АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

**Фазлић Иван**

**Радни оквир „Next.js“: основе и примена у пројектовању**

**веб апликација**

**- завршни рад -**



Београд, септембар 2022.

Кандидат: **Фазлић Иван**

Број индекса: **НЕТ-25/19**

Студијски програм: **Нове енергетске технологије**

Тема: **Радни оквир „Next.js“: основе и примена у пројектовању**

**веб апликација**

Основни задаци:

**1. Основе за рад у Next.js-у**

**2. Прављење веб апликације у радном оквиру**

**3. Бенефити коришћења радног оквира**

Ментор:

Београд, септембар 2022. годинe.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

др Зоран Ћировић, проф. ВИШЕР

Резиме:

Циљ овог рада је упознавање и стицање неопходног знања ради ефикасног рада у овом окружењу. У раду су описане технологије које се примењују приликом пројектовања веб апликација. Демонстриран је прелазак са JavaScript-a у UI библиотеку React, а затим миграција из ње у Next.js радни оквир. Извршена је аутоматизација изградње скелета апликације путем npx наредбе која BootStrap-ује основну структуру апликације. Кроз прављење блог апликације пролазимо примере функционалности радног оквира.

**Кључне речи:** : Пројектовање, апликација, окружење, технологија, Next.js, React

ABSTRACT:

The goal of this final work is to get to know and acquire the necessary knowledge for effective work in this environment. The paper describes the technologies that are applied when designing web applications. Moving from JavaScript to the React UI library and then migrating from it to the Next.js framework is demonstrated. The automation of building the skeleton of the application was performed through the npx command, which BootStraps the basic structure of the application. Through the creation of a blog application, we go through examples of the functionality of the working framework.

**Key words**: Designing, application, framework, technology, Next.js, React

садржај

[1. УВОД 1](#_Toc115047254)

[2. Опис технологија коришћених у раду 2](#_Toc115047255)

[2.1. HTML5 2](#_Toc115047256)

[2.2. CSS3 3](#_Toc115047257)

[2.3. JavaScript 4](#_Toc115047258)

[2.4. Остале технологије 4](#_Toc115047259)

[3. Основе за радни оквир Next.js 5](#_Toc115047260)

[3.1. Прелазак са JavaScript-а на React 6](#_Toc115047261)

[3.1.1. Разлика између императивног и декларативног програмирања 7](#_Toc115047262)

[3.1.2. JSX 7](#_Toc115047263)

[3.1.3. Компоненте 8](#_Toc115047264)

[3.1.4. Својства 9](#_Toc115047265)

[3.1.5. Стање 9](#_Toc115047266)

[3.2. Прелазак са React-a на Next.js 9](#_Toc115047267)

[3.2.1. Node.js - npm 9](#_Toc115047268)

[3.2.2. Миграција React кода у Next.js 10](#_Toc115047269)

[4. Прављење блог апликације 12](#_Toc115047270)

[4.1. Навигација између станица 12](#_Toc115047271)

[4.2. Средства 13](#_Toc115047272)

[4.3. Мета-подаци 14](#_Toc115047273)

[4.4. CSS 14](#_Toc115047274)

[4.5. Рендеровање пре приказивања и преузимање података 15](#_Toc115047275)

[4.6. Динамичке руте 16](#_Toc115047276)

[4.7. API руте 19](#_Toc115047277)

[5. ЗАКЉУЧАК 20](#_Toc115047278)

[6. ИНДЕКС ПОЈМОВА 21](#_Toc115047279)

[7. ЛИТЕРАТУРА 22](#_Toc115047280)

[8. Прилози 23](#_Toc115047281)

[9. Изјава о академскoј честитости 24](#_Toc115047282)

**СПИСАК КОРИШЋЕНИХ СКРАЋЕНИЦА**

*HTML - HyperText Markup Language*

*DOM* - *Document Object Model*

*CSS* - *Cascade Style Sheets*

*JS - JavaScript*

*JSON - JavaScript Object Notation*

*NPM - Node Package Manager*

*JSX - JavaScript XML*

*VSCODE - Visual Studio Code*

*TS- TypeScript*

*URL - Uniform Resource Locator*

*ID - Identification*

*CDN - Content Delivery Network*

*CMS - Content Management System*

*UI - User Interface*

*SDK - Software Development Kit*

*NPX - Node Package Execute*

*WebP - Web Picture format*

*API - Application programming interface*

**СПИСАК СЛИКА И ПРИМЕРА**

*3.1.1.1.**Императивно програмирање*

*3.1.1.2. Приказ h1 елемента*

*3.1.2.1. Декларативно програмирање*

*3.1.2.2. Основи React-a*

*3.2.1. Зависности*

*4.1.1. Садржај фајла prva\_objava.js*

*4.2.1. Image компонента*

*4.4.1. Садржај izgled.js фајла*

*4.4.2. Увођење модуларног CSS-а*

*4.5.1. Преузимање података*

*4.5.2. Прослеђивање преузетих података*

*4.6.1. Динамичке руте*[5]

*4.6.2. Садржај фајла [id].js*

*4.6.3. Садржај фајла posts.js*

*4.6.4. Ажуриран код фајла posts.js*

*4.7.1. API рута*

# УВОД

Прликом дефинисања теме завршног рада одлучено је о приказивању радног оквира Next.js и његове примене у пројектовању веб апликација. Веб апликација која ће бити направљена у овом раду проћи ће кроз фундаменталне концепте због којих се ова технологија истиче од већ дуго познатих. Са обризом на то да је овај рад примарно оријентисан ка демонстрацији Next.js-а, подразумеваће се претходно знање из:

* HTML-a;
* CSS-a;
* JavaScript-а,

а пожељно је и такође знање React-а, али није у потпоности неопходно.

JavaScript и React су обимне теме. Како знати када мигрирати у Next.js? У овом раду биће описане основе JavaScriptа, миграција са њега на React, па коначно на Next.js. Креираћемо малу блог апликацију путем које ћемо проћи кроз појмове попут:

* Навигација између страница;
* Средства, мета-подаци и CSS;
* Рендеровање пре приказивања и преузимање података;
* Динамичке руте;
* API руте.

Ови појмови су разлог због којег је ова технологија стекла своју популарност. Сваки од појмова ће бити детаљно обрађен у својем поглављу и биће дати бенефити њиховог коришћења.

Поред свих ових функционалност Next.js подржава TypeScript, а највећа предност је што има невероватан SEO(енгл. **S**earch **E**ngine **O**ptimization). Окружење нуди многе опције попут генерисања статичких сајтова, као и рендеровање са серверске стране, а уколико се захтева оба постоји и та могућност.

Примарно, исход овог рада треба да обухвата:

* Иновативна решења за проблеме који настају приликом прављења корисничких интерфејса;
* Побољшања у брзини доставе и прераде података клијенту;
* Организован и ефективан начин чувања кода ради угодног искуства приликом програмирања апликације.

# Опис технологија коришћених у раду

Next.js сам по себи функционише на принципу других техологија, поготово JavaScript-а. Споменућемо технологије које су фундементалне за правилан рад и генерално могућност рада у овом окружењу:

* *HTML5*
* *CSS3;*
* *JavaScript;*
* *JSX*
* *JSON*
* *React.js*
* *Node.js*
* *Next.js*

Текст едитор изабран за демонстрацију и анализу радног окружења ће бити *Visual Studio Code.*

Одабрани Веб прегледач ће бити *Google Chrome.* Такође ћемо користити његове *“Алатке за развој веба”.*

## HTML5

*HTML* [1] (енгл. ***H****yper****t****ext* ***M****arkup* ***L****anguage*) је код намењен за прављење и приказ структуре веб странице и њеног садржаја. Екстензија за документ је *.html.* Овај језик се унутар веб претраживача интерпретира, након чега се приказује клијенту, на њему разумљив начин. Садржи низ елемената који обухватају садржај унутар њихових тагова.

Елемент се дефинише са отвореним и затвореним тагом, где се први таг конструише са знаком “мање од” (“<”), потом тип елемента (параграф, наслов, слика), а онда са затвореним знаком “веће од” (“>”). Већина елемената мора бити затворена, а затварају се тако што се у други таг дода коса црта између знака “мање од” и типа елемента. Пример елемента са садржајем: *“<p>Садржај</p>”*. Елементи могу да садрже атрибуте који им дају додатну функционалност и омогућавају интеграцију са другим технологијама као што је на пример CSS.

Целокупан HTML код се налази унутар почетног „<*html*>“ и завршног „</*html*>“ тага. Унутар овог елемента угнеждени су *“<head></ head >”* таги *“<body></body>”* таг*.* Елемент *“<head></ head >”* служи као контејнер за метаподатке[[1]](#footnote-1) и смештен је између *html* и *body* тага. Елемент *“<body></body>”* дефинише документ. У њега је смештен целокупан садржај документа, као и сви остали тагови. Налази се између затвореног *head* тага и затвореног *html* тага.

Елементи могу садржати атрибуте унутар њих ради додатних функционалност. Ово се најчешће ради да би елемнт могао да се пронађе и стилизује. Пример је: “<div **class=’mojaKlasa’ id=’mojId’**></div>”, где су “ **class=’mojaKlasa’** “ и “ **id=’mojId’** ” атрибути.

Један веома битан појам код HTML-а је DOM (енгл. **D**ocument **O**bject **M**odel). Приликом рашчлањивања HTML-а веб претраживач конструише дрво[[2]](#footnote-2) објеката (DOM). Том структуром се може манипулисати JavaScript-ом ради динамичке измене садржаја.

## CSS3

*CSS* [1] (енгл. ***C****ascading* ***S****tyle* ***S****heets*) је језик који описује како HTML елементи треба да се приказују на екрану кориснику. Екстензија документа за овај језик је .*css*. Може се писати директно у HTML путем *“<style></ style >”* тага, али може се писати у засебном документу након чега се путем “*<link rel='stylesheet’ href=style.css’> </link>“* тага споји са HTML документом. Синтакса му је следећа:

циљани елемент {

својство: вредност;

својство: вредност;

својство: вредност;

...

}.

Циљани елемент може да буде сам елемент, име класе, ID(енгл. *Identification*) елемента итд. Уколико је сам елемент онда у ово поље стављамо име елемента (“div”). Уколико је класа, стављамо тачку и потом име класе (“.mojaKlasa”), а уколико је ID, стављамо тарабу те име ID-a (“#mojId”). На овај начин циљамо елемент чија својства желимо да доделимо или променимо. CSS може бити писан и као атрубут неког елемнта (“<div style=’svojstvo: vrednost;’></div>”), али ова пракса се ретко примењује због хиjерархиjске природе доделе својства, као и због читљивости кода. Својства која могу да се измене или доделе зависе од елемента који је у питању, а вредности зависе од својства. Неки од примера су:

* *h2 { color: lightblue; }* (*h2* – селектор; *color* – својство; *lightblue* – вредност)
* *.mojaKlasa { width: 20px; }* (*.mojaKlasa* – селектор; *width* – својство; *20px* – вредност)
* *#mojId {height: 40px; }* (*#mojId* – селектор; *width* – својство; *20px* – вредност)

Додатне могућности CSS-а су псеудо селектори који олакшавају дохватање и манипулацију елементима у DOM-у. Није потребно експлицитно наводити имена боја код доделе вредности. За то се може користити систем *RGB(A)* (енгл. ***R****ed*, ***G****reen* a*n*d ***B****lue* (***A****lpha*)) или *HSL*(*A*) (енгл. ***H****ue*, ***S****aturation*, ***L****uminance* (***A****lpha*)).

У израдама апликација често се у пракси користи CSS и LSS библиотека *Font Awesome*, која име велику количину бесплатних иконица и фонтова.

## JavaScript

*JavaScript* [1] је један од најпопуларнијих скриптних језика. За разлику од програмских језика који се компајлирају, JavaScript се интерпретира[[3]](#footnote-3). Спада у динамичке језике високог нивоа, слабо је типизиран и извршава се у потпуности унутар веб прегледача. Стандардизован је по ECMAScript-у. Као и све алатке и технологије у овом раду бесплатан је за коришћење. Екстензија документа је .js. Главни разлог за његову потребу је манипулисање DOM-ом. Уз помоћ њега, програмeру/архитекти апликације, је дата могућност за креирање HTML елемната, измену, додавање интеракције клијент-веб апликација као и многе друге ствари. Исход примене треба да буде омогућавање клијенту да има угодно искуство приликом коришћења апликације. За израду веб апликација често се користе библиотеке и радни оквири ради убрзавања процеса изградње. Такође се све чешће користи и на серверским станама ради прављења REST API-ја (енгл. ***A****pplication* ***P****rogramming* ***I****nterface*) у Node.js-у и другим технологијама.

Код овог језика у оквиру HTML-а се пише унутар “<script></script>” тага. Могуће je, а и у пракси најпопуларније, поготово приликом израде великих апликација, линковање JS документа унутар HTML документа. Ради се на сличан начин с тим што се у *“src”* атрибуту “*<script>”* тага наводи путања до JavaScript документа (пример: „*<script src='app.js'></script>“*).

Главни разлог популарности овог језика лежи у томе што унутар њега постоје асинхроне функције, које помажу програмеру да заобиђе чекање неких потенцијално дугих задатака за извршавање. Поменуте функције пружају кориснику могућност интеракције са осталим садржајем веб апликације без да је у обавези да чека извршавање иницијалног захтева док се он у подзадини обрађује. По успешном извршавању функције се кориснику приказује исход захтева који може бити успех или грешка.

## Остале технологије

Остатак технологија које су наведене у Поглављу 2 *(Опис технологија коришћених у раду)* биће обрађене детаљније у селедећем Поглављу 3 *(Основе за радни оквир Next.js).*

# Основе за радни оквир “Next.js”

Постоје ствари које треба узети у обзир приликом грађења веб апликација, као што су[5]:

* Кориснички интерфејс – како корисници долазе до података;
* Рутирање – како се корисници крећу између различитих страница;
* Дохваћање података – где се подаци налазе и како доћи до њих;
* Рендеринг - када и где се приказује статички или динамички садржај;
* Интеграција - које услуге треће стране користите (CMS, аутентификација, плаћања, итд.) и како се повезујете са њима;
* Инфраструктуре – где постављате, чувате и покрећете свој код апликације (без сервера, CDN, Edge, итд.);
* Перформансе - како да се апликација оптимизује за крајње кориснике;
* Скалабилност – како се ваша апликација прилагођава како ваш тим, подаци и саобраћај расту;
* Искуство програмера – искуство вашег тима у изградњи и одржавању ваше апликације.

За сваку ову ставку на програмеру је да ли ће сам да изгради решење или да искористи неке библиотеке/радне оквире.

React је JavaScript библиотека намењена за прављење корисничких интерфејса. Он нуди корисне функције за изградњу интерфејса, али препушта програмеру да oдлучи где ће те фунскције применити. Осим овога нема других утицаја на изградњу апликацјие и то је један од разлога због којег је доживео процват свог екосистема. Међутим, то значи да изградња React апликације из темеља захтева одређени напор. Програмери су принуђени да троше своје време ради конфигурисања алата и поновног решавања уобичајних захтева апликације.

Next.js је радни оквир за React. Служи да рукује са алатима потребним за React и пружа додатне функције, структуру и оптимизацију апликацији. За изградњу корисничких интерфејса може да се искористи React, а затим на то да се додају функицоналности Next.js-а као што су рутирање, дохваћање података, интеграције – све уз побољшање искуства програмера и корисника.

## Прелазак са JavaScript-а на React

Када корисник приступи некој веб страници, сервер му достави HTML документ, након чега претраживач прочита садржај и генерише DOM. У одабраном текстуалном докумнету ћемо да додати HTML документ по називом “index.html” и конструисати његов скелет. Пречица у VSCode-у за ово је куцање карактера “!” унутар документа и клик на дугме “enter”. У овом случају нећемо је искростити, већ ћемо ручно исписати код.

<html>

  <body>

    <div id="app"></div>

    <script type="text/javascript">

      // Selekcija div elementa sa 'app' id-em

      const app = document.getElementById('app');

      // Kreiranje novo h1 elementa

      const header = document.createElement('h1');

      // Kreiranje tekst Node-a za h1 element

      const headerContent = document.createTextNode(

        'Садржај h1 тага.',

      );

      // Dodela teksa h1 elementu

      header.appendChild(headerContent);

      // Postavljanje h1 elementa unutar div-a

      app.appendChild(header);

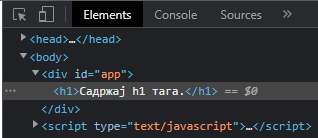
    </script>

  </body>

</html>

*Пример 3.1.1.1. – Императивно програмирање*

Код приказан у овом примеру је намењен да демонстрира да у иницијалном HTML-у не постоји елемент *h1*. Уколико ако десном тастером миша кликнемо на неко место на страници унутар претраживача, те одемо на опцију *Inspect* у менију са стране ће се отворити прозор, приликом чега идемо на опцију *Elements*.

*Слика 3.1.1.2. – Приказ h1 елемента*

Примећујемо да је додат елемнт. Зашто се он појављује овде иако га нисмо унутар HTML-а написали? Разлог за ово је што смо унутар JavaScript-а дали инструкције претраживачу да направи овај елемент, додели му вредност и коначно смести унутар div-а. Овим смо ажирирали ДОМ.

Писање оволико кода ради додељивања само једног елемента захтева доста напора, а при писању већих апликација готово је немогуће. Ради мимоилажења овог проблема користимо декларативни начин програмирања уместо императивног.

### Разлика између императивног и декларативног програмирања

Претходни код (Пример 1) демонстрира императивно програмирање. Своди се на давање детаљних инструкција претраживачу **како** да направи елемент, који садржај да му додели и где да га смести. Декларативни приступ је често префериран јер тежи да олакша процес изградње апликације. Омогућава програмеру да декларише **шта** жели да буде приказано.

React је декларативна UI(енгл. User Interface) библиотека. Унутар њега довољно је рећи шта је потребно да се прикаже а он ће сам смислити како то да реализује.

### JSX

JSX (енгл. ***J****ava****S****cript* ***X****ML*) је синтаксна екстензија JavaScript-а која нуди могућност лакшег и разумљивијег писања JavaScript кода путем познате HTML-налик синтаксе. Претраживачи не могу да разумеју ову синтаксу, зато је потребно искористити компајлер, као што је *Babel*, да би се JSX превео у JavaScript.

У следећем коду приказан је декларативан начин програмирања уз помоћ React-а и JSX -а:

<html>

  <body>

    <div id="app"></div>

    <!-- React Script -->

    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>

    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

    <!-- Babel Script -->

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

    <script type="text/jsx">

      const app = document.getElementById("app")

      ReactDOM.render("Deklatativno programiranje", app)

    </script>

  </body>

</html>

*Пример 3.1.2.1. – Декларативно програмирање*

У поређењу са императивним начином визуално се примећује разика у коду. Све што је било протребно је додавање скрипта које омогућавају коришћење React-а и његовог DOM-а унутар HTML документа. Такође је додат и Babel да би се JSX, који смо додали у линији “11” унутар ReactDOM.render методе, компајлирао у чист JavaScript.

Следећи код у овом раду (*Пример 3.1.2.2.*), биће коришћен ради кратког објашњења фундаменталних концепта React-а, а то су:

* Компоненте;
* Својства;
* Стање.

<script type="text/jsx">

      const app = document.getElementById("app")

      function Header({naslov, koga}) {

        return <h1>{naslov} {koga ? koga : 'osoba'}:</h1>;

      }

      function PocetnaStrana() {

        const imena = ['Pera Peric', 'Ivan Fazlic', 'Nemanja Nemanjic'];

        const [lajkovi,setLajkovi] = React.useState(0);

        function handleKlik() {

          setLajkovi(lajkovi+1);

        }

        return (

        <div>

          <Header naslov="Spisak" koga="ljudi"/>

          <ul>

            {imena.map((ime) => (

              <li key={ime}>{ime}</li>

            ))}

          </ul>

          <button onClick={handleKlik}>Like({lajkovi})</button>

        </div>

    );

      }

       ReactDOM.render(<PocetnaStrana />, app)

    </script>

*Пример 3.1.2.2. – Основи React-a*

### Компоненте

Прва ствар на коју треба обратити пажњу су главне функције HomePage и Header, које представљају компоненте. Представљју независне и поновно употребљиве блокове кода. Компоненте увек почињу са великим словом да би могле да се разликују од функција. Свака компонента је функција, али свака функција није компонента [3]. Оне враћају UI елементе, могу бити угнеждене и могу се користити унутар JSX-а попут HTML елемената, с тим што је име елемента назив компоненте. Пример иницијализације компоненте: “<Header />”. Унутар ове иницијализације опционо је додељивање својства.

### Својства

Својсва се додељују приликом иницијализације компоненте по принципу кључ-вредност. Прихватају се као објекат “props” (енгл. скраћеница од “*Properties*”) који се користи као параметар компоненте (функције). Садржи кључеве претходно дефинисане у оквиру иницијализације. У 27. линији кода (*Пример 3.1.2.2)* постоје два кључа и њихове вредности које прихватамо унутар компоненте којој припадју. Уместо да унутар параметара компоненте “*Header*” кориснимо објекат “*props*”, можемо да извршимо деконструкцију како би касније лакше примењивали вредности унутар компоненте. Кључеви и њихове вредности су опциони, и компоненте могу враћати вредности и без њих. У линији 17 приказано је коришћење тернарног оператора, који уколико “koga” кључ није постављен исписује се вредност “osoba”, а уколико је постављен исписује се вредност дефинисана приликом иницијализације.

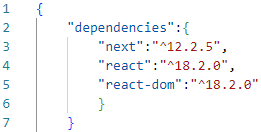
### Стање

Приликом коришћења стања, неопходно је “слушање за догађајима”. Догађај дефинишемо неком радњом коју је корисник извршио над апликацијом (клик, скрол, предавање форме итд.). Стање може да се замисли као било која информација унутар корисничког интерфејса која се промени услед догађаја. Hooks су функције које нам омогућавају рад над стањима. Једна од најпопуларнијих је “*useState*” hook. Она као повратну вредност даје низ којима је могућ приступ и коришћење путем деконструкције низа. Приликом деконструкције, стање прве променљиве је иницијализовано на вредност унутар заграда “*useState*” hook-а. Манипулацију над овом променљивом вршимо са другом променљивом, у овом случају то је функција враћеног низа. Норма и пракса у програмирању је да ова функција има префикс “set”, те име прве променљиве. У линиији 33 кликом на дугме “Like” позива се функција “handleKlik” која инкрементира вредност “lajkovi” променљиве за 1, путем функције “setLajkovi”.

## Прелазак са React-a на Next.js

### Node.js - npm

Приликом додавања Next.js-a пројекту, неће више бити потребе за React скриптама унутар HTML документа. Користиће се npm (енгл*.* ***N****ode* ***P****ackage* ***M****anager*) ради инсталације пакета локално. Node.js-ов екосистем пакета, npm, је највећи екосистем библиотека отвореног кода на свету [2]. У директоријуму ћемо креирати нови фајл по имену “package.json” и у њега додати текст “{}”. Унутар интегрисаног терминала у оквиру VSCODE-а покрећемо команду “npm install react react-dom next”. Након извршавања ове команде може да се примети да је унутар package.json фајла неки текст (слика 3.2.1.).

*Слика 3.2.1. Зависности*

Текст из слике нам говори од којих модула **зависи** програм који пишемо. Без ових модула наш програм неће моћи да функционише уколико користимо функције из поменутих модула. Package.json даје информације npm-у које модуле да инсталира приликом извршавања команде “npm install”. Једна олакшица, која као исход има штедњу времена и ресурса, је да приликом објављивања апликације у програм за контролу верзије (попут GIT-a) се само дода package.json фајл уместо свих модула који могу имати више стотина мегабајта или чак гигабајта. Разлог ове праксе је то што приликом преузимања package.json-а, претходно наведеном командом “npm install”, се инсталирају сви модули од којех програм зависи.

### Миграција React кода у Next.js

Из кода у примеру 3.1.2.2. (код из примера 3.1.2.1. с тим што је у примеру 3.1.2.2. измењен скрипт таг), писан у оквиру index.html-a, уклањамо следеће ставке:

* Линије 11, 12, 13, 14,37, 38 у потпуности,
* <html> и <body> тагове,
* Линију 21 *“React.”* део од *“React.useState(0)”.*

Потребно је додати следећу лнију кода на врх документа: “import { useState } from ‘react’” и променити екстензију фајла да буде “.js” уместо “.html”. Преостаје да се документ дода у директоријум “pages” како би Next.js знао да се ради о станици, и да се испред корен-компоненте дода “export default”. Коначан код изгледа овако:

import { useState } from "react";

const app = document.getElementById("app")

function Header({naslov, koga}) {

    return <h1>{naslov} {koga ? koga : 'osoba'}:</h1>;

}

export default function PocetnaStrana() {

    const imena = ['Pera Peric', 'Ivan Fazlic', 'Nemanja Nemanjic'];

    const [lajkovi,setLajkovi] = useState(0);

    function handleKlik() {

        setLajkovi(lajkovi+1);

    }

    return (

        <div>

            <Header naslov="Spisak" koga="ljudi"/>

            <ul>

            {imena.map((ime) => (

                <li key={ime}>{ime}</li>

            ))}

            </ul>

            <button onClick={handleKlik}>Like({lajkovi})</button>

        </div>

    );

}

*Пример 3.2.2.1. Коначан Next.js код*

Ради тестирања кода потребно је да покренемо развојнои сервер. Унутар package.json-а додајемо скрипту “"scripts": {"dev": "next dev"}” . Овим смо добили могућност да командом “npm run dev” подигнемо/покренемо сервер на порту 3000 (уобичајном порту за Node.js). Ова функционалност се зове “Брзо освежавање”. Приказуе тренутне повратне информације о свим изменама које се направе и долази унапред конфигурисан са Next.js-ом [4]. Сада приликом чувања фајла промене се рефлектују скоро у реалном времену (< 1s).

У поређењу са кодом писаним у HTML-у путем React скрипт-и и Babel-а, види се знатно побољшање у читљивости кода и генерално број линија и обимност. Ово су само неке од погодности коришћења Next.js-а.У следећем делу рада (Поглавље 4 “*Прављење блог апликација*”) биће демонстриране различите алатке попут брзог освежавања, које помажу програмеру да своје време користи што продуктивније. Биће приказан и начини достављања садржаја клијентима да њихов боравак у апликацији буде што угоднији.

# Прављење блог апликације

Приликом креирања Next.js апликације, искористићемо npx (енгл. ***N****ode* ***P****ackage* ***E****xecute*) команду која нам генерише костур сајта и конфигурише апликацију, да ми то не би морали ручно да радимо. Потребно је отворити терминал и укуцамо команду “npx create-next-app <име апликације>” након чега треба сачекати да npm инсталира модуле. По завршетку покрећемо скрипту “npm run dev”, која покреће развојни сервер на порту 3000. Унутар index.js фајла, који се налази у pages директоријуму, уклањамо саржај из main тага и остављамо само први “h1” таг.

## Навигација између станица

Унутар “pages” директоријум индекс страница је повезана са “/” рутом. У Next.js-у страница је извоз(енгл. “*Export*”) главне компоненте фајла. Страница “pages/index.js” је повезана са “/” рутом, а страница “pages/primer/prva\_objava.js” је повезана са “/primer/prva\_objava” рутом.

Унутар pages директоријума потребно је креирати фолдер по имену “primer”, и унутар њега “prva\_objava.js” документ. Садржај овог документа треба да буде:

export default function PrvaObjava() {

    return <h1>Prva objava</h1>;

}.

*Пример 4.1.1. Садржај фајла prva\_objava.js*

Приликом навигације на URL ‘<http://localhost:3000/primer/prva_objava>’ види се текст који исписује “Prva objava”. Ово је начин на који се руте дефинишу унутар овог радног оквира.

У Next.js-у користи се Link компонента да би се страница повезала са другим страницама унутар апликације. Служи ради навигације са клијентске стране и прихвата “props” објекат тј. својства (Поглавље 3.1.4.) ради боље контроле над навигацијом. Сада, у index.js фајлу је потребно увести “Link” компонент командом “import” из библиотеке next/link и изменити “h2” таг у следећи код:

        <h1 className={styles.title}>

          Procitajte <Link href="/primer/prva\_objava">ovu stranicu!</Link>

        </h1>

У документу prva\_objava.js је такође потребно увести Link компоненту и додати унутар кода линк који враћа назад на почетну - “/” страницу. То се постиже стављањем косе црте у “href” атрипут Link компоненте. Овим смо успешно спојили две странице.

Разлог коришћења Link компоненте уместо обичног HTML “а” тага је то што се коришћењем Link компоненте навигација врши на **клијентској страни**. То значи да се прелазак са једне на другу страницу ради путем JavaScript-а што је знатно брже него уобичајна навигација коју прегледач користи. Next.js аутоматски извршава раздвајање кода и почиње да преверемено учитава садржај других Link компоненти да би чекање садржаја, након што га клијен затражи, буде што краће.Ово понашање може да буде условљено (да ли је садржај на екрану корисника или корисник тек треба да дође до њега).

## Средства

Средства унутар Next.js-а се налазе унутар “public” директоријума. Public директорију, налик на pages, референцира корен апликације. Користи се такође и за складиштење robot.txt-а. Унутар HTML-а слика се додаје путем следећег кода:

‘<img src="/images/profile.jpg" alt="Ime osobe" />’.

Овај начин додавања није оптималан јер се програмер сусреће са низом проблема као што су:

* Учитавање слике само када је неопходно (када корисник скролом дође до ње),
* Оптимизација слике за неку алатку или библиотеку,
* Да је величина слике одговарајућа за све величине екрана итд.

Next.js омогућава коришћење “image” компоненте која решава све ове проблеме. Такође има уграђену “Оптимизацију Слика”. Подржава сервирање слика у модерним форматима попут WebP (енгл. ***Web P****icture format*) , наравно када тај формат има подршку од претраживача. То спречава слање великих слика кориснику ако за приступ апликацији користи уређај са малим екраном. Ова функционалност важи за слике сваких типова, чак и за слике које су хостоване са неког екстерног ресурса попут система за управљање садржајем.

Уместо да се слике оптимизују приликом изградње, слике се оптимизују приликом њиховог захтевања. Ово је велика предност у односу на генераторе статичких сајтова, јер се време изградње не мења да л' имали 10 или 10 милиона слика. Слике користе “lazyload” стратегију, ако се не напомене другачије. То значи да брзна странице не испашта због елемената до којих корисник још није стигао. Такође, оптимизоване су на начин да би што ефективније избегле наглу промену елемената на екрану. Ово понашање помаже сајту приликом рангирања садржаја на интернету. Начин на који се корисни ова компонента је следећи:

import Image from 'next/image';

const ImeKomponente = () => (

  <Image

    src="/images/profile.jpg" // Рута до слике

    height={144} // Висина слике приликом које се одржава однос

    width={144} // Ширина слике приликом које се одржава однос

    alt="Ime Korisnika"

  />

);

*Пример 4.2.1. Image компонента*

Потребно је да се уведе компонента “Image” из одговрајуће библиотеке, након чега се користи унутар функције као засебна компонента.

## Мета-подаци

Процес промене и доделе мета-података унутар Next-а је следећи. Унутар документа “prva\_objava.js” у “pages” директоријуму, уводимо/импортујемо “Head” компоненту. Следеће што је потребно урадити је унутар подразумеваног export/извоза “PrvaObjava”, изнад Script компоненте, додати Head компоненту у којој је угнежден title таг са садржајем “прва објава”. Head компонента се користи и за додавање JavaScript-а од програма треће стране. Ови програми генерално уводе функционалност као што су рекламе, аналитика и виџети.

Начин на који се ово постиже је следећи. Унутар Head компоненте додаје се script HTML таг <script src="https://connect.facebook.net/en\_US/sdk.js" />. Ова скрипта садржи Facebook SDK (енгл. ***S****oftware* ***D****evelopment* ***K****it*). Проблем са оваквим учитавањем скрипта је што не можемо да знамо када ће се скрипта учитати у односу на стални садржај сајта. Уколико нека скрипта блокира рендеровање странице, ова драстично може да одузме на преформанси.

Један од начина којим можемо заобићи овај проблем је коришћење Next.js-ове Script компоненте. Ова компонента омогућава имплементирање стратегија учитавања скрипта. Једна од њих је “lazyOnload” која омогућује бољу расподелу ресурса. Уколико се определимо за ову стратегију, она нам пружа могућност да се скрипте учитају након оставлих елемената у времену мировања претраживача. Скрипт компоненти је могуће додати и onLoad атрибут како би претраживач знао коју функцију да позове након што се иницијална скрипта изврши.

## CSS

Приликом иницијализације Next.js апликације, унутар “styles” директоријума постоје два CSS фајла, а то су глобални (globals.css) и модуларни ([Home.module.css](http://Home.module.css)). Модули CSS-а омогућавају коришћење истих имена класа у различитим документима без бриге од преклапању класа. Морају да имају “module” између имена документа и његове екстензије како би се разликовали од обичног CSS-а. Поред већ уведених модула Next.js могућава стилизовање апликације на разне начине као што су:

* Sass који омогућава увоз .css и .scss датотека
* PostCSS библиотеке као што је Tailwind CSS
* CSS-у-JS-у библиотеке као што су styled-jsx, styled-components и emotion

Приликом прављења апликације потребно је направити репозиторијум по имену компоненте. Унутар њега направити фајл izgled.js са следећим садржајем:

export default function Layout({ children }) {

    return <div>{children}</div>;

}

*Пример 4.4.1. Садржај izgled.js фајла*

Потом је потребно унутар “prva\_objava.js” импортовати ову компоненту и поставити да буде коренска компонента враћене вредности. У репозиторијуму “компоненте” треба направити фајл izgled.module.css. Да би класе из модула могле да се користе, унутар izgled.js се импортује овај модул по произвољном имену.

import styles from './izgled.module.css';

export default function Layout({ children }) {

  return <div className={styles.container}>{children}</div>;

}

*Пример 4.4.2. Увођење модуларног CSS-а*

Овако се постиже модуларно додељивање CSS-а. Други начин додељивања је глобално. Ово се постиже тако што се унутар “styles/globals.css” додају произвољне класе.

## Рендеровање пре приказивања и преузимање података

Next.js аутоматски извршава рендеровања података пре приказивања. Странице су достављене у виду HTML фајлова са минималном количином JavaScript-а. Овај начин приказивања садржаја може да помогне при преформансама и за SEO. Након што су странице учитане извршава се JavaScript код, кроз процес који се зове хидратација, и странице постају потпуно интерактивне.

Next.js подржава два типа рендеровања података пре приказивања, а то су статичко генерисање и рендеровање на серверској страни. Статичко генерисање значи да се дохватање података дешава приликом времена изградње. Генеришу се HTML странице које се изнова користе након сваког захтева. Рендеровање са серверске стране је метода која се користи за генерисање HTML страница на сваки захтев. Овај начин је спорији и уколико је могуће заобилази се. Приликом бирања начина рендеровања потребно је поставити питање да ли ове податке могу да спремим пре него што корисник да захтев. Ако је одговор да, у том случају се опредељујемо за статичко генерисање.

У овом раду фокус је примарно на статичком генерисању. Тренутно унутар пројекта нису направљене странице које захтевају податке из екстерних ресурса. Статичко генерисање је могуће и уколико се ови подаци налазне унутар фајл система, на API-ју, као и бази података. Ово се постиже са асинхроном функцијом “getStaticProps()”.

Симулација дохватања података је извршена помоћу фајл система. Направљен је директоријум “posts” који садржи два фајла (‘Renderovanje-Pre-Prikazivanja.md’ и ‘Staticko-Generisanje.md’). За рашчлањивање метаподатака из фајлова искоришћен је npm модул “gray-matter”. На врху апликације је додат фолдер по називу lib у коренском директоријуму, који садржи логику за извлачење и коришћење података из “posts” директоријума. Унутар pages/index.js користимо „getStaticProps()” асинхрону функцију.

import { getSortedPostsData } from '../lib/posts';

export async function getStaticProps() {

  const allPostsData = getSortedPostsData();

  return {

    props: {

      allPostsData,

    },

  };

}

*Пример 4.5.1. Преузимање података*

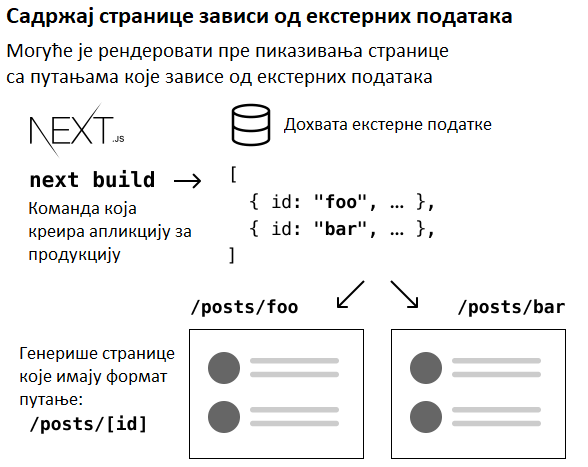
Овај код нам омогућава да проследимо податке компоненти <Home /> у виду props објекта, где их након тога прихватамо и користимо унутар компоненте.

export default function Home ({ allPostsData }) { ... }.

*Пример 4.5.2. Прослеђивање преузетих података*

## Динамичке руте

У претходној секцији Поглавља 4( *Рендеровање пре приказивања и преузимање података)* смо приказали да кориснимо функцију getStaticProps() уколико наша веб страница зависи од података који су складиштени на неком екстерном ресурсу(фајл систему, на API-ју или бази података). Динамичке руте или путање нам допуштају да статички генеришемо путање које зависе од екстерних података. Ова функционалност је погодна и долази уз Next.js.



*Слика 4.6.1. Динамичке руте*[5]

Унутар нашег пројекта је потребно да направимо динамичке руте за наше блог објаве. Ово постижемо тако што:

* Желимо да сваки пост/објава има путању “/primer/<id>”, где је “<id>” име markdown(.md) датотеке у директоријуму “posts” највишег нивоа.
* С обзиром на то да имамо два фајла унутар нашег posts директоријума (“Renderovanje-Pre-Prikazivanja.md” и “Staticko-Generisanje.md”) наше динамичке руте ће бити “/primer/Renderovanje-Pre-Prikazivanja” и “/primer/ Staticko-Generisanje”.

У оквиру Next.js-а странице које почињу са “[“ и завршавају са “]” су динамичке руте. Пример овога је “pages/posts/[id].js” где је “[id].js” динамичка рута. Унутар овог фајла је потребно експортовати асинхрону функцију getStaticPaths. Она враћа листу могућих вредности за “id”. Последњи корак је да имплементирамо getStaticProps поново, да би могли да дохватимо одговарајуће податке за страницу са тим id-ем. Функцији getStaticProps су дати params, који садрже id(из разлог што је име странице “[id].js”)

Имплементација овога је следећа. Потребно је креирати фајл унутар пројекта по називу “[id].js” и сместити га у pages/posts директоријум. Након што се ово уради, уклања се фајл “prva\_objava.js”. У “lib/posts.js” потребно је додати функцију getAllPostIds. Ова функција враћа имена фајлова унутар posts директоријума највишег нивоа. Унутар pages/posts/[id].js треба ставити следећи код:

import { getAllPostIds } from '../../lib/posts';

export async function getStaticPaths() {

  const paths = getAllPostIds();

  return {

    paths,

    fallback: false,

  };

}

*Пример 4.6.2. Садржај фајла [id].js*

Овде константа paths садржи низ који је вратила функција getAllPostIds().

Остало нам је да урадимо још пар ствари. Потребно је да дохватимо податке да би их рендеровали са одговарајућим id-ем. Унутар фајла lib/posts.js додајемо следећи код:

export function getAllPostIds() {

  const fileNames = fs.readdirSync(postsDirectory);

  // Vraca niz koji izgleda ovako:

  // [

  //   {

  //     params: {

  //       id: 'Renderovanje-Pre-Prikazivanja'

  //     }

  //   },

  //   {

  //     params: {

  //       id: 'Staticko-Generisanje'

  //     }

  //   }

  // ]

  return fileNames.map((fileName) => {

    return {

      params: {

        id: fileName.replace(/\.md$/, ''),

      },

    };

  });

}

export async function getPostData(id) {

  const fullPath = path.join(postsDirectory, `${id}.md`);

  const fileContents = fs.readFileSync(fullPath, 'utf8');

  // Iskoristi gray-matter pa parsiras metapodatke iz posts sekcije

  const matterResult = matter(fileContents);

  // Iskoristi remark da pretvoris markdown u HTML string

  const processedContent = await remark()

    .use(html)

    .process(matterResult.content);

  const contentHtml = processedContent.toString();

  // Kombinuj podatke sa id-em i contentHTML-om

  return {

    id,

    contentHtml,

    ...matterResult.data,

  };

}

У pages/posts/[id].js такође треба да додамо следећи код:

import { getAllPostIds, getPostData } from '../../lib/posts';

export async function getStaticProps({ params }) {

  const postData = getPostData(params.id);

  return {

    props: {

      postData,

    },

  };

}

*Пример 4.6.3. Садржај фајла posts.js*

Коначно нам је остало да ажурирамо Posts компоненту и тиме смо завршили прављење динамичких рута.

export default function Post({ postData }) {

  return (

    <Layout>

      {postData.title}

      <br />

      {postData.id}

      <br />

      {postData.date}

    </Layout>

  );

}

*Пример 4.6.4. Ажуриран код фајла posts.js*

Да би тестирали да ли овај код ради можемо да покренемо развојни сервер и отворимо неки од следећих линкова:

* <http://localhost:3000/primer/Staticko-Generisanje>
* [http://localhost:3000/primer/Renderovanje-Pre-Prikazivanja](http://localhost:3000/primer/Renderovanje-Pre-Prikazivanja%20)

## API руте

Прављење REST API-ја унутар Next.js-а је изузетно лако. API руте се стављају у pages/api директоријум. Унутар овог директоријума направићемо фајл по имену “helloworld.js” . Код унутар њега је следећи:

// req = HTTP dolazeca poruka, res = HTTP odgovor servera

export default function handler(req, res) {

  res.status(200).json({ name: 'Hello World' })

}

*Пример 4.7.1.*

Након што приступимо на “<http://localhost:3000/api/helloworld>”, видећемо да нам се враћа одговор у JSON формату{"name":"Hello World"}.

# ЗАКЉУЧАК

Циљ и исход овог рада је био орјентисан ка томе да се демонстрирају основне функционалности Next.js-а. Кроз рад су биле образложене технологије и код употребљен у примерима ради приказивања радног окружења. Битна ствар је прелазак са JavaScript-a на React-а потом на Next.js. Деклеративан начин програмирања је дефинитивно пожељан приликом писања великих апликација или чак апликација било којег обима. Као што је напоменуто у Поглављу 3.1.1.( *Разлика између императивног и декларативног програмирања*) Next.js је окружење у којем се пише декларативан код. Next.js, као што већ сада знамо, је радни оквир за React. Да би смо усавршили Next.js морамо бити упознати са React библиотеком тј. да знамо да примењујемо функционалности React-а као што су компоненте, својства, стање итд.

Уколико постоји могућност аутоматизације понављајућих задатака потребно је тежити ка томе. Bootstrap-овање Next.js апликације нам је омогућено уз помоћ npx команде “npx create-next-app <ime aplikacije>”.

Основе за рад и примери су дати кроз прављење блог апликације. Навигација између страница је урађена уз помоћ Link компоненте, која долази уз Next.js. Средства су приказана коришћењем Image компоненте која служи за оптимизацију формата слике, да на свим уређајима слике могу одговарајуће да се прикажу. За мета-податке је намењена компонента Head. Начини писања CSS-а могу бити модуларни или глобални. Модуларни начин је специфичан за радни оквир и у потпуности елиминише постојање две или више класе са истим именом. Ово се постиже додавањем суфикс на име класе.

Приказана је метода рендеровања пре приказивања - статичко генерисање, а поменуто је да уколико није могуће искористити ову методу постоји и рендеровање са серверске стране. Нисмо у обавези да користимо или једно или дриго већ можемо за неке делове апликације да употребимо статичко генерисање, а за друге рендеровање са серверске стране. Преузимање података, уколико је реч о статичком генерисању, се врши функцијом getStaticProps().

Динамичке руте нам нуде могућност да направимо путање које зависе од неких екстерних фајлова, а АPI руте нам омогућавају коришћење функција без сервера, такође познатие као ламбде(енгл. Lambdas).

Приликом програмирања апликације радно окружење нам је понудило организован и ефективан начин чувања кода и програмирање унутар њега је било поприлично интуитивно. Код покретања сервера за развој “Брзо освежавање” је била велика помоћ, јер сваки пут након чувања кода кориснички интерфејс рефлектује измене у милисекундама. Приликом статичког генерисања сајтова, странице које су генерисане се кеширају унутар CDN-а(енгл. **C**ontent **D**elivery **N**etwork), a то као резултат даје бржу доставу података кориснику.

# ИНДЕКС ПОЈМОВА

**С**

Својства

Скалабилност

Средства

Стање

**Т**

Текст едитор

Терминал

**А**

Асинхроне функције

Aтрибут

**Д**

Декларативно програмирање

Дрво

**Е**

Елемент

**З**

Захтев

**И**

Императивно програмирање

Интеграција

Интерпретирње

Инфраструктура

**К**

Клијент

Клијент-веб апликација

Компајлирање

Компоненте

Корисник

**П**

Прегледач

**Р**

Рендеринг

# ЛИТЕРАТУРА

[1] Julie Meloni, *HTML, CSS, and JavaScript All in One: Covering HTML5, CSS3, and ES6, Sams Teach Yourself 3rd Edition*, Sams Publishing; 3rd edition (November 30, 2018).

[2] Andrew Mead, *Learning Node.js Development*, Packt Publishing, 2018.

[3] Artemij Fedosejev, *React.js Essentials*, Packt Publishing 2015.

[4] Michele Riva, *„Real-World Next.js“*, Packt Publishing 2022.

[5] Vercel, 2022, *Documentation*, viewed 16 august 2022,< https://nextjs.org/docs>.

# Прилози

1. *izgled.js*

2. *izgled.module.css*

3. *posts.js*

4. *helloworld.js*

5. *[id].js*

6. *prva\_objava.js*

7. *\_app.js*

8. *index.js*

9. *Renderovanje-Pre-Prikazivanja.md*

10. *Staticko-Generisanje.md*

11. *profile.jpg*

12. *favicon.ico*

13. *globals.css*

14. *PocetnaStranica.module.css*

15. *utils.module.css*

16. *package.json*

17. *README.md*

# Изјава о академскoј честитости

**ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ**

|  |  |
| --- | --- |
| **Студент (име, име једног родитеља и презиме):** | Фазлић Иван |
| **Број индекса:** | НЕТ-25/19 |

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

Радни оквир „Next.js“: основе и примена у пројектовању веб апликација

1. резултат сопственог истраживачког рада;
2. да овaj рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, \_\_. 09. 2022. године

Својеручни потпис студента

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Подаци о HTML документу који се не приказују. [↑](#footnote-ref-1)
2. Структура података у којој постоји један корен елемент из којег се сви гранају. Html таг се може сматрати корен елементом јер се сви елементи налазе унутар њега. [↑](#footnote-ref-2)
3. Извршава се линију по линију. [↑](#footnote-ref-3)