tehnike. No ograničeni su što se tiče izvoza web stranice u HTML i CSS datoteke. To znači da korisnik, koji ima znanja u programiranju, svoju stranicu ne može izvesti i dodatno uređivati u vlastitom okruženju.

Cilj ovoga rada je napraviti programsko rješenje koje će korisniku omogućiti izradu osnovne web stranice koju je moguće izvesti. Korisnik će moći poslati sliku iz koje će se generirati raspored elemenata na web stranici koje je zatim moguće dodatno urediti.

Ovo programsko rješenje leži između postojećih rješenja i programerskih alata za izradu web stranica.

Početak završnog rada govori o postojećim alatima za izradu web stranica te njihovoj usporedbi. Zatim se prolazi kroz tehnologije koje će biti korištene u ovome radu. Iza toga slijedi opis programskog rješenja gdje se zasebno opisuju *frontend* i *backend* dio.

2. Pregled postojećih rješenja

Alati za izradu web stranica su online platforme koje omogućavaju korisniku da napravi vlastitu web stranicu bez pisanja koda.

2.1. Wix

Izašao na tržište u 2006., Wix je danas najpopularniji servis za izradu web stranica principu povuci-pusti (engl. *drag and drop*) tehnike koji kroz samo nekoliko minuta omogućuje jednostavnu izradu web stranica bez potrebe za pisanjem koda. Zbog toga je prikladan za početnike i ljude koji nemaju programerskog znanja. [1]

Wix je idealno rješenje i za izradu osobnih portfolija i za web stranica za kafiće i restorane zbog svoje jednostavnosti, skalabilnosti i stotina raznih predloška.

Postoje dva alata za izradu web stranica na Wix servisu – klasična platforma na principu predloška – Wix Editor i platforma na principu umjetne inteligencije Wix ADI. Za one koji ipak znaju nešto kodirati, postoji i Wix Code gdje korisnik dobiva pristup HTML datoteci predloška koju može još više prilagođavati.

2.1.1. Wix Editor

Wix editor koristi *drag and drop* tehniku uz koju korisnik sam određuje gdje će se koji element nalaziti na stranici. Korisnik u kombinaciji s *drag and drop* tehnikom može koristiti i već predefinirane predloške koje jednostavno može uređivati te dodavati vlastita svojstva i sadržaj.

Ti predlošci su zapravo već prethodno stvorene web stranice spremne za upotrebu na kojima korisnik dodaje ili uklanja sadržaj. Ukoliko želi urediti neki tekst, jednostavno upiše željeni tekst preko postojećeg teksta, a promjena veličine se promijeni preko klizača (engl. *Slider*).

[1] https://www.websitebuilderexpert.com/website-builders/wix/wix-review/ - link na web stranicu koji opisuje načine korištenja Wix servisa, njegove prednosti nedostatke

2.2. Squarespace

Alternativa za Wix je Squarespace, još jedan servis za izradu vlastite web stranice prvenstveno namijenjena kreativcima i korisnicima kojima je vizualni prikaz važan. Kao i za Wix, za korištenje Squarespace servisa nije potrebno nikakvo prethodno znanje u pisanju koda i u roku od sat vremena se može izraditi čitava web stranica. Sve što korisnik mora napraviti je izraditi korisnički račun, odabrati predložak i urediti ga po želji. [2]

Squarespace je također temeljen na *drag and drop* principu, no za razliku od Wix-a, korisnik ne može bilo što staviti bilogdje zbog toga što se Squarespace bazira na sekcijama. Usprkos tome korisnik svejedno može premještati određene elemente, kompletno mijenjati raspored slika, brisati i dodavati sekcije, upravljati s više stranica odjednom itd. Sve navedeno je moguće bez pisanja koda.

Jedna od značajki Squarespace-a je to što je on *What You See Is What You Get* u što u prijevodu znači da ono što korisnik vidi na ekranu, to će biti konačni rezultat jer kod nekih drugih servisa kao npr. WordPress, alat za uređivanje ne prikaže odmah finalni izgled stranice. Dakle, Squarespace svaku promjenu prikaže u stvarnom vremenu. Uz malo znanja u pisanju koda, korisnik može stvoriti i vlastite fontove, boje i pozadine.

Ukoliko korisnik želi promijeniti izgled stranice, može to napraviti čak i nakon što je stranica puštena u produkciju, a istovremeno može pregledavati i uspoređivati više dizajna istovremeno. Svaka promjena se automatski zabilježi na produkciji te nije potrebno ništa naknadno mijenjati. Također svi Squarespace predlošci su sami po sebi prilagodljivi za manje rezolucije.

2.3. Usporedba

Oba servisa su trenutno najbolji izbori ukoliko korisnik bez puno muke želi napraviti web stranicu. Iako oboje imaju mnoge zajedničke značajke, postoji niz razlika. Dok Squarespace omogućava naknadnu promjenu izgleda web stranice, Wix, nakon što korisnik jednom postavi stranicu online, ne može se više išta promijeniti. Squarespace ima otprilike oko 60

[2] https://www.websitebuilderexpert.com/website-builders/squarespace/squarespace-review/ - link na web stranicu koji opisuje načine korištenja Squarespace servisa, njegove prednosti nedostatke

predložaka vodeći se po izreci „*quality over quantity*”, a Wix nudi više od 500 različitih predložaka.

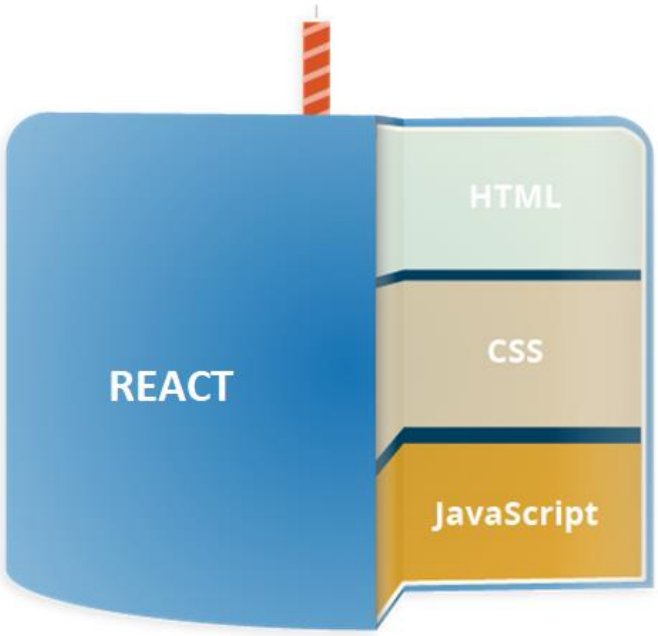
Squarespace je savršen izbor za fotografe i dizajnere zbog naglaska na vizualni dizajn, no to znači da zahtijeva slike s visokom rezolucijom. Ukoliko korisnik želi stranicu s dosta tekstualnog sadržaja i nema dovoljno kvalitetne slike, stranica neće izgledati dobro.

S druge strane, Squarespace nije toliko jednostavan za upotrebu kao što je Wix i potrebno je neko vrijeme da se korisnik privikne na njega i prouči sve opcije. Prikladniji je za korisnike koji imaju barem nekog iskustva ili dosta strpljenja. Ukoliko se korisnik Squarespace-a odluči za kodiranje i prilagodbu detalja po vlastitom kodu, moglo bi doći do neželjenih promjena jer Squarespace dozvoljava prvenstveno vizualne promjene. Razlog tome je to što se Squarespace bazira na vlastitim JavaScript i CSS podacima te njihovo manipuliranje nije preporučeno.

Što se tiče mogućnosti izvoza web stranica, moguće je dobiti podatke jedino u XML obliku koje se kasnije može uređivati u WordPress-u. CSS i JavaScript se ne izvoze.

3. Korištene tehnologije

Za *frontend* dio završnog rada koristi će se programski jezik JavaScript u kombinaciji s bibliotekom React, dok za *backend* će biti korišten programski jezik Java i biblioteka za prepoznavanje oblika na slici OpenCV. Slika 3.1. predstavlja međusobnu povezanost *frontend* tehnologija koje su međusobno povezane React bibliotekom kao nekakva torta.



Slika 3.1. Iskorištene *frontend* tehnologije
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

3.1. JavaScript

JavaScript je programski jezik za skriptiranje za klijentsku stranu prilagođen *ECMA Script* specifikacijama. Omogućuje korisniku implementaciju kompleksnih stvari na web stranicu kao što su dinamičko ažuriranje sadržaja, animiranje slike, multimedija itd.

Razvijen od strane Brendana Eicha, originalno se zvao LiveScript no u posljednji trenutak preimenovan je u JavaScript zbog marketinških razloga.

Iako su izašli u isto vrijeme i nose sličan naziv, Java i JavaScript su dva potpuno različita jezika, iako dijele neke zajedničke karakteristike poput sintakse te oboje potječu iz C i C++.[3]

Uz HTML i CSS, JavaScript je jedna od temeljnih tehnologija korištenih na webu. Podržana je u skoro svim web preglednicima i korištena na većini web stranica.

JavaScript je *lightweight* interpretirani programski jezik. Kod je u obliku teksta koji je čitljiv programeru. Nije potrebno pretvarati kod u neki drugi oblik prije nego ga preglednik pokrene, kao što je to slučaj kod kompajliranog koda.

JavaScript se u pregledniku pokreće od strane *browser enginea* za JavaScript, tek nakon što su se HTML i CSS spojili u web stranicu. To osigurava da je struktura i izgled stranice na svom mjestu u vrijeme pokretanja JavaScript skripte.

Application Programming Interfaces (API) omogućavaju dodatne funkcionalnosti u JavaScript kodu. API su već prethodno napravljeni setovi izvršnih naredbi za izradu koda koji omogućavaju razvojnom inženjeru da implementira programske funkcije koje bi inače bilo teško ili nemoguće za implementirati.

Postoje dvije vrste API-ja – API preglednika te *3rd party* API. [4]

Jedan od najvažnijih API preglednika je DOM API koji omogućava manipulaciju HTML-om i CSS-om i dinamičko dodavanje promjena na stranicu.

3.1.1. DOM

*Document Object Mo*del (DOM) je programsko sučelje za HTML i XML dokumente. DOM dokument predstavlja kroz čvorove i objekte. Sve je to prikazano kroz strukturu logičkog stabla gdje svaka grana završava s čvorom, a svaki čvor sadrži objekte. Kroz DOM metode program pristupa stablu i uz pomoć njih može provoditi promjene na dokumentu. Također, čvorovi mogu na sebi sadržavati rukovatelje događajima (engl. *event handler*) tako da čim se okine event, izvrši se *event handler*. [5]

[3] https://quirksmode.org/js/intro.html - link na web stranicu u kojem je opisana povijest JavaScript jezika i usporedba s Javom
[4] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript - link na stranicu od Mozilla-e na kojoj se nalazi dokumentacija za JavaScript jezik uz razna objašnjenja
[5] https://en.wikipedia.org/wiki/Document_Object_Model - link na stranicu Wikipedije na kojoj je objašnjen DOM

Sadržaj stranice je pohranjen u DOM-u i može mu se pristupiti i biti manipuliran samo preko JavaScript-a.

3.2. React

React je jedna od najpopularnijih i najkorištenijih JavaScript biblioteka na principu otvorenog koda. Koristi se za izradu korisničkih sučelja za *single page* aplikacije, kao i za mobilne aplikacije. Razvijen je od strane Facebook-ovog programskog inženjera Jordana Walkea te je React prvi put primijenjen na Facebook stranici s vijestima 2011., a zatim i na Instagram-u 2012.

React omogućuje programskim inženjerima stvaranje velikih web stranica na kojima se mogu mijenjati podaci bez ponovnog učitavanja stranice. Glavne odlike React-a su brzina, skalabilnost i jednostavnost.

Njegovo najbitnije svojstvo je mogućnost pružanja ponovo iskoristivih komponenti koje razvojnim inženjerima dozvoljavaju ponovnu upotrebu prethodno napravljenih komponenti u nekoj drugoj aplikaciji zadržavajući istu funkcionalnost.

React je veoma intuitivan za rad i pruža interaktivnost izgledu korisničkog sučelja i dozvoljava brzi i kvalitetni razvoj aplikacije što rezultira u štednji vremena i za klijenta i razvojnog inženjera. [6]

Umjesto običnog JavaScript koda, React koristi JSX. JSX, skraćeno od JavaScript XML, je kombinacija JS i HTML sintakse koja automatski pojednostavnjuje cijeli proces pisanja koda tako što omogućava pisanje HTML oznaka unutar JavaScript datoteke. Pruža način strukturiranja ispisivanja (engl. *render*) komponente koristeći sintaksu sličnu HTML-u. Kod pisan u JSX obliku zahtjeva konverziju korištenjem alata kao što je, recimo, Babel, da bi web preglednik mogao razumjeti taj kod. Taj proces se odradi tokom obrađivanja (engl. *build*) prije nego li se aplikacija stavi u pogon (engl. *deploy*).

Ukoliko razvojni inženjer koristi JSX da bi manipulirao DOM-om, React će stvoriti strukturu podataka unutar memorije zvanu *Virtual Document Object Model*. To je kopija DOM-a web stranice koju React koristi da bi promijenio podatke, a zatim i ažurirao u pregledniku. React

[6] https://www.c-sharpcorner.com/article/what-and-why-reactjs/ - link na web stranicu koja objašnjava kvalitete React biblioteke

skenira virtualni DOM da vidi koje je zapravo promjene korisnik napravio te zatim selektivno ažurira samo odabranu sekciju DOM-a.[7]

[7] https://www.javatpoint.com/react-features - link ne web stranicu sa popisom značajki za React biblioteku

3.3. Java

Java je objektno orijentirani programski jezik. Glavna namjera Java programskog jezika je bila da nakon što programeri jednom napišu kod, da se taj kod može pokrenuti bilo gdje. To znači da se kompajlirani kod može pokretati na svim platformama koje podržavaju Javu bez potrebe za ponovnim kompajliranjem.

Java programski kod se kompajlira u *intermediate representation* tzv. *Java bytecode*. *Java bytecode* izvršava Java Virtual Machine (JVM). Krajnji korisnici koriste Java Runtime Environment (JRE) za pokretanje Java aplikacija. [8]

3.4. Servlet

Java Servlet je komponenta koja nasljeđuje mogućnosti koje ima server. Uglavnom se koriste za implementaciju web kontejnera za pružanje usluge poslužitelja (engl. *hosting*) web aplikacija na web poslužiteljima te samim time može smatrati kao servlet web API na strani poslužitelja (engl. *server-side*).

U ovom radu Servlet se koristi za implementaciju poslužiteljskog API-ja za procesiranje slike.

3.5. OpenCV

OpenCV skraćeno od *open source computer vision*, je biblioteka funkcija namijenjenih prvenstveno za *computer vision* u stvarnom vremenu. Razvijena od strane Intel-a 2000. godine.[9]

OpenCV ima modularnu strukturu što znači da se sastoji od nekoliko statičnih biblioteka. U ovom radu biti će korišteni moduli za osnovnu funkcionalnost – *core* i modul za procesiranje slike – *imgproc*.

[8] https://en.wikipedia.org/wiki/Java_(programming_language) – link na stranicu Wikipedije na kojoj je objašnjen Java programski jezik

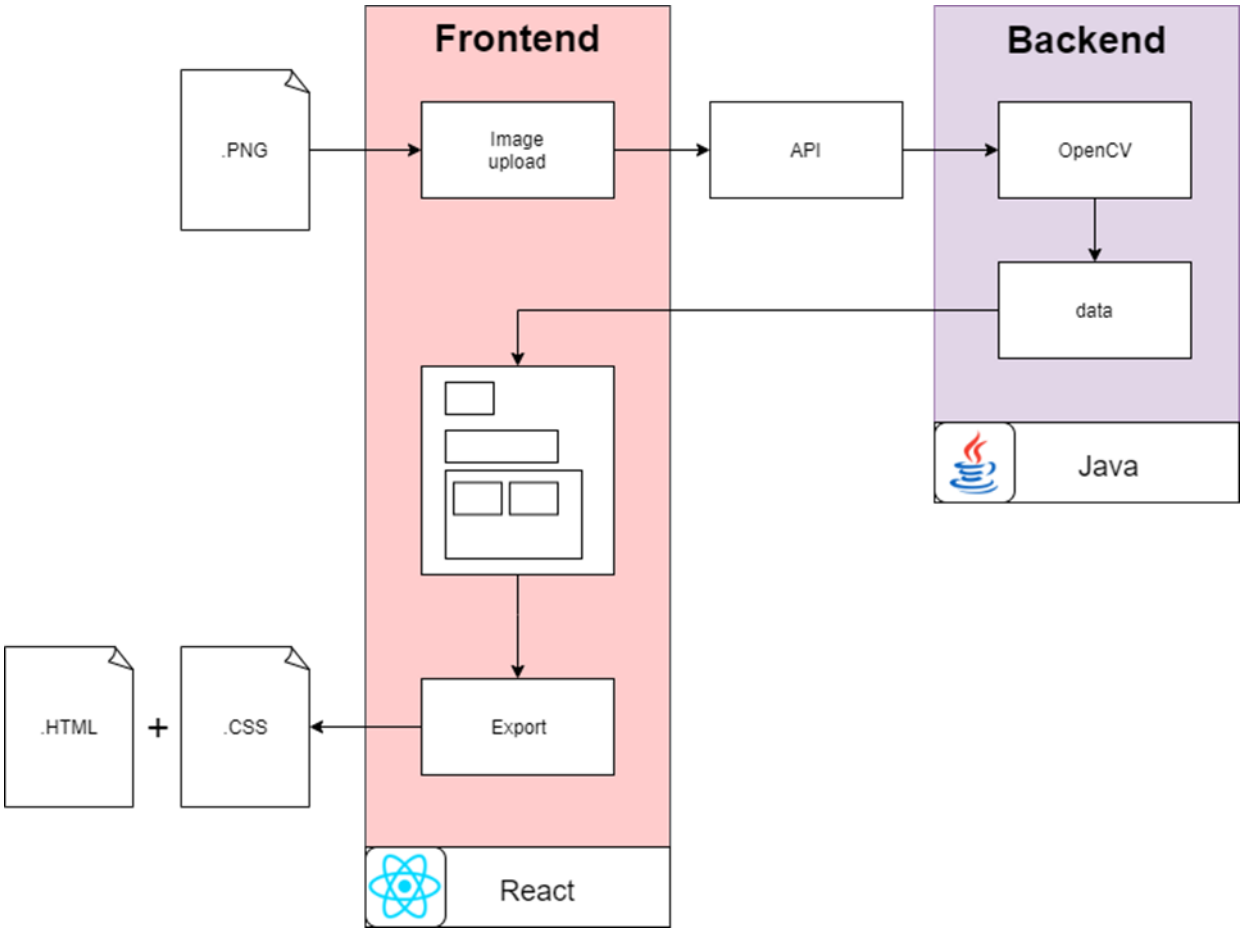
[9] https://en.wikipedia.org/wiki/OpenCV - link na stranicu Wikipedije na kojoj je objašnjena OpenCV biblioteka

Osnovni modul služi za definiranje osnovnih funkcija koje koriste drugi moduli i definiranje struktura podataka. Modul za procesiranje slike sadrži linearno i nelinearno filtriranje slike, geometrijske transformacije slike itd.

4. Sastavni dijelovi rada

Praktični dio završnog rada podijeljen je na nekoliko cjelina. Dvije temeljne cjeline su *frontend* i *backend*.

Slika 4.1. opisuje tok završnog rada. Započinje se s učitavanjem slike koja se uz pomoć *drag and drop* ubaci u alat za učitavanje slike. Nakon uspješnog učitavanja, radi se API poziv na *backend* koji je pisan u Javi uz korištenje OpenCV biblioteke. *Backend* procesira sliku, raspoznaje oblike na njoj te sakuplja podatke o koordinatama gornjeg lijevog kuta, duljini i visini te provjerava da li element sadrži *children* elemente. Podatke vraća u obliku JSON datoteke nazad u *frontend* dio gdje se dodatno obrađuju podaci.



Slika 4.1. Grafički prikaz toka završnog rada

5. *Frontend* struktura

Frontend se sastoji od `Navigation` komponente (statički dio) i rezultata renderiranja React Router-a (dinamički dio). React Router omogućava da se renderira sadržaj ovisno na kojem se URL korisnik nalazi. U kodu 5.1. mogu se vidjeti `Switch` i `Route` koji pripadaju React Router biblioteci. `Switch` će vratiti samo jednu rutu. `Route` specificira URL pod kojim će se renderirati njen sadržaj.

```
const App = () => {
  return (
    <div className={styles.app}>
      <Navigation />
      <Switch>
        <Route exact path="/">
          <FileUploadPage />
        </Route>
        <Route exact path="/layoutedit">
          <LayoutEditPage />
        </Route>
      </Switch>
    </div>
  );
};
```

Kôd 5.1. Osnova *frontenda*

`Navigation` je komponenta koja sadrži gumbe za navigaciju.

`FileUploadPage` i `LayoutEditPage` su komponente koje predstavljaju stranice. Svaka od njih se sastoji od drugih komponenta. `FileUploadPage` služi za slanje slike. `LayoutEditPage` služi za uređivanje rasporeda stranice te izvoz iste.

6. Učitavanje slike

Prvi korak s kojim započinje završni rad je učitavanje i slanje slike. Slika može biti nacrtana ručno ili uz pomoć alata u programima poput MS Paint. Poželjno je da bude u .png ili .jpeg formatu.

6.1. Učitavanje i slanje slike

Unutar `FileUploadPage` datoteke nalazi se `DragDropper` komponenta koja sadrži funkcije za *event handler* za *drag and drop* način učitavanja slike koje su prikazane u kodu 6.1.

```
const handleOnDrop = event => {
    setIsHovering(false);
    const { type } = event.dataTransfer.files[0];

    if (SUPPORTED_TYPES.indexOf(type) > -1) {
      const reader = new FileReader();
      reader.onload = event => setPreviewImage
            (event.target.result);
      reader.readAsDataURL(event.dataTransfer.files[0]);

      onDrop(event.dataTransfer.files[0]);
    }
    event.preventDefault();
  };
```

Kôd 6.1. *Event handler* za ispuštanje slike

Korisnik, nakon što odabere i ispusti sliku u `DragDropper` komponentu, dobije prikaz slike te se prikaže gumb za sljedeći korak. Istovremeno se pripreme podaci za stvaranje API poziva kao što je prikazano u kodu 6.2.

```
const handleDropperDrop = file => {
    const payload = new FormData();
    payload.append("image", file);

    const xhr = new XMLHttpRequest();
    xhr.onreadystatechange = () => {
      if (xhr.readyState === 4) {
```

```
        setResponse(JSON.parse(xhr.response));
      }
    };
    xhr.open("POST",
      "http://localhost:8080/App/upload");
    xhr.send(payload);
  };
```

Kôd 6.2. Funkcija za slanje slike

Pritiskom na gumb, poziva se metoda *handleClick* koju opisuje isječak koda 6.3. Kroz nju se šalju podatci primljeni kao odgovor od API poziva i prebacuje se na novu stranicu čija putanja (engl. *path*) odgovara poslanom parametru `/layoutedit`.

Jedna od biblioteka koju sadrži React Router je `history`. Ona omogućava lakše upravljanje sesijom povijesti uz korištenje JavaScript-a. Uz njen API je moguće je lakše upravljati stogom povijesti (engl. *history stack*), navigirati te očuvati stanje (engl. *state*) između sesija.

```
const handleClick = () => {
    history.push("/layoutedit", { data: response });
  };
```

Kôd 6.3. *Event handler* nakon što se pritisne gumb

7. Obrada slike

API za procesiranje slike opisan u kodu 7.1. API dobiva dijelove (`parts`) iz `request-a` i provjerava da li je koji od njih slika. Sadržaj `part-a` se dobiva u obliku `InputStream-a`.

`BabVision` je centralna klasa za procesiranje slika te sadrži funkcije: `getImageManager` koja vraća instancu klase `ImageManager` i `getImageProcessor` koja vraća instancu klase `ImageProcessor`.

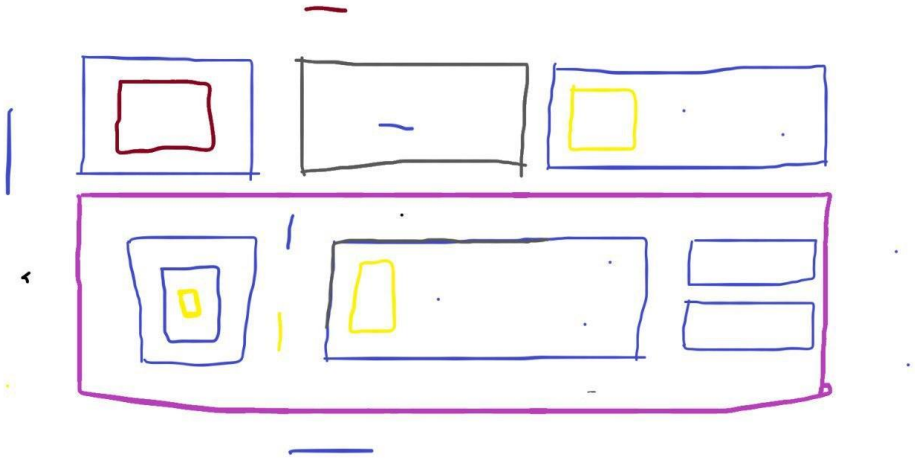
```
Collection<Part> parts = request.getParts();

        if (parts.size() >= 1) {
            Part filePart = request.getPart("image");
            InputStream imageInputStream = filePart
                        .getInputStream();

            String json = BabVision.getImageProcessor(
                BabVision.getImageManager()
                    .loadImage(imageInputStream))
                    .applyGrayscale()
                    .applyThreshold(127, 255)
                    .applyDilation(4, 4)
                    .findContours()
                        .filterByPointCount(4)
                        .filterByBoundingHeight(20)
                        .filterByBoundingWidth(20)
                        .filterByFrame(40, 40)
                        .exportDataAsJson();

            PrintWriter out = response.getWriter();
            out.print(json);
            out.flush();
}
```

Kôd 7.1. API za procesiranje slike

7.1. Učitavanje slike – `Image Manager`

`ImageManager` je klasa koja je odgovorna za učitavanje slike putem funkcije `loadImage`. Funkcija `loadImage` učitava sliku iz `InputSteam-a` i pohranjuje ju u `Mat`.

`Mat` je klasa u OpenCV koja zapravo predstavlja matricu. Sastoji se od dva dijela: zaglavlja matrice i *pointera* na matricu. Matrično zaglavlje sadrži informacije o veličini matrice, metodu koja se koristi za pohranu, gdje je pohranjena matrica itd. *Pointer* na matricu sadrži vrijednosti piksela. Zaglavlje matrice je konstantne veličine iako veličina same matrice varira ovisno o veličini dobivene slike.



Slika 7.1. Izvorna slika

7.2. Procesiranje slike – `Image Processor`

Klasa `ImageProcessor` omogućava primjenu različitih operacija na izvornoj slici koja se dobije u `Mat` obliku. Konačni cilj je smanjiti razinu detalja za potrebu prepoznavanja bitnih karakteristika, u ovom slučaju oblika.

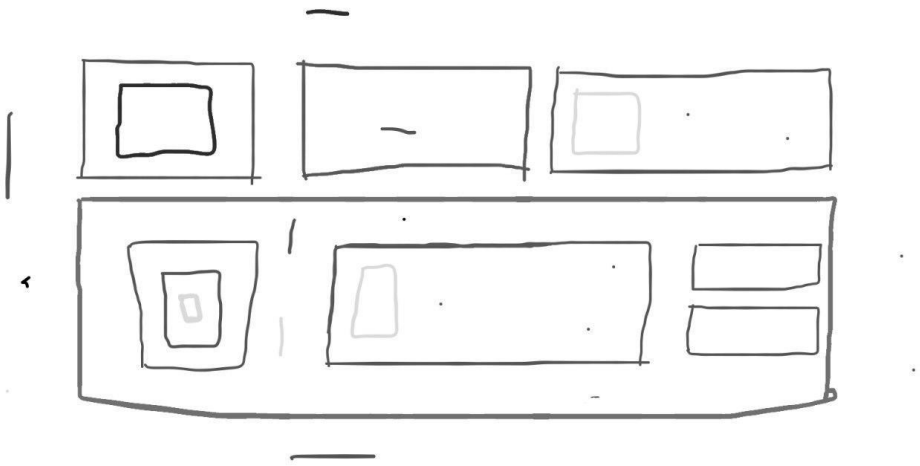
7.2.1. *Grayscale*

Dobivenu sliku je potrebno pretvoriti u sliku sa samo sivim tonovima (engl. *Grayscale*) čime se postiže smanjenje razine detalja u njoj. Funkcija `cvtColor` pretvara sliku iz jednog spektra boja u drugi. Prima tri parametra: izvornu sliku, izlaznu sliku te šifru

konverzije (engl. *conversion code*) koja određuje iz kojeg spektra u koji korisnik želi da se boja promijeni. U isječku koda 7.2. to je `COLOR_BGR2GRAY`.

```
public ImageProcessor applyGrayscale() {
      Imgproc.cvtColor(current, temporary, Imgproc.COLOR_BGR2
      GRAY);
      clearTemporary();

      return this;
    }
```

Kôd 7.2. Funkcija za promjenu boje slike u sive tonove



Slika 7.2. Slika nakon promijene boja

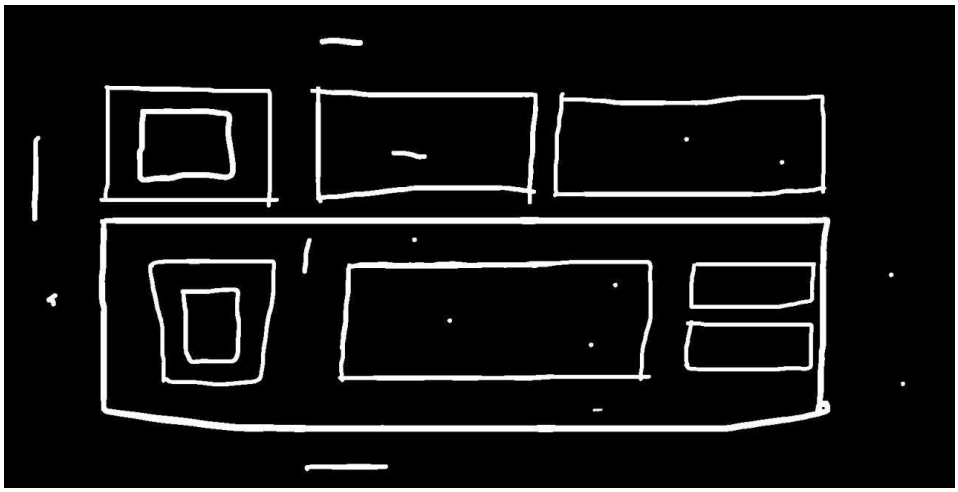
7.2.2. *Treshold*

Tresholding je metoda segmentiranja slike u svrhu stvaranja binarne slike kako bi se dodatno pojednostavili detalji (kod 7.3.). Slika u sivim tonovima može sadržavati 255 različitih vrijednosti. *Tresholding* pretvara sliku, koja je prethodno bila u sivim tonovima, u sliku u isključivo s crnom i bijelom bojom. Ukoliko je stavljena vrijednost *tresholda* na 125, sve što ima vrijednost manju od 125 pretvara se u crnu boju, odnosno sve, što ima vrijednost iznad 125, postaje bijelo.

```
public ImageProcessor applyThreshold(double th, double max) {
      Imgproc.threshold(current, temporary, th, max, Imgproc.
      THRESH_BINARY_INV);
      clearTemporary();

      return this;
```

```
}
```

Kôd 7.3. Funkcija za primjenu *treshold*-a



Slika 7.3. Binarna slika

7.2.3. Proširenje – *Dilation*

Morfološke transformacije su operacije bazirane na promjeni oblika slike. Jedna od njih je proširenje (engl. *dilation*) koja prima tri parametra kao što je opisano u isječku koda 7.4. Prvi parametar je originalna slika, drugi parametar je izlazna slika, a treći je strukturirajući element, odnosno *kernel* koji se koristi za proširenje. Funkcija proširuje izvornu sliku koristeći strukturirajući element koji određuje oblik susjednih piksela. Proširenjem se podebljavaju elementi kako bi se „zakrpala“ neka linija koja nije u potpunosti zatvorena.

```
public ImageProcessor applyDilation(double width,
double height) {
      Imgproc.dilate(current, temporary, CVUtils.createKernel
      (width, height));
        clearTemporary();

        return this;
    }
```

Kôd 7.4. Funkcija za primjenjivanje proširenja na elementima

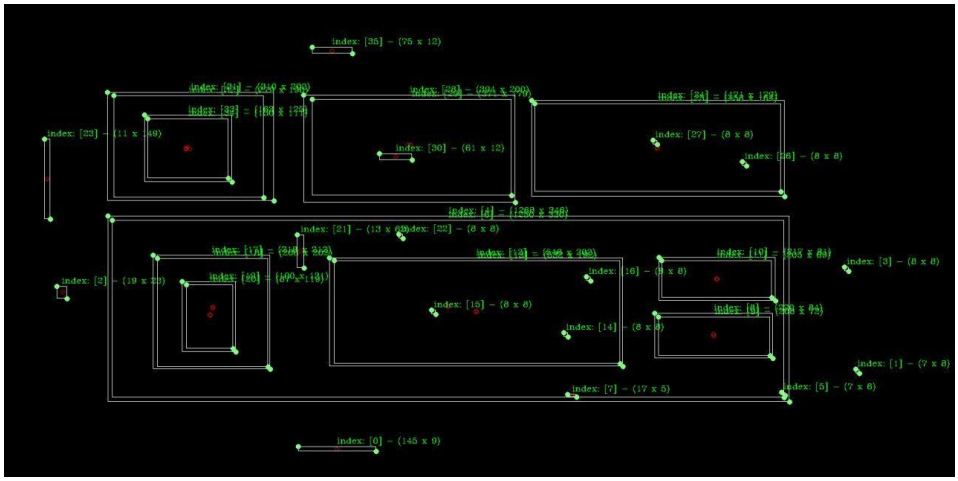
7.2.4. Konturiranje

Konture su skup točaka istih boja i intenziteta, a međusobno se nastavljaju jedna na drugu. Sadrže x i y koordinate rubova nekog oblika. Koriste se za analizu oblika, detekciju i prepoznavanje oblika. OpenCV dok pretražuje oblike, traži ih kao bijeli objekt na crnoj pozadini te zbog toga bi pozadina trebala biti crne boje.

```
public ContourProcessor findContours() {
        Mat hierarchy = new Mat();
        List<MatOfPoint> points = new ArrayList<>();

        Imgproc.findContours(current, points, hierarchy, Imgp
        roc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE);

        return new ContourProcessor(points, hierarchy);
    }
```

Kôd 7.5. Funkcija za pronalazak i određivanje kontura

Funkcija `findContours` iz koda 7.5. prima 5 parametra. Prvi je izvor slike na kojoj se traže konture. Drugi parametar su konture koje su zapravo lista točaka. `Hierarchy` predstavlja međusobni odnos između točaka u konturama. Zadnja dva parametra su funkcije *contour retrieval mode* i *contour approximation method. Contour approximation method* funkcija određuje da li konture pohranjuju sve koordinate nekog oblika. `Chain_approx_simple` u obzir uzima samo krajnje točke neke linije.



Slika 7.4. Slika prepoznatih elemenata nakon konturiranja

Osim konturiranja potrebno je i provesti filtriranje kako bi se kao rezultat dobilo samo ono što je potrebno za ovaj projekt. Svi elementi koji ne zadovolje slijedeće kriterije ne uklanjaju

se, nego se samo uklanja njihov indeks u nizu elemenata koji se provjeravaju. Prvo filtriranje se provodi kroz `filterByPointCount` funkciju, prikazanu u kodu 7.6., koja u obzir uzima samo elemente koji imaju najmanje 4 kuta.

```
public ContourProcessor filterByPointCount(int points) {
        for (int index : data.getIndexes()) {
            if (data.getPolygon(index)
                  .toArray().length < points) {
                data.exclude(index);
                root.remove(index);
            }
        }
        return this;
    }
```

Kôd 7.6. Funkcija za provjeru broja kutova

Zatim se filtrira po minimalnoj visini i širini kroz `filterByBoundingHeight` i `filterByBoundingWidth` funkcije prikazane u kodu 7.7.

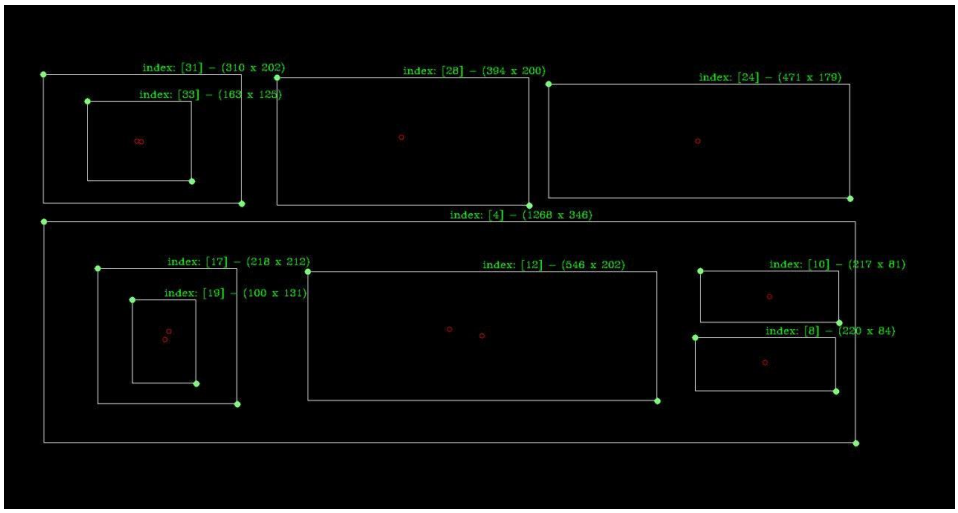
```
public ContourProcessor filterByBoundingHeight(double min) {
        for (int index : data.getIndexes()) {
           if (data.getBoundingRectangle(index).height < min)
            {
                data.exclude(index);
                root.remove(index);
            }
        }
        return this;
    }

 public ContourProcessor filterByBoundingWidth(double min) {
        for (int index : data.getIndexes()) {
            if (data.getBoundingRectangle(index).width < min)
            {
                data.exclude(index);
                root.remove(index);
            }
        }
        return this;
    }
```

Kôd 7.7. Funkcije za određivanje minimalne visine i širine

U konačnici `filterByFrame` provjerava da li minimalna razlika u visini i širini kontura elemenata odgovara zadanoj (kod 7.8.). Svaki element ima unutarnju i vanjsku konturu. Vanjska kontura sadrži točno jedan *child* element i nikad nema više od jednog. Ukoliko apsolutna razlika u visini ili širini *parent* i *child* elementa je manja od zadane vrijednosti, uklanja se indeks tog elementa iz niza indeksa.

```
public ContourProcessor filterByFrame(int width, int height)
{
        for (int index : data.getIndexes()) {
           ContourHierarchyNode parent = root.find(index);
           if (parent != null && parent.getChildCount() == 1)
            {
                ContourHierarchyNode firstChild = parent
                  .getChildren().get(0);
                Rect parentRect = data.getBoundingRectangle
                  (parent.getIndex());
                Rect childRect = data
                  .getBoundingRectangle(firstChild.getIndex()
            );
                double widthDiff = Math
                  .abs(parentRect.width - childRect.width);
                double heightDiff = Math
                  .abs(parentRect.height - childRect.height);
              if (widthDiff <= width || heightDiff <= height)
                {
                  data.exclude(firstChild.getIndex());
                  parent.remove(firstChild.getIndex());
                }
            }
        }
        return this;
    }
```

Kôd 7.8. Funkcija za provjeru razlike u visini unutarnjeg i vanjskog obruba elementa



Slika 7.5. Konačni rezultat filtriranja

Nakon što se završi filtriranje, od preostalih elemenata se stvara JSON objekt koji se dalje šalje kao `response` kroz API.

8. Uređivanje rasporeda elemenata i izvoz

Iz podataka dobivenih u `response` u `LayoutEditPage` datoteci stvara se komponenta `Rectangle` koja prima podatke o svakom četverokutu i njegovim *children* ukoliko ih ima kao što se može vidjeti u kodu 8.1. Također dobiva podatke o dubini (engl. *depth*) koja označava dubinu *child* četverokuta.

```
<div className={styles.editScreen}
ref={editScreenRef}>
  {rectangles &&
    rectangles.map(rectangle => {
      return (
        <Rectangle
          data={rectangle.data}
          children={rectangle.children}
          depth={1}
        ></Rectangle>
      );
    })}
</div>
```

Kôd 8.1. Generiranje četverokuta

8.1. `Rectangle`

`Rectangle` je komponenta u kojoj se nalaze podaci i funkcije za generiranje četverokuta.

```
const [position, setPosition] = useState({ x: realX, y: realY
});
const [size, setSize] = useState({ w: width, h: height });
const [rectangleId] = useState("rectangle_" + index);
```

Kôd 8.2. *State hooks* u `Rectangle` kompomenti.

U kodu 8.2. se nalaze tri *state hooks* koja služe za čuvanje stanja (engl. *state*) o koordinatama, dimenzijama i ID četverokuta.

Za *render* komponente četverokuta koristi se vanjska biblioteka react-rnd koja stvara komponentu koju je moguće micati i mijenjati joj dimenzije.

```
return (
```

```
    <Rnd
      id={rectangleId}
      tabIndex="0"
      bounds="parent"
      disableDragging={isActive ? false : true}
      size={{ width: size.w, height: size.h }}
      maxHeight="100%"
      maxWidth="100%"
      position={{ x: position.x, y: position.y }}
      onDragStop={(e, d) => {
        setPosition({ x: d.x, y: d.y });
      }}
      onResizeStop={(e, direction, ref, delta, position) => {
        setSize({ w: ref.style.width, h: ref.style.height });
      }}
      className={styles.rectangle}
      style={{
        backgroundColor: getBackgroundColor()
      }}
      key={index}
    >
      <span>{index}</span>
      {renderChildren()}
    </Rnd>
  );
};
```

Kôd 8.3. Ispis `Rnd` komponente

U kodu 8.3. prikazano je kako se ispisuje `Rnd` komponenta koja sadrži atribute za dimenzije, poziciju te funkcije za upravljanje događajima nakon što se završe akcije pomicanja i promjene dimenzija u kojima se mijenjaju vrijednosti stanja iz koda 8.2.

Komponenta se može pomicati samo ako je aktivna, što sprječava istovremeno pomicanje *parent* elementa zajedno da *child* elementom.

```
const active = useActiveElement();
const isActive = active && active.id === rectangleId;
```

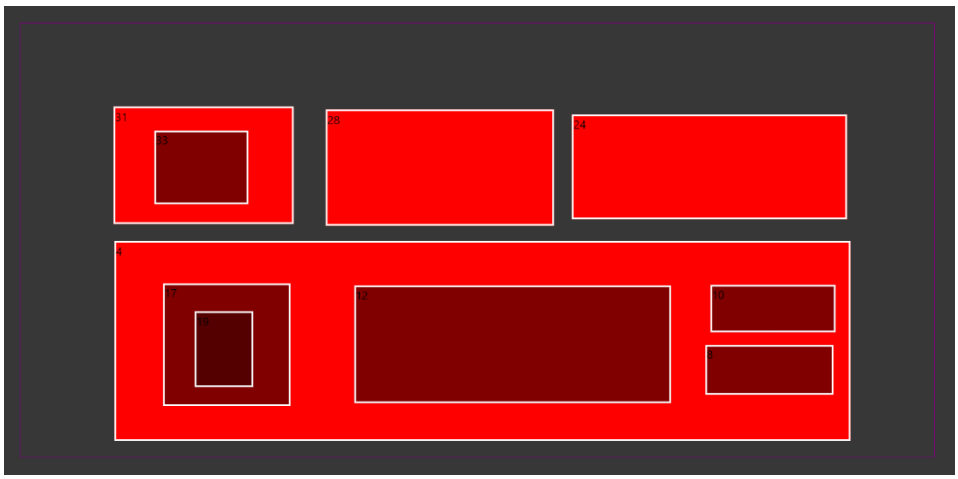
Kôd 8.4. Provjera da li je element aktivan

Isječak koda 8.4. određuje da li je element aktivan. Ovisno o vrijednosti *boolean*-a `isActive`, `disableDragging` iz koda 8.3. omogućuje odnosno onemogućava pomicanje elemenata. `useActiveElement` je *custom hook*. Prilagođeni *hook* (engl. *custom*) je *hook* koji je stvoren od strane korisnika. To je funkcija koja započinje sa *use* i poziva druge *hook*-ove.

```
const useActiveElement = () => {
  const [active, setActive] = useState(document.activeElement
);

  const handleFocusIn = e => {
    setActive(document.activeElement);
  };

  useEffect(() => {
    document.addEventListener("focusin", handleFocusIn);
    return () => {
      document.removeEventListener("focusin", handleFocusIn);
    };
  }, []);

  return active;
};
```

Kôd 8.5. `useActiveElement` *custom hook*

U kodu 8.5. prikazan je `useActiveElement` *custom hook* koji služi za dohvaćanje trenutno aktivnog elementa (element koji je u fokusu) na stranici.



Slika 8.1. Prikaz ekrana za uređivanje rasporeda elemenata

8.2. Izvoz datoteka

Nakon što korisnik završi s uređivanjem izgleda željene web stranice, pritiskom na gumb „Set current data as text“ postavlja trenutno prikazane podatke kao vrijednosti tekstualnih datoteka. Proces je opisan u isječku koda 8.6. gdje to odrađuje funkcija `handleSetText`.

```
const handleSetText = () => {
    const { current } = editScreenRef;

    if (current) {
      const { children } = current;

      if (children.length > 0) {
        let text = {
          html: [],
          css: [
            "\n.rectangle {
            \n max-width:100%;
            \nmax-height:100%;
            \n position:absolute;\n}\n"
          ]
        };

        for (const child of children) {
          const { html, css } = processRectangle(child, 0);
          if (css) text.css.push(css);
          if (html) text.html.push(html);
        }

        const basicHtmlTemplate = `\n<html>
            \n<head>\n<title>
            \nHello\n</title>
            \n</head>\n<body>
            ${text.html.join("")}
            \n</body>
            \n</html>\n`;

        setHtmlText(basicHtmlTemplate);
        setCssText(text.css.join(""));
      }
```

```
    }
  };
```

Kôd 8.6. funkcija za postavljanje vrijednosti tekstualnih datoteka

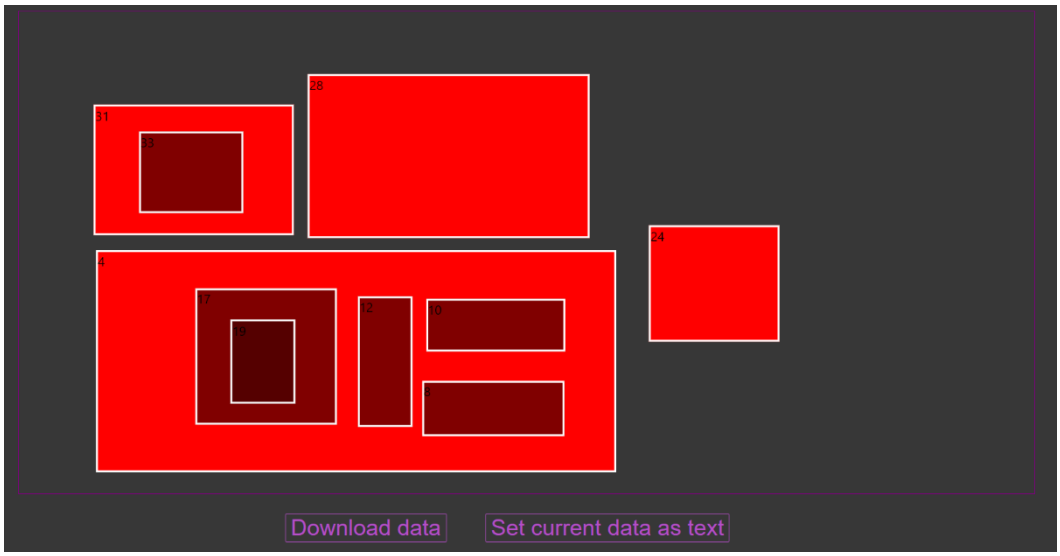
Iz reference na DOM element, uzima se kolekcija *child* elemenata te za svaki od njih izvršava se funkcija `processRectangle` čiji je rezultat objekt koji sadržava HTML i CSS tekst za taj element. Rezultati su kombinirani u jedan objekt koji sadrži HTML i CSS tekst cijele stranice.

Prethodno spomenuta funkcija `processRectangle` prima *child* element s njegovom `depth` vrijednosti. Destrukturiranjem primljenog elementa dobiva se `style` koji sadrži sve podatke o CSS-u svakog elementa. U isječku koda 8.7. opisan je proces dobivanja CSS svojstava potrebnih za ovaj završni rad, a to su `width`, `height` te `transform`. Njihovi nazivi pohranjeni su unutar `INDIVIDUAL_CSS_PROPS` konstante.

```
if (cssText) {
          const cssProperties = cssText.split(";");
          for (let i = 0; i < cssProperties.length - 1; i++){
            const cssProp = cssProperties[i];
            let isUseful = false;

            for (const useful of INDIVIDUAL_CSS_PROPS) {
              if (
                cssProp.includes(useful) &&
                !cssProp.includes("max-width") &&
                !cssProp.includes("max-height")
              ) {
                isUseful = true;
                break;
              }
            }

            if (isUseful)
            childrenText.css.push(`\n${cssProp};`);
          }
        }
```

Kôd 8.7. Upisivanje CSS atributa i vrijednosti

Prolaskom kroz sva CSS svojstva, filtriraju se samo oni koji su potrebni te se spremaju u `childrenText` objekt koji se kasnije vraća kao što je prikazano u isječku koda 8.6.



Slika 8.2. Prikaz ekrana za uređivanje nakon provedenih promjena

Na slici 8.2. prikazan je ekran za uređivanje nakon provedenih željenih promjena. Ukoliko je korisnik zadovoljan prikazanim, pritiskom na gumb „Download data“ dobiva generirane HTML i CSS datoteke čiji je izgled prikazan u isječcima koda 8.8. i 8.9.

```
<!DOCTYPE html>
<html>

<head>
  <title>Hello</title>
  <meta charset="utf-8" />
  <link href="StyleSheet.css" rel="stylesheet" />
</head>

<body>

<div id="rectangle_31" class="rectangle">
  <div id="rectangle_33" class="rectangle">
  </div>
</div>
<div id="rectangle_28" class="rectangle">
</div>
<div id="rectangle_24" class="rectangle">
</div>
<div id="rectangle_4" class="rectangle">
  <div id="rectangle_17" class="rectangle">
    <div id="rectangle_19" class="rectangle">
    </div>
```

```
  </div>
  <div id="rectangle_12" class="rectangle">
  </div>
  <div id="rectangle_10" class="rectangle">
  </div>
  <div id="rectangle_8" class="rectangle">
  </div>
</div>

</body>

</html>
```

Kôd 8.8. Izgled dobivene HTML datoteke

```
.rectangle {
 max-width:100%;
 max-height:100%;
 position:absolute;
}

#rectangle_31 {
 width: 249px;
 height: 162px;
 transform: translate(282.6px, 214px);
}

#rectangle_33 {
 width: 130px;
 height: 102px;
 transform: translate(54px, 31px);
}

#rectangle_28 {
 width: 315px;
 height: 161px;
 transform: translate(575.6px, 218px);
}

#rectangle_24 {
 width: 380px;
```

```
 height: 145px;
 transform: translate(915.6px, 225px);
}

#rectangle_4 {
 width: 1017px;
 height: 276px;
 transform: translate(283.6px, 400px);
}

#rectangle_17 {
 width: 176px;
 height: 170px;
 transform: translate(65px, 56px);
}

#rectangle_19 {
 width: 81px;
 height: 105px;
 transform: translate(42px, 37px);
}

#rectangle_12 {
 width: 438px;
 height: 163px;
 transform: translate(329px, 59px);
}

#rectangle_10 {
 width: 173px;
 height: 66px;
 transform: translate(821px, 58px);
}

#rectangle_8 {
 width: 177px;
 height: 69px;
 transform: translate(814px, 141px);
}
```

Kôd 8.9. Izgled dobivene CSS datoteke

Zaključak

Postojeći alati za izradu web stranica su jednostavni za korištenje, pružaju mnogo opcija korisniku za uređivanje i stvaranje web stranica, iako ne posjeduju znanja potrebna za pisanje koda. Međutim, korisnici koji žele nešto više od tih servisa, poput mogućnosti izvoza HTML i CSS datoteka za kasnije uređivanje u okruženju po želji, nemaju tu mogućnost.

U ovome radu omogućena je korisniku maksimalna sloboda stvaranja izgleda rasporeda web stranice. Korisnik može vlastoručno ili uz pomoć alata u npr. MS Paint napraviti skicu željenog izgleda rasporeda web stranice. Nakon slanja i učitavanja slike, korisnik ima dodatnu mogućnost promjene pozicije i veličine elemenata. U konačnici, dobiva rezultat u obliku HTML i CSS datoteka.

Ova aplikacija sadrži veliki razvojni potencijal. U budućnosti, aplikacija se može proširiti s dodatnim mogućnostima poput postavljanja boje obruba ili pozadine elemenata te dodavanjem mnogih drugih funkcionalnosti.

Popis kratica

API	*Application Programming Interface*	sučelje za programiranje aplikacija
CSS	Cascading Style Sheets	listovi kaskadnog stila
DOM	*Document Object Model*	model objekta dokumenta
ECMA	*European Computer Manufacturers Association*	udruženje europskih proizvođača računala
HTML	*HyperText Markup Language*	jezik za označavanje hiperteksta
HTTP	*Hypertext Transfer Protocol*	protokol za transfer hiperteksta
JSON	*JavaScript Object Notation*	notacija za JavaScript objekt
JSX	*JavaScript XML*	JavaScript XML
JVM	*Java Virtual Machine*	Java virtualni stroj
XML	*Extensible Markup Language*	jezik za označavanje podataka
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja

Popis slika

Popis kôdova

Literatura

[1] ACKERMANN, F. *JavaScript Das umfassende Handbuch*. Rheinwerk Computing, 2016.

[2] BASSET, L. *Introduction to JavaScript Object Notation*. O'Reilly Media, 2015.

[3] DUCKETT, J. *HTML & CSS design and build websites*. John Wiley & Sons, 2011.

[4] OPENCV, https://opencv.org, listopad. 2019.

[5] REACT, https://reactjs.org, studeni. 2019.

ALGEBRA
VISOKO UČILIŠTE

JAVA WEB ALAT ZA IZRADU STATIČKE HTML STRANICE

Pristupnik: Iva Vukelić, 0068212643

Mentor: prof. Aleksander Radovan