VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

SMOTRNOST PRENOSA SPLETNIH APLIKACIJ IZ OGRODJA ASP.Net DNN v ReactJS

Kandidat: Rene Krajnc

Vrsta študija: študent izrednega študija

Študijski program: Informatika

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: dr. Ambrož Stropnik

Lektor: Vesna Breznik, prof.

Maribor, 2022

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Rene Krajnc, sem avtor diplomskega dela z naslovom Smotrnost prenosa spletnih aplikacij iz ogrodja ASP.Net DNN v ReactJS, ki sem ga napisala pod mentorstvom mag. Ervina Schaffa.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela,
- sem poskrbel, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor,
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli kot moje lastne kaznivo po Zakonu o avtorski in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili,
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Maribor, september 2022 Podpis študenta:

POVZETEK

Ogrodje ASP.NET DNN (prej imenovano DotNetNuke) je namenjeno razvoju spletnih aplikacij. DNN je CMS (ang. content management system) in vsebuje uporabniški vmesnik, ki omogoča upravljanje administracije nad uporabniškimi računi, ustvarjanje ter urejanje spletnih strani in urejanje naprednih konfiguracij. Zaradi vseh funkcionalnosti CMS je spletna aplikacija v ogrodju DNN večja v velikosti datotek in potrebuje več časa, da naloži vsebino v brskalniku kot brez CMS. Iz tega razloga je spletno aplikacijo, ki nima potrebe po CMS, v določenih primerih smiselno prenesti v drugačno spletno tehnologijo. V diplomskem delu smo si prizadevali izdelati oceno smotrnosti procesa tranzicije spletne aplikacije iz ogrodja DNN v knjižnico ReactJS glede na potrebe aplikacije in glede na zmogljivosti podjetja.

Prvi pogoj za raziskavo procesa tranzicije je bila spletna aplikacija na ogrodju DNN. Iz tega razloga smo razvili lastno spletno aplikacijo QoL (polni naslov je Quality of Life), ki vsebuje elemente in funkcionalnosti najdene v mnogih sodobnih uporabniških vmesnikih, kot je dinamični prikaz podatkov glede na odgovor API klica ali lokalizacija spletnih strani. Naslednji korak je bila analiza QoL v DNN, načrtovanje izvedbe tranzicije in priprava okolja. V nadaljevanju smo se lotili procesa tranzicije na ReactJS, ki je zaradi velikih razlik med spletnima tehnologijama terjala novo implementacijo.
Celotni proces tranzicije smo opisali v drugem poglavju. Primerjava obeh implementacij spletne aplikacije je razkrila, da je vzdrževanje QoL v ReactJS bistveno lažje kot vzdrževanje QoL v DNN. Ker smo spletno aplikacijo razvili v obeh tehnologijah, smo lahko izdelali primerjavo potrebnih znanj za vsako posebej, hkrati pa smo na obeh implementacijah aplikacije QoL lahko izvedli meritve v istem okolju. Izvedene meritve časa nalaganja vsebin v brskalniku in odzivnega časa uporabniških akcij so razkrile, da so pridobitve tranzicije v tej kategorij bile ogromne.
Na podlagi pridobitev tranzicije spletne aplikacije QoL iz DNN v ReactJS, ki se nanašajo na čas nalaganja, odzivnost in težavnost vzdrževanja spletne aplikacije, je za aplikacije, ki so implementirane v DNN in ne potrebujejo CMS, smotrno izvesti tranzicijo na knjižnico ReactJS. V primeru, da v podjetju ni potrebnega znanja za izvedbo, je smotrno vložiti čas in sredstva v izobraževanje razvijalcev, saj je ReactJS dobra izbira za mnogo vrst spletnih aplikacij.

Ključne besede: *Spletni razvoj, front-end, DNN, ReactJS, tranzicija, sodobne spletne tehnologije.*

ABSTRACT

The rationality of migrating web applications from ASP.Net DNN to ReactJS

The ASP.NET framework DNN serves the development of web applications. DNN is a CMS and contains a user interface that allows management of user accounts, creation and modification of websites, and management of advanced settings and configurations. Because of all the functionalities of CMS a web application created on the DNN is bigger in file size and takes more time to show content on the browser than without CMS. That's why it could be beneficial that applications with no need for CMS would be transitioned to a different technology. In the diploma thesis, we tried to make an assessment of the expediency of the transition process of a web application from DNN to the ReactJS library according to the requirements of the application and according to the capabilities of the company.

The first condition for researching the transition process was a web application based on the DNN framework. For this reason, we developed our own QoL web application, which contains elements and functionalities found in many modern user interfaces, such as dynamic display of data according to the response of an API call or localization of web pages. The next step was the analysis of QoL in DNN, the planning of the transition and the preparation of the environment. Next we started the transition process which required a new implementation due to the big differences between the two web technologies. The entire transition process and the findings of the process were described in the second chapter. A comparison of the both implementations of the web application revealed that QoL maintenance in ReactJS is significantly easier than in DNN. Since we developed the web application in both technologies, we were able to make a comparison of the necessary skills for each, and at the same time we were able to perform measurements of load speeds. Measurements of browser load time and user action response time revealed tremendeous gains.

Based on the results of the transition of QoL from DNN to ReactJS, which relate to the loading time, responsiveness and difficulty of maintaining the web application, it is expedient to undertake the transition process for applications that are implemented in DNN and do not need a CMS. In case the company does not have the necessary knowledge for implementation, it is advisable to invest time and resources in the education of developers, since ReactJS is a good choice for many types of web applications.

Keywords: *Web development, front-end, DNN, ReactJS, transition, modern web technologies*

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

KAZALO GRAFIKONOV

Razlaga strokovnih izrazov in kratic

- ASP.NET kontrole (ang. ASP.NET Server Controls): spletne kontrole, ki ustvarijo HTML komponente, ki so dosegljive na strežniški in odjemalčevi strani (ang. server and client based) (Rožman, 2016).
- ReactJS: odprtokodna JavaScript knjižnica za razvoj uporabniških vmesnikov v obliki komponent (React, a JavaScript library, b.d.).
- ASP.NET: odprtokodni okvir spletnih aplikacij na strani strežnika, zasnovan za spletni razvoj za izdelavo dinamičnih spletnih strani (Rožman, 2016).
- DNN: odprtokodno ogrodje ASP.NET za razvoj spletnih aplikacij s CMS (prej imenovano DotNetNuke) (DNN urejevalnik spletnih vsebin, brez datuma).
- CMS: sistem za upravljanje vsebine (ang. Content Management Stytem), ki v spletnih aplikacijah predstavlja uporabniški vmesnik in zaledje za administracijo in kreiranje ter urejanje spletnih vsebin (MOJWEB solutions, brez datuma).
- QoL: kratica za ime lastne spletne aplikacije Quality of life. V nadaljevanju bomo govorili o dveh implementacijah spletne aplikacije QoL, implementiranih v različnih spletnih tehnologijah (MOJWEB solutions, brez datuma).
- URL (ang. Uniform Resource Locator): naslov spletnega mesta v svetovnem spletu.
- Komponenta: osnovni gradnik ReactJS aplikacij in predstavlja neodvisni kos programske kode za večkratno uporabo. ReactJS ima dve vrsti komponent – razredne (ang. Class Components), ki imajo svoje stanje in funkcijske (ang. Functional Components), ki so brez stanja (ang. stateless) (React, a JavaScript library, b.d.).
- Props: props so podatki, posredovani ReactJS komponentam preko HTML atributov. Za razliko od HTML atributov (ki so lahko le string), so lahko katerikoli podatkovni tip (React, a JavaScript library, b.d.).
- Kavelj (ang. hook): funkcija ReactJS, ki funkcijskim komponentam omogoča stanje in upravljanje s stanjem (React, a JavaScript library, b.d.).
- ES6: ECMAScript 6 (skrajšano ES6) je najnovejši JavaScript standard, ki v JavaScript prinaša novosti, kot so nove vrste spremenljivk, novi podatkovni tipi, nove funkcije za iteracijo po seznamih itd (ECMAScript 6, brez datuma).
- Sass (ang. Syntactically Awesome Style Sheets): skriptni jezik predprocesorja, ki se prevede v kaskadne sloge (CSS). Razširitev datoteke Sass je scss. (Sass, brez datuma)
- Responsive design (skrajšano responsive) je prilagajanje uporabniškega vmesnika na različne velikosti zaslonov.

- SEO: optimizacija spletnih strani (ang. Search Engine Optimization) je proces za izboljšanje vidljivosti na spletnih iskalnikih (Google Search Central, brez datuma).
- DOM: objektni model dokumenta (ang. Document Object Model). Ko se spletna stran naloži, brskalnik ustvari DOM spletne strani. DOM definira HTML elemente kot objekte ter njihove lastnosti, metode in dogodke (ang. events). Natančneje je DOM API za JavaScript in omogoča manipulacijo objektov HTML elementov s funkcijami (ECMAScript 6, brez datuma).
- Navidezni DOM (ang. Virtual DOM): JavaScript predstavitev dejanskega DOMa, ki jo ReactJS prikazuje uporabniku. Posodabljanje navideznega DOMa je hitrejše od dejanskega (ECMAScript 6, brez datuma).
- jQuery: JavaScript knjižnica za lažjo manipulacijo DOM (jQuery.com, brez datuma).
- JSON: Notacija JavaScript objekta (ang. JavaScript Object Notation) je odprti standardni format za izmenjave podatkov na podlagi prenosa podatkovnih objektov, sestavljenih iz parov atributov in vrednosti (JSON, b.d.).
- API: vmesnik za programiranje aplikacij (ang. application programming interface) je posrednik, ki aplikacijam omogoča medsebojno komunikacijo s pomočjo zahtevkov ali klicev (Red Hat, brez datuma).
- REST: predstavitveni prenos stanja (ang. Representational State Transfer) je arhitekturni slog programske opreme, ki omogoča enotno komunikacijo med oddaljenimi odjemalci in strežniki (Code Academy, b.d.).
- API klic: zahtevek, poslan na končno točko APIja. V tem diplomskem delu besedna zveza API klic predstavlja HTTP zahtevek na končno točko spletnega APIja Teleport.
- Končna točka (ang. endpoint): digitalna lokacija za sprejemanje API klicev, ki določa, kje bo API pridobival podatke in kaj bo vrnil v odgovoru (ang. response).
- Teleport API: spletni REST API podjetja Teleport, ki na spletu ponuja nabor končnih točk za pridobivanje podatkov (Teleport public APIs, brez datuma).
- Lokalizacija: postopek adaptacije spletne aplikacije v določeni državi ali regiji. V tem diplomskem delu gre za zagotovitev vsebine v slovenskem in angleškem jeziku.
- SPA: enostranska aplikacija (ang. Single Page Application) je aplikacija, ki uporabniku prikazuje eno spletno stran, na kateri vsebino dinamično prepisuje (SPA, brez datuma).
- TypeScript: microsoftov programski jezik, ki je sintaktična nadmnožica JavaScripta in omogoča statično tipkanje (TypeScript, brez datuma).
- Stanje (ang. state): objekt, ki hrani določene spremenljivke, uporabljene v komponenti.

- Globalno stanje: objekt Redux knjižnice, ki hrani spremenljivke, dostopne vsem komponentam in katerih vrednosti se spreminjajo s proženjem akcij (Getting Started with Redux Toolkit, brez datuma).
- Akcija: redux akcija je objekt s tovorjem (ang. payload), ki se pošilja v globalno stanje.
- px: okrajšava za piksel (ang. pixel), najmanjšo enoto digitalne slike ali grafike, ki je lahko prikazana na napravi z digitalnim zaslonom (techopedia, 2020).

1 UVOD

1.1 Opis področja in opredelitev problema

Raziskava v diplomskem delu se navezuje na področje spletnega razvoja. Za učinkovit razvoj spletnih aplikacij obstaja veliko tehnologij, ogrodij ter pristopov. Ker se na trgu vsako leto pojavljajo nove in bolj učinkovite tehnologije, morajo razvijalci sprejemati odločitve, kdaj je spletne aplikacije smiselno vzdrževati in nadgrajevati na obstoječih tehnologijah in kdaj je bolje izvesti tranzicijo na druge tehnologije. V podjetju, v katerem sem opravljal praktično izobraževanje, je večina projektov bila postavljena na Microsoftovem ASP.NET ogrodju DNN. To ogrodje omogoča razvoj kompleksnih spletnih aplikacij s CMS in zajema podporo za lokalizacijo, sistem uporabniških računov in vlog, administracijo (upravljanje uporabniških računov, SEO (ang. Search Engline Optimization) podatkov in naprednih konfiguracij) in več (Ocvirk, 2010).

Zaradi vseh teh funkcionalnosti so aplikacije v ogrodju DNN večje velikost in kompleksnosti, njihov razvoj in vzdrževanje pa zahtevnejše, kot bi to bilo optimalno za potrebe nekaterih projektov v podjetju. Takšne spletne aplikacije je v nekaterih primerih smiselno prenesti v drugo tehnologijo, ki olajša razvoj in zmanjša stroške vzdrževanja. Primer takšne tehnologije je ReactJS (React, a JavaScript library, b.d.), odprtokodna JavaScript knjižnica za gradnjo uporabniških vmesnikov, katero vzdržuje podjetje Meta (DNN urejevalnik spletnih vsebin, brez datuma). Vprašanje, katerega razrešuje ta raziskava je, kako zahteven je proces tranzicije spletne aplikacije iz DNN na ReactJS in kakšne so pridobitve zadanega procesa. Na podlagi tega želimo oceniti ali je proces smiseln, hkrati pa želimo razdelati, kolikšen del procesa se da izvesti z delom od doma.

1.2 Namen, cilji in osnovne trditve

Namen raziskave je prikazati potrebne korake in potrebna znanja za izvedbo procesa tranzicije spletne aplikacije iz DNN v ReactJS ter analizirati pridobitve, kot so povečana hitrost in odzivnost ter olajšano vzdrževanje, na podlagi česa je možno izpeljati oceno smotrnosti samega procesa glede na spletno aplikacijo in zmogljivosti podjetja. Cilj raziskave je razvijalcem in podjetjem v podobnem položaju olajšati sprejemanje odločitev o tranziciji spletne aplikacije iz DNN v ReactJS.

V raziskavi smo iskali odgovore na naslednja vprašanja:

- Ali je smotrno spletne aplikacije, ki so implementirane v ogrodju DNN implementirat s pomočjo ReactJS knjižnice? Odgovorjeno v četrtem poglavju.
- Zakaj bi bilo smiselno narediti prehod iz DNN na ReactJS? Odgovorjeno v poglavju 1.1.
- Katere DNN funkcije se izgubijo ob prehodu na ReactJS? Odgovorjeno v poglavju 2.6.
- V katerih aplikacijah je smotrno nadomestit DNN z ReactJS? Odgovorjeno v 4.1.
- Kakšni so potrebni koraki, da v spletni aplikaciji nadomestimo ogrodje DNN s knjižnico ReactJS? Odgovorjeno v poglavjih 2.1-2.5.
- Kakšne so prednosti prehoda na ReactJS? Odgovorjeno v poglavju 3.5.
- Kolikšen del zadanega procesa se lahko opravi od doma? Odgovorjeno v poglavju 2.6.

Raziskovalno delo si prizadeva dokazati naslednje hipoteze:

- Spletne aplikacije v ogrodju DNN, ki ne potrebujejo CMS je smotrno implementirati s knjižnico ReactJS (H1). Potrjeno v poglavju 4.1.
- Za prehod iz ogrodja DNN na knjižnico ReactJS je potrebna nova implementacija spletne aplikacije (H2). Potrjeno v poglavju 2.1.
- Spletna stran postavljena s pomočjo knjižnice ReastJS se nalaga in odziva hitreje kot ekvivalentna spletna stran v ogrodju DNN (H3). Potrjeno v poglavju 3.4.
- ReactJS spletno aplikacijo je lažje vzdrževati kot DNN (H4). Potrjeno v poglavju 3.2.2.
- Proces prenosa projekta iz DNN na ReactJS se lahko v celoti opravi od doma (H5). Potrjeno v poglavju 2.6.

1.3 Predpostavke in omejitve

Smotrnost prenosa spletne aplikacije iz DNN na ReactJS se lahko močno razlikuje od aplikacije do aplikacije, saj je proces lahko zahteven, pridobitve pa v nekaterih primerih zanemarljive, če se pri tem izgubi preveč funkcionalnosti, ki jih aplikacija potrebuje. Ker ni izvedljivo razviti vse vrste aplikacij in jih med seboj primerjati, je raziskava postavljena na podlagi spletne aplikacije, ki smo jo izdelali posebej za to diplomsko delo. Ta aplikacija zajema večino različnih funkcionalnosti, kot jih zajema večina spletnih aplikacij na trgu. Te so lokalizacija, dinamičen prikaz podatkov, prejetih iz oddaljenega APIja, responsive design itd. Kot mnoge druge moderne spletne apliakcije tudi naša aplikacija do podatkov dostopa preko klicev na spletni REST API. Druga omejitev se nanaša na dokazljivost hipoteze o lažjem vzdrževanju aplikacije ReactJS kot aplikacije DNN (H4), saj to ni v celoti odvisno od tehnologije, ampak v veliki meri tudi od samega načina pisanja izvorne kode. Da bi zmanjšali vpliv tega, smo pri razvoju aplikacije v DNN in razvoju v ReactJS poskušali upoštevati enaka ali čim bolj podobna načela kodiranja. Zadnja omejitev se nanaša na funkcionalnosti izgubljene ob prehodu na ReactJS. Ker ogrodje DNN zajema ogromno funkcionalnosti, ki jih knjižnica ReactJS ne pokriva, je za učinkovit prenos aplikacije v ReactJS za določene aplikacije potrebno vključit dodatne knjižnice in orodja, kot so i18next (za implementacijo lokalizacije) in Redux (za upravljanje globalnega stanja) (i18next ducumentation).

1.4 Uporabljene raziskovalne metode

Uporabili smo dva raziskovalna pristopa – pregled literature in primerjalno metodo.

1.4.1 Pregled literature

Za vsa ogrodja, knjižnice in programe, uporabljene za izdelavo tega diplomskega dela, obstaja dokumentacija. To je uradna literatura, ki vsebuje opis tehnologije in navodila, kako jo uporabljati. Dokumentacija, ki jo je objavila ekipa DNN (DNN Software, brez datuma), nam je pomagala pri vzpostavitvi projekta v DNN ter razvoju modulov in teme. Pri implementaciji mobilnega menija in responsive designa smo si pomagali z dokumentacijo ogrodja Bootstrap (Grid System, b.d.).

React dokumentacija (React, a JavaScript library, b.d.) nam je pomagala pri implementaciji same ReactJS aplikacije in postavitvi komponent, po navodilih iz dokumentacije i18next (i18next ducumentation, brez datuma) smo implementirali lokalizacijo ter po navodilih iz Redux dokumentacije (Getting Started with React Redux), globalno stanje in akcije za upravljanje z njim.

1.4.2 Primerjalna metoda

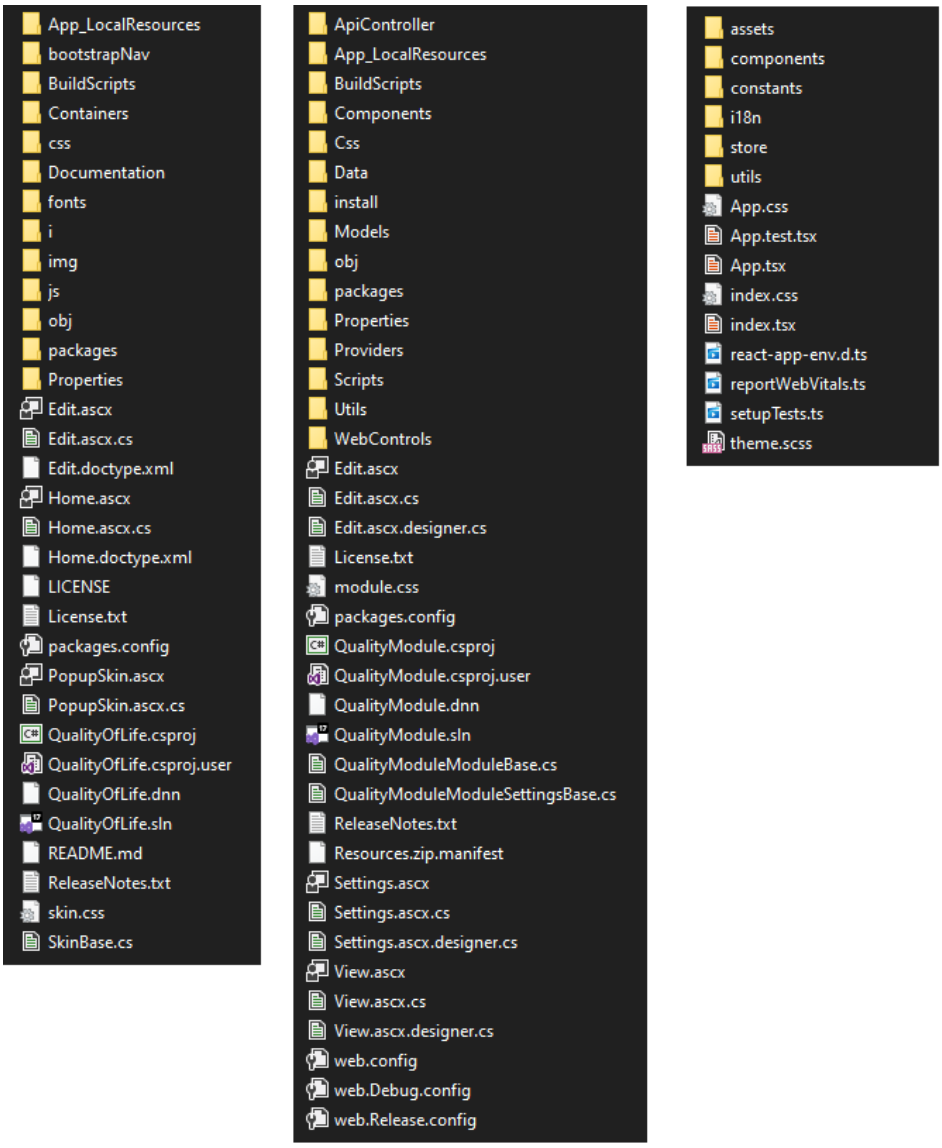
Ključni del za izdelavo ocene smotrnosti procesa tranzicije spletne aplikacije iz DNN v ReactJS je bila primerjava samih aplikacij (DNN urejevalnik spletnih vsebin, brez datuma). Obe spletni aplikaciji smo zagnali v istem okolju in izvajali meritve z istimi orodji. Na ta način smo lahko postavili primerjave različnih dejavnikov, kot so hitrost nalaganja, odzivnost akcij in velikost datotek. Primerjali smo tudi potrebno znanje razvijalcev za obe tehnologiji, vzdrževanje obeh aplikacij ter njune osnovne sestavne dele – komponente in module.

2 PROCES TRANZICIJE IZ ASP.NET DNN NA REACTJS

2.1 Analiza in načrtovanje

Spletna aplikacija, katero si prizadevamo prenesti iz ogrodja DNN na ReactJS, se imenuje Quality of Life (v nadaljevanju QoL). Gre za avtorsko spletno aplikacijo, ki za posamezna urbana okolja prikazuje podrobnosti o ekonomskem položaju, davkih, življenjskih stroških in več, hkrati pa ponuja možnost primerjave mesečnih plač za razne poklice med državami. QoL je sestavljen iz ciljne strani, strani za primerjavo plač in strani za prikaz podrobnosti o urbanih področjih. Spletna aplikacija ima tudi stran za vpis, saj je DNN CMS, ki uporabnikom s pravicami (običajno Administrator in Superuser) omogoča urejanje vsebin in upravljanje z nastavitvami ter spletnimi stranmi preko grafičnega vmesnika (DNN urejevalnik spletnih vsebin, brez datuma). QoL ima na vseh spletnih straneh glavo (ang. header) z logom, navigacijo in gumbom za menjavo jezika ter nogo (ang. footer) s podatki o avtorskih pravicah. Aplikacija je lokalizirana v angleški in slovenski jezik. Spletne strani so s pomočjo ogrodja Bootstrap prilagodljive na večino velikosti zaslonov (ang. responsive design). Vsi podatki o celinah, državah in urbanih okoljih so pridobljeni od javnega spletnega API ponudnika Teleport (Teleport public APIs, brez datuma). Seznami celin, držav in urbanih okolij na pojavnih oknih, seznam kategorij in seznam poklicev ter podrobnosti se na grafičnem vmesniku dinamično generirajo na podlagi pridobljenih podatkov. Za lažjo manipulacijo elementov se uporablja javascript knjižnica jQuery (jQuery.com), ki je že vključena v DNN. Pri razvoju QoL v DNN ni bil uporabljen TypeScript in Sass.

Za tranzicijo spletne aplikacije QoL iz DNN na ReactJS je potrebna nova implementacija, saj se tehnologiji popolnoma razlikujeta v strukturi datotek, procesu razvoja, načinu zagona in v mnogih drugih kritičnih točkah.



Slika 1: Od leve proti desni direktorij DNN teme, direktorij DNN modula in izvorni direktorij ReactJS

Vir: Lastni vir, 2022.

Za razvoj QoL v DNN smo s pomočjo namestitvene datoteke namestili ogrodje. S pomočjo DNN razširitev v MS Visual Studio smo kreirali DNN temo (ang. theme) in QoL modul (ang. module). DNN deluje na način, da razvijalci razvijajo module in teme, administratorji pa potem na uporabniškem vmesniku ustvarjajo spletne strani, jim nastavijo teme in na njih postavljajo module. Podatki o straneh, njihovih temah in modulih postavljenih na njih se zapisujejo v bazo podatkov DNN, med tem ko se v aplikacijah v ReactJS to vse določa v izvorni kodi (React, a JavaScript library, b.d.). QoL na DNN je sestavljen tudi iz HTML modulov (več o HTML modulih je v poglavju 3.1.1, četrti odstavek) katerih vsebina ni zabeležena v datotekah z izvorno kodo, temveč v bazi podatkov.

Aplikacije v ReactJS same po sebi nimajo baz podatkov in uporabniškega vmesnika za upravljanje vsebine ali teme. ReactJS služi samo izgradnji uporabniškega vmesnika v obliki komponent. Fundamentalna razlika je tudi v vrstah datotek z izvorno kodo. DNN omogoča postavljanje HTML elementov v datotekah z razširitvijo ascx. Ta vrsta datoteke poleg pisanja HTML omogoča tudi pisanje kode C# in postavljanje ASP.NET kontrol, kot so Repeater in UpdatePanel. S temi elementi se upravlja v zalednih datotekah (ang. codebehind) z razširitvijo cs, kjer se piše vso poslovno logiko (ang. business logic). Programski jezik C# se pri razvoju spletnih aplikacij z ReactJS ne uporablja. Namesto tega je vsa poslovna logika v JavaScript ali TypeScript datotekah. Komponente in logika zanje se razvijajo v JSX (JavaScript XML) ali TSX (TypeScript različica JSX) datotekah, ki omogočajo pisanje HTML v ReactJS (React, a JavaScript library, b.d.).

Without JSX:

```
const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Slika 2: Ustvarjanje HTML elementov v JavaScript brez JSX.

Vir: (w3schools, brez datuma).

JSX:

```
const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Slika 3: Ustvarjanje HTML elementov v JavaScript z JSX.

Vir: (w3schools, brez datuma).

Razlike v sintaksi in načinu postavitve elementov so med temi datotekami zelo različne. Iz kode za uporabniški vmesnik QoL v DNN lahko uporabimo le preprostejše JavaScript funkcije in do neke mere HTML, vendar ga moramo prilagoditi za ReactJS (React, a JavaScript library, b.d.).

Primer je HTML atribut class, ki je v ReactJS poimenovan className ali dejstvo, da večje strukture HTML elementov zaradi lažje berljivosti in lažjega vzdrževanja v ReactJS razgradimo na manjše ponovno uporabne komponente.

Velika razlika je tudi to, da QoL v DNN za dinamično manipulacijo elementov uporablja jQuery (DNN Software, brez datuma). Ta knjižnica je namenjena manipulaciji objektov HTML elementov v DOMu in na QoL v DNN poskrbi za prikazovanje in skrivanje elementov, dodajanje in odstranjevanje CSS razredov ter dinamičnemu spreminjanju vrednosti vnosnih polj. Na spletu obstaja jQuery različica za ReactJS, vendar za manipulacijo elementov ni potrebna. ReactJS komponente prejmejo vrednosti kot props. Te vrednosti glede na implementacijo vplivajo na prikazane elemente in ko se katera izmed njih spremeni, se v brskalniku posodobi le ta komponenta. Komponente imajo nekatere vrednosti zapisane tudi v svojem stanju in ko se te spremenijo, se ponovno posodobi dotični del uporabniškega vmesnika. Med tem ko bi z uporabo jQuery določenemu elementu direktno dodelili ali odvzeli CSS razred v DOM, bi v ReactJS samo spremenili vrednost spremenljivke (propa ali spremenljivke v stanju), ki pogojuje dodelitev tega razreda in posledično bi se v virtualnem DOMu element posodobil (jQuery.com).

ReactJS aplikacijo se lahko kreira na več načinov. Glede na potrebe zadanega projekta smo izbrali uporabo orodne verige (ang. toolchain) Create React App, ki generira razvojno okolje s preprosto strukturo datotek, s potrebnimi knjižnicami in vgrajenimi ukazi za zagon aplikacije in za kreiranje optimizirane produkcijske verzije. Za lažje vzdrževanje in manjšo možnost napak med razvojem smo se odločili za uporabo TypeScript in Sass (Sass, brez datuma). Lokalizacija bo urejena z ogrodjem za internacionalizacijo imenovanim i18next (i18next ducumentation, brez datuma). Ker bodo nekateri podatki, kot izbran jezik, vplivali na vse komponente, bo aplikacija uporabljala predvidljiv vsebnik (ang. container) stanja imenovan Redux-Toolkit, ki omogoča globalno stanje, ki je dostopno vsem komponentam in orodja za upravljanje s tem stanjem (Getting Started with React Redux, brez datuma). Http zahtevki na Teleport API bodo kreirani s pomočjo knjižnice Axios. Posamezne spletne strani bodo kreirane v obliki ReactJS komponent, navigacija med temi komponentami in URLji v brskalniku bodo urejeni s standardno knjižnico React Router (Rapidapi, brez datuma). Vsebine na posameznih straneh bodo prav tako kreirane kot samostojne in ponovno uporabne (reusable) komponente. Ker aplikacije v ogrodju ReactJS ne vsebujejo CMS, QoL na ogrodju ReactJS ne bo imel grafičnega vmesnika, ki uporabnikom omogoča urejanje vsebin in nastavitev. Iz tega razloga spletna aplikacija ne bo imela strani za vpis.

Za usklajen videz in prilagoditev spletne aplikacije na različne velikosti zaslonov QoL na DNN uporablja Bootstrap, ReactJS pa nadomestek imenovan React Bootstrap, vendar je razlika med njima velika in bo za našo implementacijo bolje uporabiti enako Bootstrap knjižnico kot na DNN (Grid System, b.d.). Za upravljanje različic bomo uporabili spletni gostiteljski servis za odlagališča (ang. repositories) GitHub, ki omogoča odlaganje kode na različne veje (ang. branches) oddaljenih odlagališč, varnostne kopije in nenazadnje dober pregled nad zgodovino sprememb posameznih datotek.

2.2 Vzpostavitev okolja

Na operacijskem sistemu MS Windows 10 smo namestili integrirano razvojno okolje MS Visual Studio Code (vscode), ki je namenjeno razvoju in razhroščevanju spletnih aplikacij, ima podporo za Git in ima na razpolago ogromno število razširitev. Hkrati smo namestili program GitHub Desktop, ki ima grafični vmesnik za lažje upravljanje GitHuba. V GitHub Desktop smo ustvarili nov oddaljen repozitorij in izbrali lokacijo na disku za lokalni repozitorij. Pogoj za kreiranje ReactJS aplikacije z orodno verigo Create React App je bila namestitev upravitelja JavaScript paketov npm (Node Package Manager), in Node.js, ki je medplatformsko izvajalno okolje JavaScript in omogoča izvajanje JavaScript kode zunaj brskalnika. V vscode smo namestili razširitev imenovano Prettier, urejevalnik kode, ki samodejno popravlja razmike in prelome v vrsticah in s tem zagotavlja enoten stil programske kode. Naslednji korak je bila kreacija spletne aplikacije. V vscode smo odprli terminal (powershell) in pognali ukaz:

```
npx create-react-app quality-of-life --template typescript
```

Ta ukaz je generiral vse potrebne datoteke in direktorije za ReactJS aplikacijo z imenom quality-of-life in s podporo za TypeScript. Zadnji korak pri vzpostavitvi razvojnega okolja je bila namestitev orodja za internacionalizacijo i18next, knjižnice axios za http zahtevke, CSS razširitve imenovane Sass (Super Awesome Style Sheets) in Redux za upravljanje z globalnim stanjem. Za namestitev smo v terminalu pognali ukaze:

- za i18next:

```
npm install react-i18next i18next --save
```

- za axios:

```
npm install axios
```

- za Sass:

```
npm install sass
```

- za React Redux:

```
npm install react-redux
```

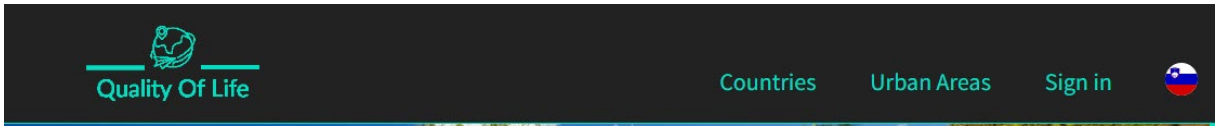
2.3 Kreiranje spletnih strani in vsebin v obliki komponent

Orodna veriga Create React App nam je ustvarila SPA, kar pomeni, da se mora naložiti le ena spletna stran, na kateri se vsebine dinamično prepisujejo. Da ima ReactJS spletna aplikacija videz večih spletnih strani, smo morali za vsako stran ustvariti posamezno komponento in za vsako svojo datoteko scss (SPA, brez datuma). Komponente so lahko razredne ali funkcijske (ang. class components, functional component). Pri naših komponentah smo se odločili za funkcijske, saj smo po preučevanju literature (Cleary, 2020) odločili, da je to najboljši pristop. Povezovanje strani z URL-ji se je uredilo s komponento React-Router. To se je doseglo na tak način, da smo v datoteki index.tsx ovili komponento App v komponento BrowserRouter, nakar smo kreirali svojo komponento imenovano Router, ki vsebuje seznam vseh poti (ang. routes) in komponente, ki predstavljajo spletne strani. To komponento smo potem postavili v telo komponente App. Ker imajo vse strani na QoL enako glavo in nogo, smo ustvarili komponento, ki služi kot vsebnik za preostale strani, da bosta glava in noga samo tam in ne na treh mestih. To komponento smo poimenovali PageBody in jo postavili v telo App komponente na način, da smo z njo ovili prej postavljeno Router komponento. Zaradi takšne postavitve PageBody prejme Router komponento kot prop in jo prikaže ovito v svojo kodo. S tem smo dosegli, da se ob vpisani poti v brskalnik na aplikaciji prikaže željena komponenta ovita s PageBody.

2.3.1 Vsebnik strani

Prej omenjena komponenta PageBody vsebuje le glavo, nogo in prirejen drsnik za telo, saj je le to enako na vseh straneh, vsebino strani oz. komponente, ki predstavljajo strani, pa prejme kot prop in jo prikaže v svojem telesu.

Prvi korak je bil, da smo implementirali novo komponento za glavo strani. Ta je sestavljena iz logotipa, navigacije in gumba za menjavo jezika. Logotip smo implementirali kot preprosto sliko z event listenerjem za klik, ki uporabnika pelje na domačo stran. Navigacija je komponenta, znotraj katere smo kreirali sliko zastavice ovito v gumb za spreminjanje jezika, ki ob kliku spremeni stanje (ang. state) komponente za glavo in s tem spremeni ikono, ki prikazuje zastavo za izbor jezika. Poleg tega gumba smo ustvarili v obliki komponent še dva gumba, ki prikažeta stran z državami ali stran z urbanimi področji. Glavo smo s CSS oblikovali, da se ujema z QoL na DNN ter uredili responsive (prikaz na različnih velikostih zaslonov). To med drugim pomeni tudi implementacijo drugačnega prikaza navigacije v obliki dropdown menija na mobilnih zaslonih (DNN Software). To smo dosegli s preurejanjem obstoječih elementov s CSSjem.

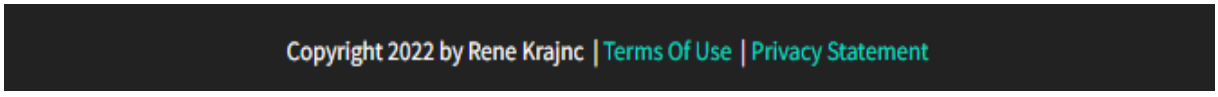


Slika 4: Glava spletne aplikacije QoL.

Vir: Lastni vir, 2022.

Komponenta za nogo strani je preprosta in sestavljena iz osnovnih HTML elementov. Vsebuje ime in priimek avtorja ter URL-je do pogojev uporabe in izjave o zasebnosti.

Potem, ko smo implementirali glavo in nogo strani, je bilo potrebno implementirat posebni drsnik prilagojen na način, da sta glava in noga vedno vidni. Nazadnje smo prilagodili glavo, da se z animacijo zmanjša, če je zaslon manjši od 991px ali uporabnik podrsa telo strani vsaj 40px od vrha.

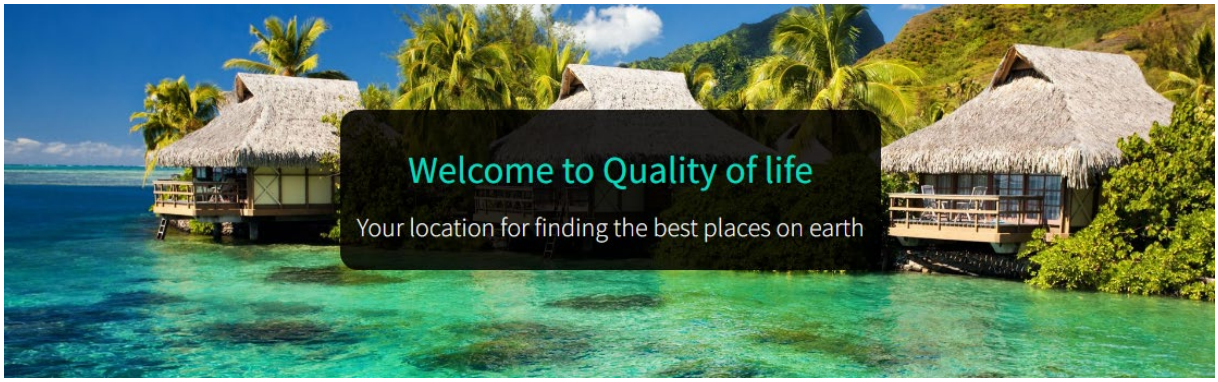


Slika 5: Noga spletne aplikacije QoL.

Vir: Lastni vir, 2022.

2.3.2 Ciljna stran

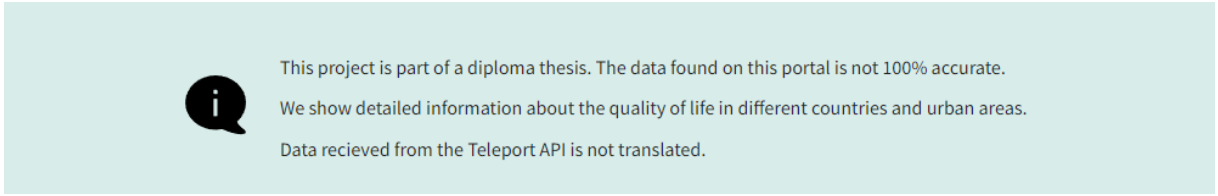
Ciljna oziroma domača stran na QoL prikazuje podatke o spletni aplikaciji in o njenem avtorju. Po analizi strukture spletne strani smo naredili načrt, katere elemente bodo zajemale posamezne komponente. Prva komponenta, ki smo jo implementirali, je bila naslovna pasica z glavnim naslovom, podnaslovom in s sliko za ozadje.



Slika 6: Naslovna pasica na domači strani QoL.

Vir: Lastni vir, 2022.

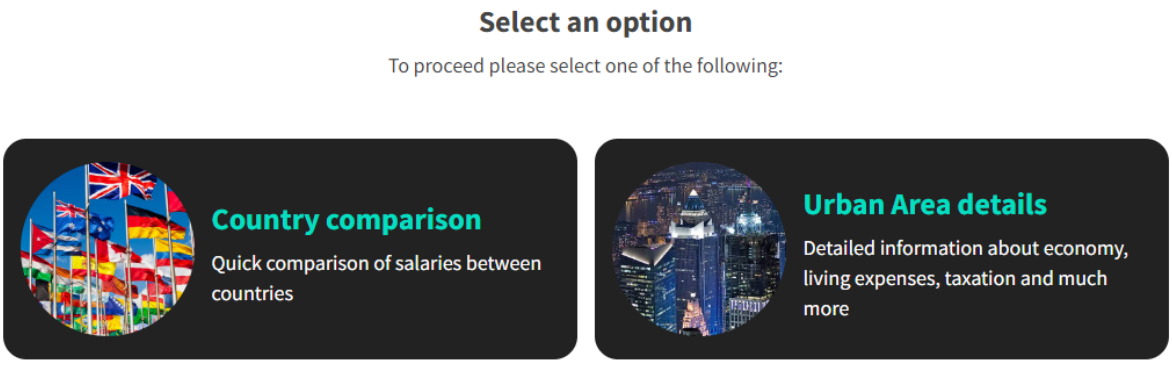
Ker bo ta komponenta uporabljena tudi na ostalih dveh straneh, vendar z drugim besedilom in sliko, smo jo implementirali na način, da naslov, podnaslov in sliko prejme kot props. To pomeni, da lahko komponento postavimo na večih mestih z različnimi podatki.
Naslednja komponenta je obvestilo o tem, da podatki prikazani na tej spletni aplikaciji niso popolnoma natančni in da podatki prejeti preko Teleport API niso prevedeni.



Slika 7: Obvestilo na domači strani QoL.

Vir: Lastni vir, 2022.

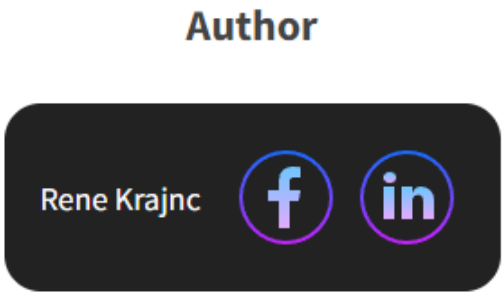
Ta komponenta je sestavljena le iz slike, besedila in barvnega ozadja. Ta komponenta se pojavi le na tej strani, zato nismo uporabili propsov. Pod obvestilom je izbor željene vsebine, ki je sestavljen iz naslova, podnaslova in dveh možnosti v obliki kartic sestavljenih iz slike, naslova in podnaslova. Klik na posamezno sliko ali naslov uporabnika pelje na drugo spletno stran.



Slika 8: Naslov in kartice za navigacijo prikazane na domači strani QoL .

Vir: Lastni vir, 2022.

Naslov in podnaslov smo implementirali kot samostojno komponento, saj se pojavi pod tem sklopom še en naslov, ki je enake oblike. Komponenta vzame za props naslov in podnaslov. Ker je na drugi strani še en naslov drugačne oblike, smo se odločili, da tudi za njega uporabimo enako komponento in med propse dodamo še vrsto naslova (mala ali velika). Za kartice s temnim ozadjem smo ustvarili novo komponento, ki kot props vzame sliko, naslov, opis in pot do željene vsebine ter jo dvakrat postavili na domačo stran, vsakič z drugimi podatki. Zadnji sklop na domači strani so podatki o avtorju, sestavljeni iz naslova in kartice z imenom avtorja ter ikonami s povezavami do njegovih strani na družbenih omrežjih.



Slika 9: Naslov in kartica s povezavami do strani avtorja na družbenih omrežjih.

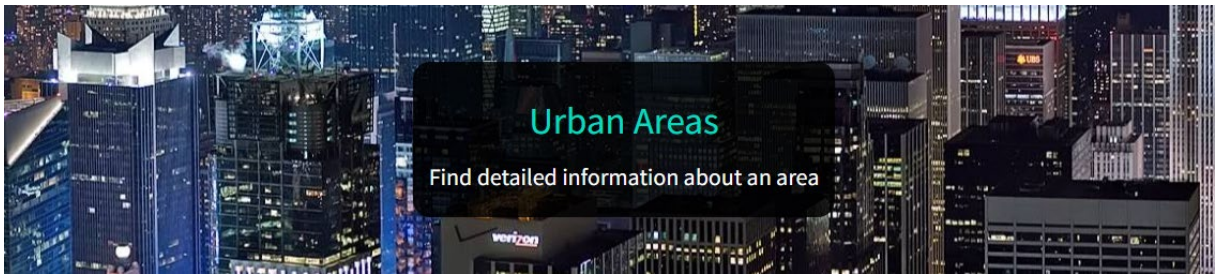
Vir: Lastni vir, 2022.

Za naslov smo postavili komponento, ki smo jo ustvarili v prejšnjem koraku, za kartico smo pa ustvarili novo komponento brez propsov, saj se podatki o avtorju ne bodo spreminjali.

2.3.3 Stran za prikaz podatkov o življenjskem standardu v urbanih okoljih

Stran z urbanimi področji je namenjen prikazu podrobnih podatkov o kvaliteti in stroških življenja za posamezno urbano področje. Uporabniški vmesnik uporabniku omogoča izbor celine in glede na izbrano celino, izbor urbanega področja. Celine in urbana področja uporabnik izbira na pojavnem oknu imenovanem modal. Za izbrano urbano področje se na strani prikaže fotografija področja in tako imenovana harmonika (ang. accordion) kategorij, od katerih vsaka vsebuje tabelo s podrobnimi podatki.

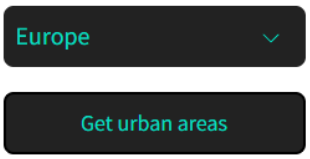
Prvi sklop elementov je podoben kot na domači strani in smo zanj lahko uporabili isto komponento, vendar podali drug naslov, podnaslov in sliko kot props.



Slika 10: Naslovna pasica na strani za prikaz podrobnosti o urbanih področjih.

Vir: Lastni vir, 2022.

Naslednji sklop elementov je uporabniški vmesnik za izbor urbanega področja in prikaz podrobnosti. Ker gre za bolj zapleten uporabniški vmesnik, smo ga implementirali kot komponento sestavljeno iz več manjših komponent. Prva od teh komponent je vmesnik za izbor urbanega področja.



Slika 11: Vnosno polje za izbor celine in gumb za pridobitev seznama urbanih področij.

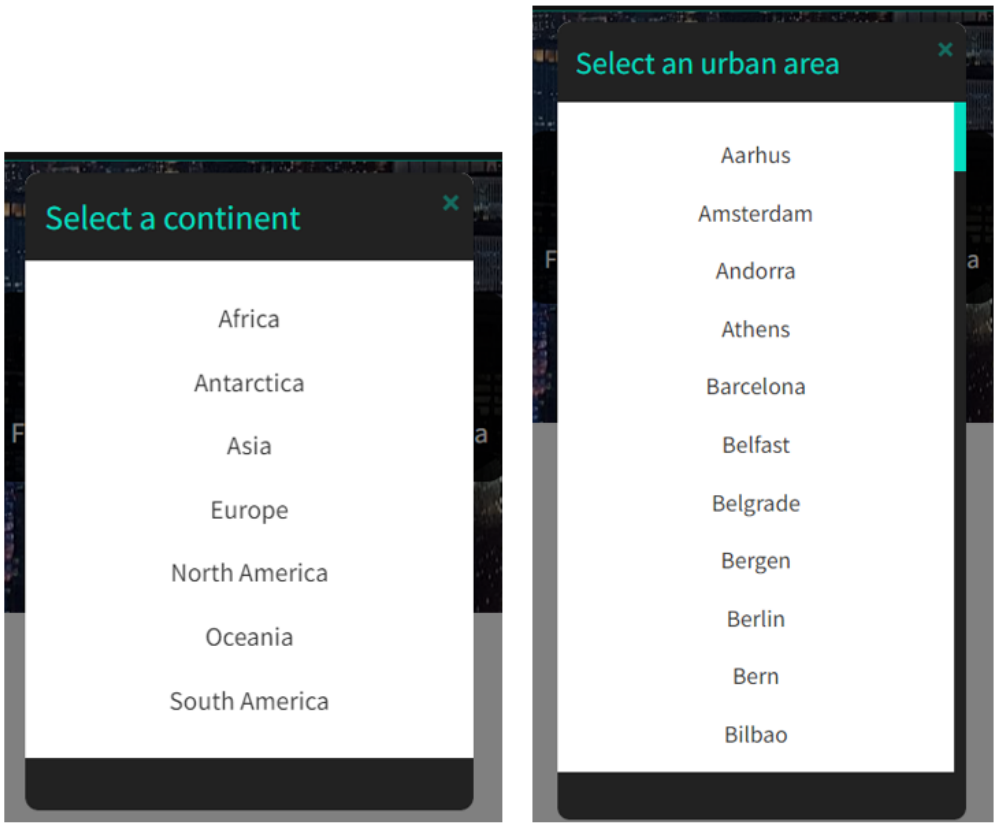
Vir: Lastni vir, 2022.

Za ta sklop smo razvili dve novi komponenti. Ena je za vnosno polje, ki ob kliku prikaže modal s seznamom celin ali urbanih področij ter spremeni prikazano besedilo glede na izbrano lokacijo, druga pa je preprost gumb. Komponento z vnosnim poljem smo postavili dvakrat – prvič za izbor celine in drugič za izbor urbanega področja. Tudi gumba smo postavili dva, prvega za pridobitev seznama urbanih področij in drugega za pridobitev podrobnosti o izbranem urbanem področju.

Slika 12: Potrjeno vnosno polje za izbor celine, vnosno polje za izbor urbanega področja in gumb za pridobitev podrobnosti.

Vir: Lastni vir, 2022.

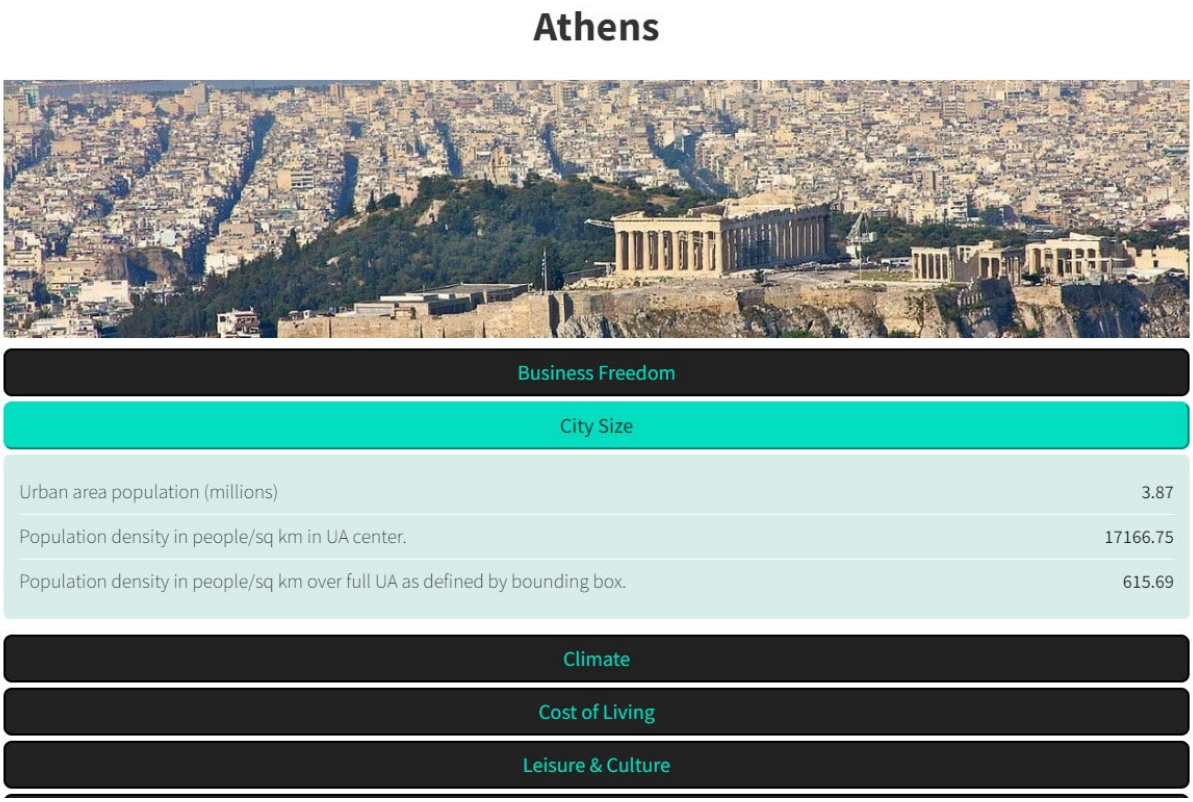
Ker bodo podatki nastavljenih v teh dveh vnosnih poljih prišli iz pojavnega okna, ki bo ločena komponenta, in vplivali na podrobnosti, ki bodo prav tako ločena komponenta, smo se odločili, da se bodo podatki o izbrani celini in državi ter o njunih končnih točkah (ang. endpoints) zapisali v globalno stanje in se od tam brali. Globalno stanje smo ustvarili že za namene lokalizacije in je opisano v poglavju 2.4.1 Redux in Redux Toolkit (Getting Started with React Redux). Naslednji korak je bila implementacija pojavnega okna za izbor celin in za izbor urbanih področij.



Slika 13: Pojavno okno QoL, levo za izbor celine in desno za izbor urbanega področja.

Vir: Lastni vir, 2022.

Pojavno okno smo implementirali kot ločeno komponento na način, da le-ta prejme podatke o tem, kakšne lokacije prikazovati iz globalnega stanja. Vsakič, ko se proži logika za odpiranje pojavnega okna, se sproži akcija, ki v globalno stanje zapiše potrebne podatke kot seznam celin, njihove končne točke in podatek, kam naj se izbrana lokacija zapiše. Pojavno okno smo razširili še za potrebe primerjave plač med državami. Ker so na pojavnem oknu prikazani seznami celin, držav ali urbanih področij pridobljenih iz Teleport APIja, smo kot naslednje implementirali logiko za klice na APIje in za zapisovanje pridobljenih podatkov v globalno stanje. Proces je opisan v poglavju 2.5. Po tem ko smo implementirali še logiko za prikazovanje in skrivanje posameznih elementov z animacijami, kot imamo na QoL DNN, smo ustvarili novo komponento za prikaz podrobnosti urbanega področja.



Slika 14: Prikaz podrobnosti o urbanem področju na QoL.

Vir: Lastni vir, 2022.

Ta komponenta je sestavljena iz naslova, slike in seznama kategorij s podrobnimi podatki v obliki harmonike. Kot props prejme ime, sliko in seznam kategorij izbranega urbanega področja. Kategorije smo implementirali na način, da se prikazujejo dinamično, saj jih prejmemo iz Teleport APIja in lahko katera za določeno urbano okolje manjka ali se v prihodnosti preimenuje. Komponente za kategorije kot props prejemajo naslov ter seznam posameznih vnosov z vrednostmi.

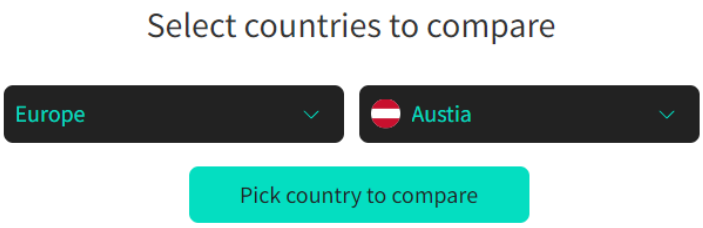
2.3.4 Stran za primerjavo plač med državami

Za prvi sklop elementov smo uporabili enako komponento kot na domači strani in strani z urbanimi področji, vendar ponovno z drugimi propsi.

Slika 15: Naslovna pasica na strani za primerjavo plač med državami.

Vir: Lastni vir, 2022.

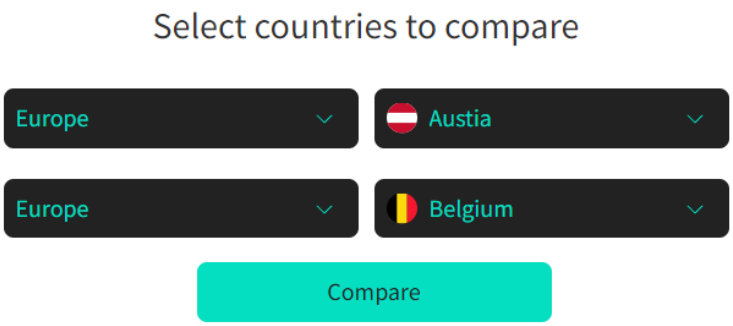
Naslednji sklop elementov je izbirnik držav, ki je podoben izbirniku urbanih področij. Na strani je vnosno polje za izbor celin, ki odpre pojavno okno s seznamom le-teh. Ob potrditvi celine se prikaže vnosno polje za državo, ki odpre pojavno okno s seznamom držav.



Slika 16: Izbor prve celine in prve države za primerjavo.

Vir: Lastni vir, 2022.

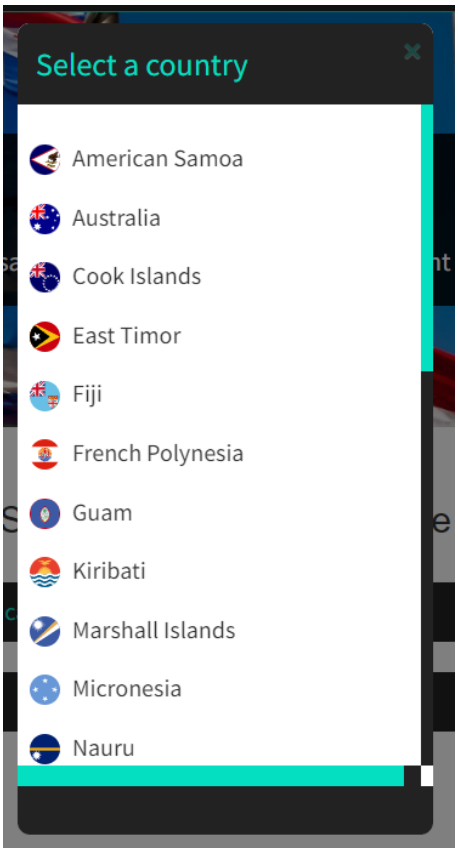
Razlika od urbanih področij je v tem, da se ob potrditvi države ne prikažejo podrobnosti, ampak novo vnosno polje za celino ter po potrditvi še vnosno polje za državo s katero želimo narediti primerjavo.



Slika 17: Izbor obeh celin in držav za primerjavo.

Vir: Lastni vir, 2022.

Izbirnik držav se od izbirnika urbanih področij razlikuje tudi v tem, da ima naslov. Komponento za naslove smo že implementirali in smo jo zato morali le postaviti z drugačnim besedilom in drugačnim stilom definiranim v propsih. Kot naslednje smo v komponenti, ki predstavlja izbirnik, postavili vnosna polja za prvo in drugo državo, vnosna polja za prvo in drugo celino ter gumbe za potrditve le-teh. Sledila je implementacija logike za prikazovanje in skrivanje posameznih vnosnih polj in gumbov, logike za prikazovanje pojavnega okna ter logike zapisovanja izbranih podatkov v globalno stanje. Naslednji korak je bila prilagoditev komponente za pojavno okno tako, da lahko prikaže seznam držav ter ob vsaki državi sliko zastave.

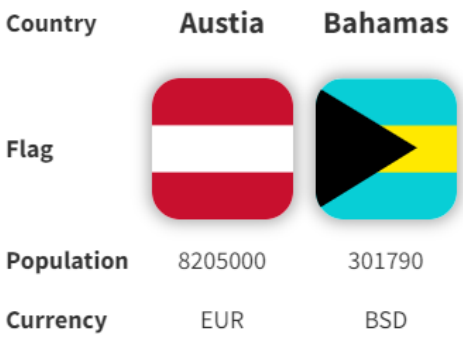


Slika 18: Pojavno okno za izbor države.

Vir: Lastni vir, 2022.

Iz direktorija QoL DNN smo prekopirali slike zastav ter implementirali logiko za njihov prikaz. Zastave so prikazane tudi v vnosnem polju za izbor države, v kolikor je ta izbrana. Iz tega razloga smo razširili komponento za vnosna polja z opcijskim propom za kratico države, na podlagi katere najdemo pravo ikono.

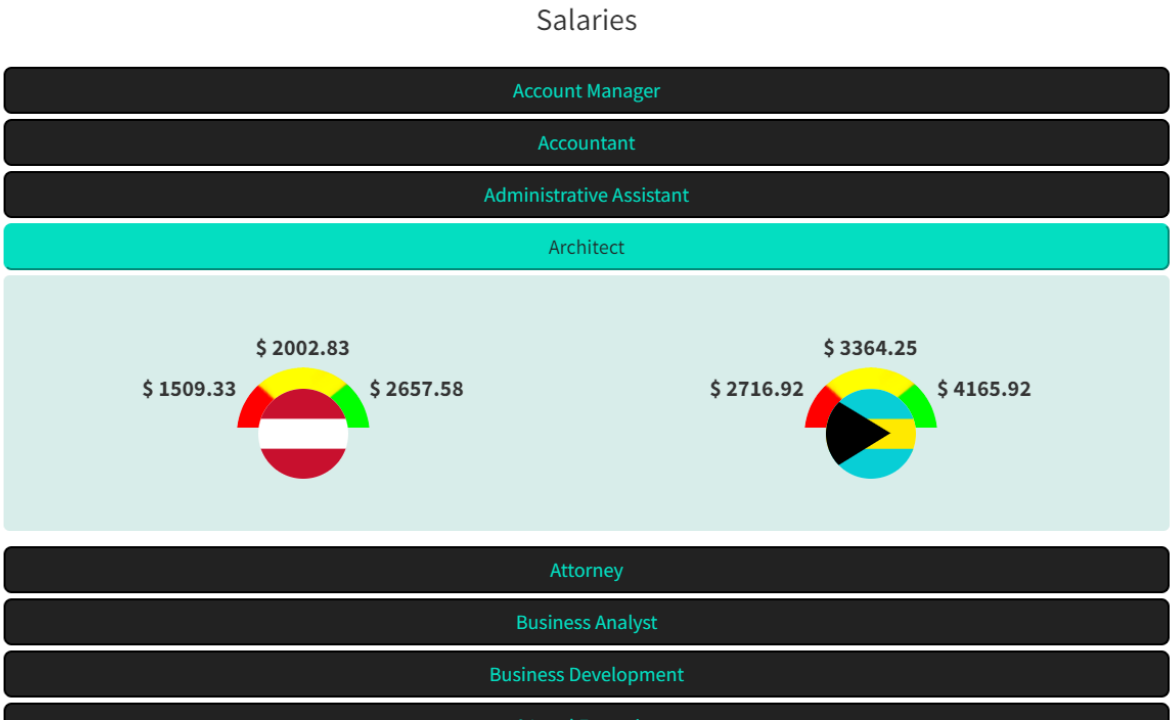
Naslednji sklop je splošna primerjava izbranih držav, ki prikazuje imeni obeh držav, sliki zastav, podatke o številu prebivalcev ter podatke o valuti.



Slika 19: Splošna primerjava držav v spletni aplikaciji QoL.

Vir: Lastni vir, 2022.

Splošno primerjavo smo implementirali kot preprosto komponento, vse potrebne podatke prejme kot props v obliki dveh objektov – enega za vsako državo in jih prikaže v tabeli. Podatke za vsako državo smo pridobili z API klicem na končno točko za pridobivanje podrobnosti o državah. Klice smo implementirali na tak način, da so zaporedni in da je se klic za drugo državo izvede le, če je klic za prvo bil uspešen. Prejete podatke za obe državi smo v stanje komponente zapisali istočasno. V nadaljevanju smo implementirali še klice za pridobivanje seznama plač za različne službe znotraj izbranih držav. Ponovno gre za dva zaporedna klica, kjer je drugi pogojen z uspešnostjo prvega in podatke za obe državi istočasno zapišejo v stanje komponente. Zadnji sklop na strani za primerjavo plač je primerjava plač sama. Ta je podobno kot na strani urbanih področij v obliki harmonike.



Slika 20: Primerjava mesečnih plač na spletni aplikaciji QoL.

Vir: Lastni vir, 2022.

Prvi korak je bil implementacija nove komponente, ki prejme seznam poklicev s podatki o podpovprečnih, povprečnih in nadpovprečnih plačah za obe državi. Seznama služb in plač za obe državi, ki smo ju prejeli od Teleport API, smo združili v en seznam in vsaki državi dodali podatek o kratici, da smo lažje generirali poklice in prikazali zastave v uporabniškem vmesniku. Znotraj te nove komponente smo postavili našo komponento za naslov. Harmoniko smo implementirali na podoben način kot za urbana področja, vendar so podatki za prikaz in uporabniški vmesnik do te mere različni, da nismo uporabili teh komponent, temveč smo razvili nove. Prva komponenta zajema gumb za posamezni poklic in vsebnik za drugo komponento, ki je postavljena dvakrat, vsakič za eno izbrano državo, in prikazuje podatke o plačah ter zastavo. Sledili so le še manjši popravki in s tem smo razvili in postavili vse potrebne komponente.

2.4 Lokalizacija

Za implementacijo lokalizacije smo se odločili uporabiti internacionalizacijsko ogrodje i18next (i18next ducumentation, brez datuma), ki omogoča implementacijo večjezičnih spletnih aplikacij z uporabo prevodov zapisanih v formatu JSON (JSON, b.d.). Uporabnik bo jezik izbiral s klikom na zastavico v glavi strani. Podatek, kateri jezik je izbran, je do sedaj bil zapisan le v stanju komponente za glavo, kar ni dovolj, saj podatek o izbranem jeziku vpliva na večino komponent v aplikaciji. Za reševanje tega problema smo uporabili Redux in Redux Toolkit (Getting Started with React Redux, brez datuma) in (Getting Started with React Redux).

2.4.1 Redux in Redux Toolkit

Redux je odprtokodna knjižnica za upravljanje in centralizacijo stanja, ki omogoča globalno stanje dostopno povsod v aplikaciji. Dodatno smo namestili še Redux-Toolkit, ki je razširitev Reduxa in omogoča hitrejši in preglednejši razvoj (Getting Started with Redux Toolkit). To smo storili z ukazom:

```
npm install @reduxjs/toolkit
```

Za vzpostavitev globalnega stanja je bilo potrebno ustvariti objekt, ki predstavlja stanje in ima lastnosti s prvotnimi vrednostmi. To je inicialno stanje. Redux vzame to stanje in iz njega s pomočjo akcij imenovanih reducer ustvari in vrne novo spremenjeno stanje. Zato je bil naslednji korak kreiranje in izvoz reducerja za spremembo jezika. Da smo globalno stanje povezali s samo aplikacijo, smo morali v Index.tsx datoteki oviti App in RouterBrowser komponenti v Provider komponento od Reduxa in ji kot prop dodeliti izvožen objekt globalnega stanja. Zadnji korak je bil, da smo ustvarili kavlje (hooks), s pomočjo katerih bomo lahko klicali reducer za spremembo jezika ali pridobili podatek o izbranem jeziku kjerkoli v aplikaciji (Getting Started with React Redux).

2.4.2 i18next

Za implementacijo večjezičnosti smo se odločili za knjižnico i18next, katero smo namestili že pri vzpostavitvi okolja. Slovenska besedila smo iz QoL na DNN kopirali v JSON datoteko translations-si.json, jih razdelili v sklope po straneh ter vsakemu dodelili ključ. Enako smo storili z angleškimi prevodi, vendar smo jih dali v datoteko translations-en.json. V istem direktoriju smo ustvarili konfiguracijsko datoteko config.ts, v kateri smo opredelili katera JSON datoteka je za kateri jezik in iniciirali knjižnico (JSON, b.d.).
Kjer bi sicer bila besedila, se sedaj kliče i18next kavelj (ang. hook) useTranslation, ki vzame ključ prevoda kot argument in vrne besedilo v izbranem jeziku (i18next ducumentation).

2.5 Axios

Axios je knjižnica za lažje izvajanje klicev na oddaljene APIje (API explorer, brez datuma). V direktoriju utils smo ustvarili novo TypeScript datoteko imenovano axios, v katero smo uvozli samo knjižnico. S pomočjo knjižnice smo ustvarili asinhrono funkcijo, ki kliče Teleport API na končno točko za pridobivanje celin in logiko, ki iz rezultata izlušči seznam celin in ga vrne v pravilnem željenem tipu. Celine se potem zapišejo v globalno stanje. Takšne funkcije smo ustvarili še za pridobivanje seznama urbanih področij glede na celino, seznama držav glede na celino, podrobnosti o izbranem urbanem področju in seznam plač glede na izbrano državo. Na tej točki smo prilagodili tudi logiko pojavnega okna. Seznam lokacij, ki jih pojavno okno prikazuje, je bil odvisen od seznama, določenega v globalnem stanju v objektu pojavnega okna. Ker smo v globalno stanje po novem zapisovali vse vrste lokacij, smo morali prilagoditi akcijo za odpiranje pojavnega okna. Kot argument v akciji smo namesto seznama lokacij poslali vrsto lokacij in prilagodili vir, od kod pojavno okno črpa seznam glede na izbrano vrsto lokacij.

2.6 Ugotovitve

Za tranzicijo spletne aplikacije QoL iz DNN v ReactJS je bila potrebna nova implementacija, saj so razlike med tehnologijami prevelike za druge vrste tranzicij.

DNN je CMS, zato so se v tranziciji določene funkcionalnosti neizogibno izgubile. Večino teh funkcionalnosti ni bilo potrebno nadomestiti, ker v QoL niso uporabljene. Med najbolj opazne spadajo:

- uporabniški vmesnik za administracijo (zajema uporabniški vmesnik za dodajanje, urejanje in odstranjevanje spletnih strani, postavitev, urejanje in odstranjevanje modulov, upravljanje z uporabniki, uporabniškimi vlogami in pravicami, upravljanje naprednih konfiguracij);
- podpora za uporabniške račune (zajema registracijo novih uporabniških računov, upravljanje s profilom, kot je dodajanje profilne slike in sprememba prikaznega imena, sprememba gesla, obnovitev izgubljenega gesla, klepet med uporabniki, avtomatizirano pošiljanje povratnih elektronskih sporočil ob registraciji, spremembo gesla ipd.);
- lokalizacija – DNN omogoča preprosto vključitev lokalizacije in vsebuje pakete najpogostejših jezikov in možnost dodajanja novih. V QoL v ReactJS smo morali lokalizacijo implementirati s pomočjo i18next;
- DNN trgovina (ang. DNN Store), ki ponuja velik nabor kompleksnih modulov, kot so v celoti nastavljive spletne trgovine in velik izbor tem.

Celotni proces tranzicije smo opravili z delom od doma. Vsa potrebna dokumentacija je bila dosegljiva na spletu. Razvoj je v celoti potekal na računalniku doma. Datoteke z izvorno kodo smo nalagali na oddaljen GitHub repozitorij na spletu. To je služilo večim namenom – kodo smo si lahko delili na daljavo, imeli smo vpogled v zgodovino sprememb in imeli smo večjo varnost, saj v primeru okvare računalnika ne bi izgubili napredka.

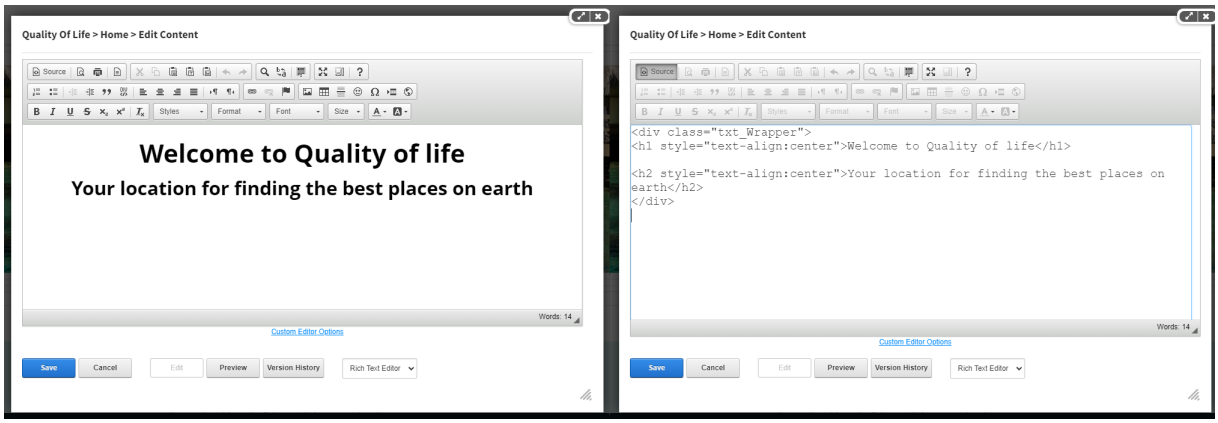
3 PRIMERJAVA SPLETNIH APLIKACIJ DNN IN REACTJS

3.1 Komponente in moduli

Komponente so osnovni gradniki spletnih aplikacij v ReactJS, moduli pa osnovni gradniki aplikacij v DNN. Razvijalci lahko oboje razvijajo po lastnih potrebah ali izbirajo med širokim naborom obstoječih. Tako komponente kot moduli lahko vsebujejo HTML elemente in logiko ter so lahko samostojne celote brez odvisnosti. Kljub tem skupnim točkam se ReactJS komponente močno razlikujejo od DNN modulov (React, a JavaScript library, b.d.). Nastavitve in vsebino DNN modulov lahko spreminja oseba, ki ni spletni razvijalec, saj ima na voljo za to namenjen uporabniški vmesnik. Uporabnik s pravo uporabniško vlogo lahko module tudi dodaja, odstranjuje in prestavlja. Komponente v ReactJS lahko postavlja, spreminja in prestavlja le razvijalec, ki ima dostop do izvorne kode, saj ReactJS ni CMS. Med tem ko se DNN moduli na spletno stran postavljajo z miško (ang. drag and drop), se komponente ReactJS postavljajo v HTML predel App komponente ali katere druge komponente, in sicer v sintaksi HTML elementa. Atributi komponente se imenujejo props in se uporabljajo za podajanje podatkov in nastavitev komponenti.

3.1.1 HTML moduli

V spletni aplikaciji QoL v DNN smo za razvoj preprostih predstavitvenih sklopov uporabili HTML modul. To je eden od modulov, ki so privzeto vključeni v DNN in jih ni potrebno razvijati ali pridobivati iz zunanjih virov.

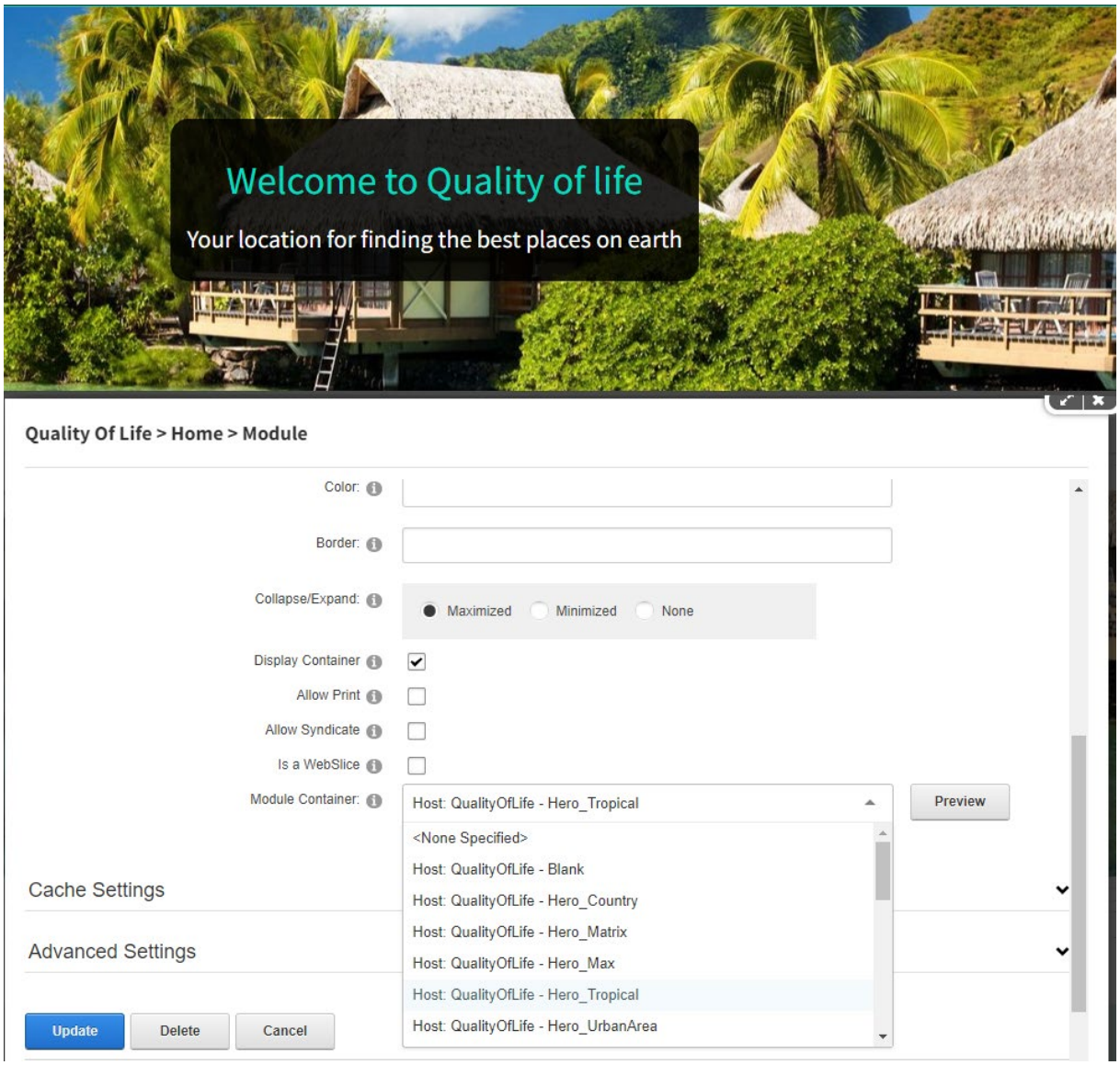


Slika 21: DNN urejevalnik HTML modula - levo urejevalnik besedila, desno urejevanik kode HTML.

Vir: Lastni vir, 2022.

HTML modul ob postavitvi vsebuje le HTML element za odstavek z navodilom, kako odpreti urejanje komponente (ang. Select "Edit Content" from the Edit (Pencil Icon) Action Menu). Uporabniški vmesnik za urejanje HTML modula je bogat urejevalnik besedila z možnostmi spreminjanja stila pisave, velikosti besedila, poravnave, dodajanja slik, tabel, točkovnih ali številčenih seznamov itd. Urejevalnik omogoča tudi možnost urejanja vsebine v HTML, kar nam je omogočilo, da smo po potrebi HTML elementom dodali CSS razrede. Stil za vse HTML module smo definirali v CSS datoteki teme, ki smo jo razvili za QoL. Večjezičnost na DNN je implementirana na način, da se za spletno stran za katero želimo, da je večjezična, ustvari dodatna spletna stran, ki predstavlja lokalizirano različico. To pomeni, da sta na primer angleška in slovenska domača stran v DNN dve različni spletni strani, ki imata različni URL, različne nastavitve in lahko imata postavljene različne module. Module postavljene na angleški strani smo morali na novo postaviti na slovenski strani in spremeniti besedilo v slovenski jezik. Če bi želeli odstraniti en določen modul ali ga spremeniti, ga je v primeru HTML modula potrebno na vsaki jezikovni različici spletne strani posebej. V urejevalniku HTML kode lahko v elementu script pišemo tudi JavaScript, vendar smo v primeru QoL večino JavaScript kode napisali v JavaScript datoteko, ki se nahaja v direktoriju teme.

DNN omogoča možnost vsebnikov za module, ki jih lahko razvijalci razvijajo sami. Za QoL v DNN smo razvili vsebnike, ki so namenjeni prikazovanju ozadja za modul.



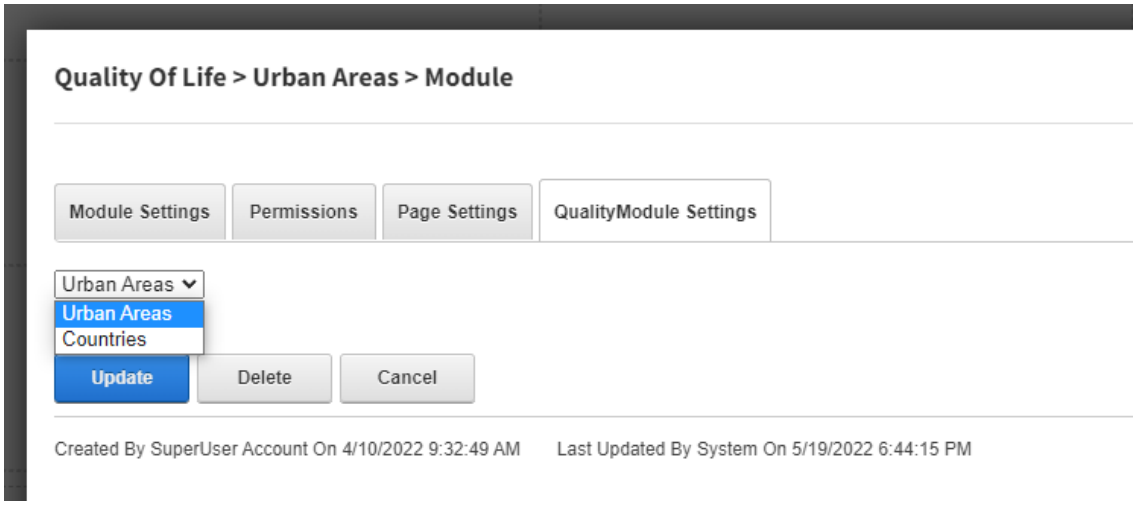
Slika 22: DNN urejevalnik nastavitev modula - izbor vsebnikov za module.

Vir: Lastni vir, 2022.

Prednost pristopa z vsebniki je ta, da jih lahko administrator nastavlja katerem koli modulu in si s tem olajša kreiranje novih modulov v enakem ali podobnem slogu kot obstoječe.

3.1.2 Quality of Life modul

QoL vsebuje dva bolj zapletena sklopa uporabniškega vmesnika – prikaz podrobnosti o urbanih področjih in primerjava plač med državami. Oba sklopa glede na vnosne podatke pošiljata zahtevke na spletni API in glede na odgovore dinamično generirata elemente. Iz tega razloga smo razvili svoj DNN modul. Znotraj modula smo razvili dve uporabniški kontroli (ang. User Control) in prilagodili modul na način, da lahko administrator v uporabniškem vmesniku izbira katero uporabniško kontrolo bo prikazoval.



Slika 23: Izbor uporabniške kontrole za prikaz v QoL modulu.

Vir: Lastni vir, 2022.

Ta pristop nam je omogočil, da smo lahko na spletno stran za primerjavo plač in spletno stran za podrobnosti urbanih področij postavili enak modul in mu le spremenili nastavitve, da smo prikazali pravi uporabniški vmesnik. Uporabniške kontrole v DNN modulih so sestavljene iz ascx in C# datoteke, zaradi česar smo lahko klice na spletni API, formatiranje odgovorov, prikazovanje vsebine in ostalo logiko določili v zalednih datotekah. V ascx datotekah smo postavili HTML in ASP.NET kontrole. Vsaki uporabniški kontroli smo ustvarili tudi JavaScript datoteko, vendar smo morali nekaj JavaScript kode napisati tudi v ascx datotekah, saj imamo v njih dostop do zaledja in C# metod, kot je metoda za prevajanje. Za razliko od HTML modulov, kjer je vsebina definirana v vsakem modulu posebej, so besedila in prevodi v QoL modulu zbrani v slovenski in angleški resx datoteki (ang. application resource file), od koder se po načelu ključ – vrednost pridobivajo z C# metodo glede na trenutno izbran jezik.

Posledično lahko postavimo QoL modul na slovensko in angleško spletno stran in bo glede na izbran jezik prikazoval vsebino v pravem jeziku.

3.1.3 ReactJS komponente

Za QoL v ReactJS smo razen App komponente, ki je vstopna točka v aplikacijo, razvili vse komponente sami. Aplikacije ReactJS so SPA (SPA, brez datuma), kar pomeni, da so sestavljene iz ene same spletne strani, kateri se vsebina dinamično prepisuje. Iz tega razloga smo ustvarili komponente, ki predstavljajo spletne strani in nanje postavili manjše komponente, ki predstavljajo posamezne sklope elementov in posamezne elemente. Vse komponente, ki smo jih ustvarili, so funkcijske, kar pomeni, da se za stanje in metode življenjske dobe komponente uporabljajo kavlji. Komponenta po potrebi sprejema obvezne in opcijske propse. QoL v ReactJS je pisan v TypeScript in zato je podatkovni tip posameznega propa definiran v komponenti sami. V komponenti je definirano tudi, kaj se zgodi z uporabniškim vmesnikom glede na posamezni prop (React, a JavaScript library, b.d.).

Primer je komponenta za prikaz naslova z ozadjem, ki se pojavi kot prvi element v telesu komponent, ki predstavljajo domačo stran, stran za primerjavo plač med državami in stran s podrobnostmi o urbanih področjih.

```
import React, { FC } from "react";

import classes from "./HeroImage.module.scss";

interface HeroImageProps {
  title: string;
  subtitle: string;
  background: string;
}

const HeroImage: FC<HeroImageProps> = ({ title, subtitle, background }) => {
  return (
    <div
      className={classes.Wrapper}
      style={{ backgroundImage: `url(${background})` }}
    >
      <div>
        <h1>{title}</h1>
        <h2>{subtitle}</h2>
      </div>
    </div>
  );
};

export default HeroImage;
```

Komponenta prejme naslov, podnaslov in sliko kot prop. Prop za naslov prikaže kot glavni naslov, prop za podnaslov prikaže kot podnapis naslova in sliko prikaže kot ozadje komponente. Ta komponenta je postavljena na vsaki strani, vendar ima na vsaki drugačne propse.

Drug primer je komponenta za harmoniko, ki ima prop accordionVisible, ki določa, če bo komponenta vidna ali ne.

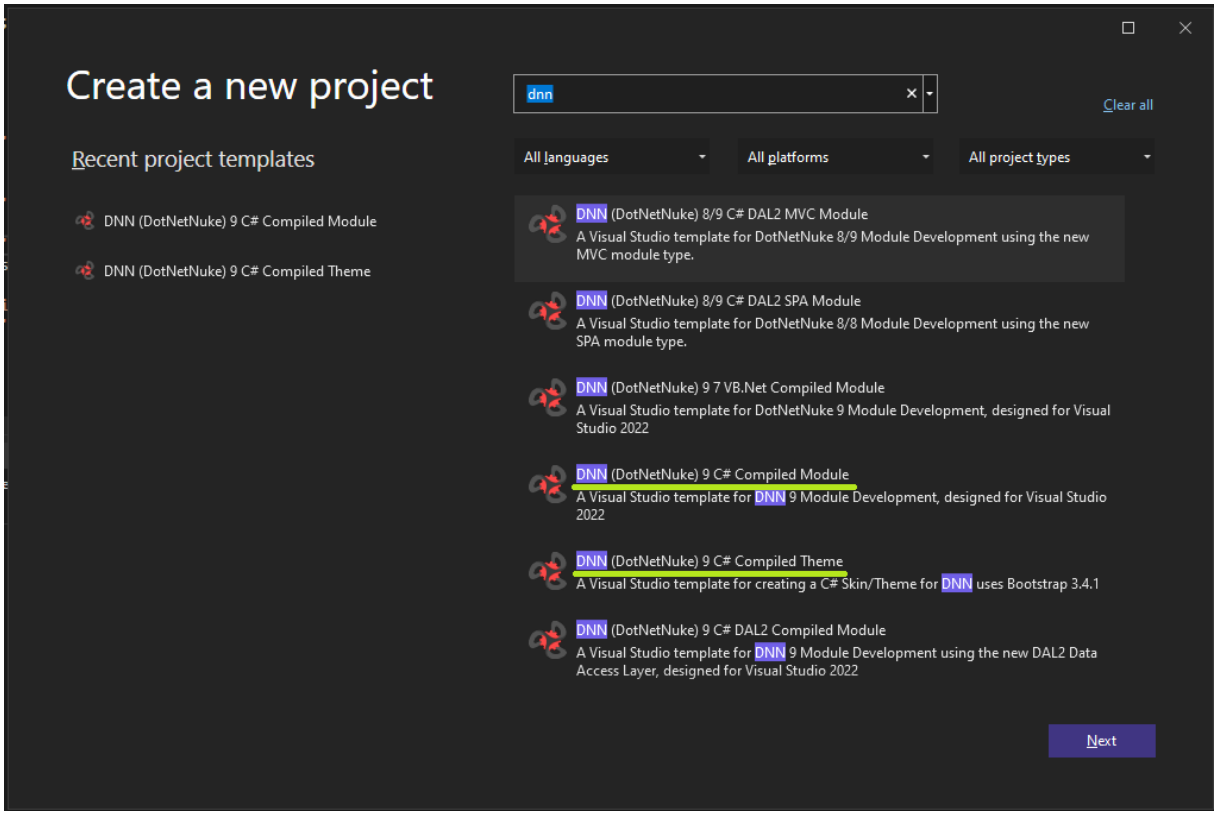
Ker lahko postavljeni komponenti kot prop nastavimo spremenljivko, se ta lahko spreminja. Zaradi navideznega DOM-a se ob spremembi propa ali stanja v brskalniku posodobi le spremenjen del uporabniškega vmesnika, kar pripomore k odzivnosti spletne aplikacije. Podobno kot props ima komponenta s pomočjo kaveljev tudi stanje, katerega vrednosti vplivajo na prikaz vsebine in ob spremembi vrednosti sprožijo osvežitev dotičnega dela uporabniškega vmesnika. Razlika je v tem, da se propsi določijo v starševski komponenti, stanje pa pripada komponenti sami.

3.2 Implementacija in vzdrževanje

3.2.1 Implementacija

Za implementacijo spletne aplikacije QoL v ogrodju DNN je bilo potrebno na operacijski sistem Windows prenesti namestitvene datoteke iz spletne strani DNN. Uporabniškemu računu sistema Windows, ki zaganja spletno aplikacijo, je bilo potrebno dodeliti pravice za urejanje (ang. modify) za namestitven direktorij. V IIS (ang. Internet Information Services Manager) smo dodali novo spletno stran, za katero smo določili pot do datotek, ime gostitelja (ang. host name) in ostale potrebne nastavitve. Zadnji korak vzpostavitve spletne aplikacije v DNN je bil zagon namestitvenega čarovnika (ang. Installation wizard), kjer smo nastavili uporabniško ime, elektronski naslov in geslo za glavni administratorski račun (ang. superuser) (DNN Software).

S pomočjo DNN razširitve smo v MS Visual Studio ustvarili novo DNN temo in kasneje nov DNN modul.



Slika 24: Izbor predloge za razvoj DNN modulov in tem v MS Visual Studio 2020.

Vir: Lastni vir, 2022.

Znotraj nove teme smo ustvarili predlog postavitve (ang. layout template), ki je določal glavo in nogo ter vsebnike oz. vsebinska podokna (ang. content pane) za module. DNN modul smo razvili na način, da glede na izbrano nastavitev prikaže enega od dveh najbolj zapletenih sklopov uporabniškega vmesnika – primerjavo plač med državami in prikaz podrobnosti urbanih področij. Modul in temo smo namestili na DNN spletno aplikacijo v njenem uporabniškem vmesniku. V vmesniku smo prav tako ustvarili vse spletne strani, jih konfigurirali, vklopili lokalizacijo ter postavili HTML in QoL module na njih. CSS za HTML module smo definirali v CSS datoteki teme.

Za vzpostavitev spletne aplikacije v ReactJS smo na računalnik morali namestiti NPM in Node.js. V urejevalniku kode MS Visual Studio Code smo odprli terminal, v katerem smo pognali ukaze, ki so s pomočjo NPM namestili potrebne knjižnice. V terminalu smo prav tako pognali ukaz za kreiranje spletne aplikacije ReactJS z orodno verigo Create React App (React, a JavaScript library, b.d.). Sledila je implementacija spletnih strani in vsebin v obliki funkcijskih komponent ter logika za navigacijo in lokalizacijo. Komponente so samostojni ponovno uporabni deli uporabniškega vmesnika, ki vsebujejo elemente HTML, CSS in logiko ter se prilagajajo glede na props. Sklope CSS kode, ki se v aplikaciji QoL pojavljajo pogosto, smo implementirali na način, da jih izvažamo iz enega mesta in uvažamo po vsej aplikaciji, kjer je to potrebno. CSS smo implementirali s Sass kar nam je omogočilo definirati barve in prelomne točke za responsive design v obliki spremenljivk.

```
// color palette

$color-txt: #333;
$color-title: #04dec1;
$color-bg-btn: #090909;
$color-bg-header: #222;
$color-bg-main: #fff;
$color-bg-disclamer: #d8edea;

// breakpoints

$screen-extra-large: 1200px;
$screen-large: 991px;
$screen-tablet: 768px;
$screen-small: 576px;
$screen-mobile: 480px;
```

Te spremenljivke smo lahko izvozili v vse SCSS datoteke, kjer smo jih potrebovali. Ta način nam omogoča, da spremenimo barvo za na primer naslove na enem mestu in se spremeni povsod.

3.2.2 Vzdrževanje

Vzdrževanje spletnih aplikacij DNN se izvaja na več načinov. Preproste spremembe vsebine in menjave slik v HTML modulih lahko ureja uporabnik brez znanja spletnega razvijalca, vendar mora vse spremembe izvesti ročno na vsaki jezikovni različici posebej. DNN omogoča povezane module, ki delujejo na način, da se sprememba shranjena za en HTML modul odraža na povezanem modulu na drugi spletni strani, kar je priročno za module glave ali noge, ki so enaki na vseh straneh, vendar ta povezava ne velja za spletne strani v drugem jeziku. Administrator lahko postavlja ali prestavlja module na spletnih straneh, vendar je pri tem omejen s postavitvijo vsebinskih podoken, katere definira spletni razvijalec. ReactJS ni CMS, zato vso vzdrževanje izvajajo razvijalci. Prednost tega je, da so vsa besedila za vsak jezik v svoji datoteki in se jih pridobiva s ključem, kar omogoča veliko večjo preglednost vsebine in lažjo implementacijo večjezičnosti. To je še večji faktor za obseg dela pri vzdrževanju, če je projekt postavljen na več okoljih, kot je testno in produkcijsko, kjer je vse spremembe v HTML modulih potrebno izvesti na vsakem okolju posebej, med tem ko se za aplikacijo ReactJS samo v obe okolji prenese nove datoteke s prevodi. Podobno velja za vse aspekte spletne aplikacije DNN. Edine spremembe, ki se lahko v obliki datotek prenašajo iz lokalnega okolja na testno in produkcijsko, so teme in moduli razviti iz strani razvijalcev. Postavitev in konfiguracija modulov, ustvarjanje in konfiguriranje spletnih strani ter vse nastavitve, ki se določajo v DNN uporabniškem vmesniku, je na vseh okoljih potrebno izvesti ločeno. V ReactJS se vse spletne strani, komponente, postavitve in vsebine implementira v datotekah. Za posodobitev spletnih aplikacij ReactJS na več okoljih je potreben samo prenos datotek. Prednost ReactJS je tudi, da razvijalec pri postavitvi komponent ni omejen na vsebinska podokna, saj jih lahko postavi kjerkoli, hkrati pa jim lahko določi napredne videze, vsebine ali nastavitve ter jih lažje prilagodi na preostanek uporabniškega vmesnika (React, a JavaScript library, b.d.).

3.3 Potrebno znanje razvijalcev

Za DNN:

- HTML:

Uporabniški vmesnik v DNN se gradi iz HTML elementov in po potrebi ASP.NET

kontrol, ki se poslužujejo enake sintakse. Za učinkovit razvoj modulov in tem je potrebno dobro znanje HTML.

- CSS:

Kot HTML je CSS osnovni pogoj za razvoj uporabniškega vmesnika, saj z njim definiramo videz posameznih elementov. Potrebno znanje je odvisno od zahtevnosti projekta. QoL je iz vidika CSS zahtevnejši projekt in zahteva dobro poznavanje jezika.

- JavaScript:

Dinamična manipulacija objektov HTML elementov v DOM se upravlja z JavaScript. Z JavaScript smo delno tudi manipulirali podatke in jih dodeljevali različnim elementom. Za razvoj QoL v DNN je bilo potrebno osnovno znanje JavaScript. Uporaba standarda ES6 ni bila potrebna (ECMAScript 6, brez datuma).

- Bootstrap:

DNN ima že privzeto nameščeno knjižnico Bootstrap, s pomočjo katere se implementira responsive design. Bootstrap deluje na način, da se elementom dodeljuje Bootstrapove CSS razrede, ki določajo prikaz elementov za različne velikosti zaslonov. Razreda za vsebnik razdeli določeno površino na dvanajst enakih kolon, razred za kolono, ki ga ima element postavljen v vsebnik, pa določa, koliko od dvanajst delov bo element zavzemal na kateri velikosti zaslona. Poleg responsive designa se QoL v DNN poslužuje tudi Bootstrap Modala. To je pojavno okno, ki ima v Bootstrapu definirano vedenje (prikaz in zapiranje z animacijo, zapiranje ob kliku na ozadje) in videz. Za razvoj QoL je bilo potrebno dobro znanje Bootstrapa (Grid System, b.d.).

- jQuery:

jQuery je JavaScript knjižnica, ki omogoča lažjo manipulacijo objektov elementov v DOM. Za razvoj QoL je nujno potrebna le za prikaz in skrivanje Bootstrap pojavnih oken. Ostale funkcionalnosti, ki smo jih v QoL implementirali z jQuery, bi lahko implementirali tudi s klasičnim JavaScript, vendar jQuery delo olajša. Potrebno znanje jQuery je iz tega razloga za razvoj QoL minimalno, vendar priporočljivo (jQuery.com).

- C#:

DNN je zgrajen na ogrodju ASP.NET, kar pomeni, da je zaledna logika pisana v jeziku C#. Za razvoj QoL v DNN je bilo potrebno razviti QoL modul, ki v svojih nastavitvah administratorjem omogoča izbor uporabniške kontrole za prikaz. Logiko za te nastavitve smo implementirali v datotekah C#. V jeziku C# smo prav tako implementirali klice na spletni API, C# razrede, iz katerih smo ustvarjali objekte za države, urbana področja ipd. ter logiko prikazovanja, skrivanja in generiranja vsebin v zalednih datotekah uporabniških kontrol. Za učinkovit razvoj je bilo potrebno osnovno znanje jezika C#.

- DNN:

Prvi pogoj za razvoj spletne aplikacije v DNN je razumevanje samega ogrodja. Slednje deluje na način, da se nanj nameščajo moduli in teme DNN, ki so pridobljene od zunanjih virov ali razvite po lastnih potrebah. V uporabniškem vmesniku omogoča ustvarjanje spletnih strani in nastavljanje tem ter postavljanje modulov. DNN ima zelo obsežni uporabniški vmesnik za urejanje administracije. Za razvoj QoL je bilo potrebno znati kreirati nove spletne strani, jih konfigurirati, jim nastaviti temo, nanje postaviti module in modulom nastaviti vsebnike. Poleg naštetega je bilo potrebno znanje o vključevanju lokalizacije, kreiranju jezikovnih različic spletnih strani, poznavanje privzetih DNN modulov kot so HTML modul in modul za vpis z uporabniškim računom in poznavanje ASP.NET kontrol (DNN Software).

Za ReactJS:

- HTML:

Uporabniški vmesnik v aplikacijah implementiranih v ReactJS je sestavljen iz HTML elementov in iz ReactJS komponent, ki so na najnižjem nivoju prav tako sestavljene iz HTML elementov. Za razvoj je potrebno dobro znanje HTML.

- CSS:

Kot pri spletnih aplikacijah v DNN je za razvoj aplikacijah v ReactJS CSS osnova. Kljub razliki v tehnologijah je za implementacijo obeh aplikacij QoL bilo potrebno enako dobro poznavanje CSS.

- JavaScript:

ReactJS je JavaScript knjižnica. Vsa koda ReactJS aplikacije je JavaScript. Tudi HTML se piše v JavaScriptu v JSX ali TSX datotekah. JavaScript je najbolj osnovni pogoj za razvoj spletnih aplikacij v ReactJS in zahteva od vseh zajetih tehnologij največ znanja. Vsa logika vključno s klici na spletne APIje in uporabo zunanjih knjižnic je napisana v JavaScriptu (Cleary, 2020).

- Sass:

Sass omogoča definiranje barv, pisav in pogosto uporabljenih CSS pravil v obliki spremenljivk, ki se po potrebi izvozijo v scss datoteke komponent. Prednost tega pristopa je, da lahko tem spremenljivkam spremenimo vrednost na enem mestu in s tem spremenimo na primer primarno barvo v celotni spletni aplikaciji. Sass za razvoj QoL v ReactJS ni obvezen, je pa zaradi lažjega vzdrževanja priporočljiv. Potrebno je le osnovno znanje (Sass).

- ES6:

ReactJS uporablja ES6 in za razvoj spletnih aplikacij v ReactJS in razumevanje dokumentacije ReactJS ter dokumentacije njegovih knjižnic je potrebno dobro poznavanje ES6 novosti, kot so razredi in vmesniki (ang. classes, interfaces), puščične funkcije (ang. arrow functions), moduli (izvažanje in uvažanje JavaScript elementov), razširitveni operator, vrste spremenljivk (var, let, const), in nenazadnje metode za sezname (ang. array methods) (ECMAScript 6).

- NPM:

Med vzpostavitvijo okolja za razvoj aplikacije QoL v ReactJS in med samim razvojem je bilo potrebno namestiti več knjižnic, kot so Axios, i18next in Redux. V ReactJS aplikacijah se te knjižnice nameščajo in odstranjujejo z NPM. V spletnem registru NPM je ogromni nabor odprtokodnih knjižnic v obliki paketov. Za razvoj QoL v ReactJS je bilo potrebno osnovno razumevanje NPM (React, a JavaScript library, b.d.).

- ReactJS:

Za razvoj QoL v ReactJS je bilo potrebno dobro poznavanje knjižnice ReactJS. Razvijalec mora poznati vrste komponent, razumeti, kaj so props in kako se uporabljajo, razumeti koncept stanja, manipulacijo elementov s stanjem, kavlje in JSX sintakso. Poleg naštetega je bilo za razvoj QoL potrebno tudi dobro razumevanje ReactJS knjižnic, kot so React Redux (Getting Started with Redux Toolkit, brez datuma), Redux Toolkit, Axios, in i18next (React, a JavaScript library, b.d.).

Tabela 1: Razpredelnica potrebnih znanj za implementacijo QoL v ReactJS.

	DNN	ReactJS
HTML	X	X
CSS	X	X
JavaScript	X	X
Bootstrap	X	-
DNN	X	-
jQuery	X	-
C#	X	-
DNN	X	-
ReactJS	-	X
Sass	-	X
ES6	-	X
NPM	-	X

Vir: Lastni vir, 2022.

V primeru, da ima razvijalec ali razvojna ekipa le znanje potrebno za razvoj QoL v DNN in si prizadeva izvesti tranzicijo ekvivalentne aplikacije na ReactJS, mora osvojiti globje razumevanje JavaScript, ES6, NPM, ReactJS in ReactJS knjižnic.

3.4 Hitrost in odzivnost

Meritve hitrosti in odzivnosti spletnih aplikacij QoL v DNN in v ReactJS smo izvedli na enem računalniku v lokalnem okolju, da so rezultati zanesljivi in niso odvisni od zasedenosti in omejitev spletne povezave. Hitrost v naši raziskavi predstavlja odmerjen čas od začetka do konca nalaganja posamezne spletne strani. Odzivnost predstavlja odmerjen čas od proženja akcije na uporabniškem vmesniku do konca generiranja spremembe na uporabniškem vmesniku. Meritve smo opravili v brskalniku Google Chrome, v katerem smo si pomagali z razvojnimi orodji (Chrome DevTools) (Prokopets, 2022). Aplikaciji smo odprli v zavihkih brez beleženja zgodovine, da v predpomnilniku ni bilo ničesar in smo lahko izmerili realno hitrost nalaganja. DNN zajema nastavitve za omogočanje uporabe predpomnilnika, kar izboljša čas nalaganja spletne aplikacije ob nadaljnjih obiskih, vendar ne ob prvem, in hkrati ne zagotavlja, da uporabnik vedno vidi najsodobnejšo vsebino, ker je lahko prikazana stara iz predpomnilnika. Iz tega razloga smo meritve izvedli brez uporabe predpomnilnika (React, a JavaScript library, b.d.).

3.4.1 Hitrost nalaganja

Meritve smo izvedli na način, da smo v brskalniku Chrome odprli zavihek brez beleženja zgodovine zanj pognali DevTools ter odprli zavihek za omrežje (ang. network). Naslednji korak je bil vnos naslova, ki pelje do spletne aplikacije. Podatki v spodnjih tabelah so odčitani iz zavihka za omrežje. Ker obe spletni aplikaciji tečeta na operacijskem sistemu Windows, kjer v ozadju teče veliko procesov, se posledično prioriteta pri dodeljevanju sredstev namenjenih za delovanje spletne aplikacije spreminja in so rezultati hitrosti nalaganja pri vsaki meritvi drugačne. Odstopanja so bila minimalna, vendar smo v vsaki aplikaciji izvedli deset meritev in v tabele spodaj zapisali povprečni čas nalaganj vseh meritev.

- **Domača stran**

Tabela 2: Čas nalaganja domače strani na QoL v DNN in v ReactJS.

	Št. omrežnih zahtev	Preneseno preko omrežja	Naloženo na strani	DOM elementi naloženi po	**Končano po**
DNN	34	1,2 MB	2,0 MB	10970 ms	**11022 ms**
ReactJS	13	1,5 MB	2,1 MB	130 ms	**289 ms**

Vir: Lastni vir, 2022

- **Urbana področja**

Tabela 3: Čas nalaganja strani Urbana področja na QoL v DNN in v ReactJS.

	Št. omrežnih zahtev	Preneseno preko omrežja	Naloženo na strani	DOM elementi naloženi po	**Končano po**
DNN	31	874 kB	1,7 MB	3873 ms	**3944 ms**
ReactJS	10	1,2 MB	1,8 MB	131 ms	**292 ms**

Vir: Lastni vir, 2022.

- **Primerjava plač med državami**

Tabela 4: Čas nalaganja strani Primerjava plač med državami na QoL v DNN in v ReactJS.

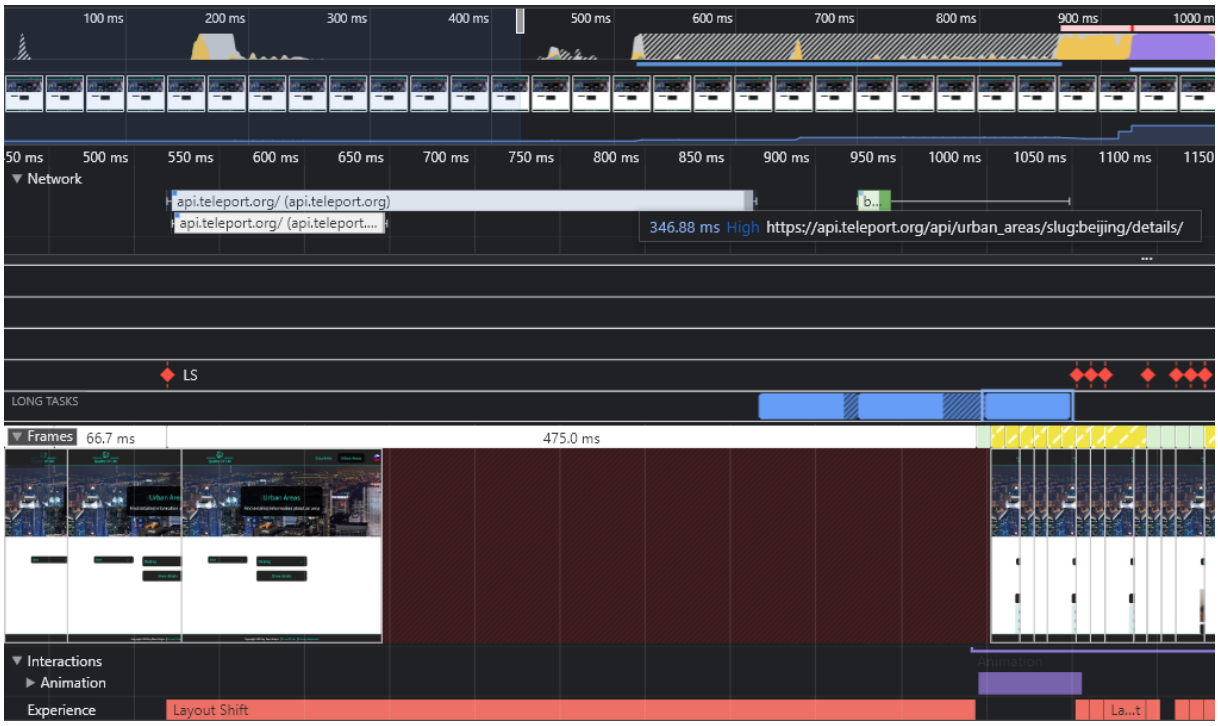
	Št. omrežnih zahtev	Preneseno preko omrežja	Naloženo na strani	DOM elementi naloženi po	**Končano po**
DNN	30	1,1 MB	2,0 MB	3992 ms	**4063 ms**
ReactJS	10	1,5 MB	2.0 MB	133 ms	**274 ms**

Vir: Lastni vir, 2022.

3.4.2 Odzivnost

Meritve odzivnosti smo izvedli na spletnih straneh za prikaz podrobnosti urbanih področij in za primerjavo plač med državami, saj se tam večji del uporabniškega vmesnika generira dinamično glede na prejete podatke iz APIja. Meritve smo ponovno izvedli z DevTools v brskalniku Chrome, vendar tokrat v zavihku za delovanje (ang. performance).

Na DNN in ReactJS smo izbrali isto urbano področje in iste države za primerjavo, da so meritve zanesljivejše. V zavihku za delovanje smo odčitali, v kateri milisekundi meritve smo sprožili klik, v kateri milisekundi se je sprememba prikazala na uporabniškem vmesniku in koliko časa so trajali API klici.



Slika 25: Google Chrome DevTools, zavihek za merjenje delovanja.

Vir: Lastni vir, 2022.

- **Izris podrobnosti urbanega področja:**

Izbrali smo celino in urbano področje ter ob kliku na gumb za prikaz podrobnosti pričeli z meritvami, ki smo jih zaključili ob končanem nalaganju podrobnosti. V obeh spletnih aplikacijah smo meritve izvedli na prikazu podrobnosti urbanega področja Peking na angleških straneh.

Tabela 5: Čas nalaganja do zaznanega klika ali prikaza spremembe na upor. Vmesniku.

	DNN	ReactJS
Čas od pričetka meritve do zaznanega klika.	671,0 ms	517 ms
Čas od pričetka meritve do prikaza spremembe na uporabniškem vmesniku.	4809,55 ms	1002,7 ms
Čas od zaznanega klika do prikaza spremembe na uporabniškem vmesniku.	**4138,55 ms**	**485,7 ms**

Vir: Lastni vir, 2022.

- **Izris primerjave plač med državami:**

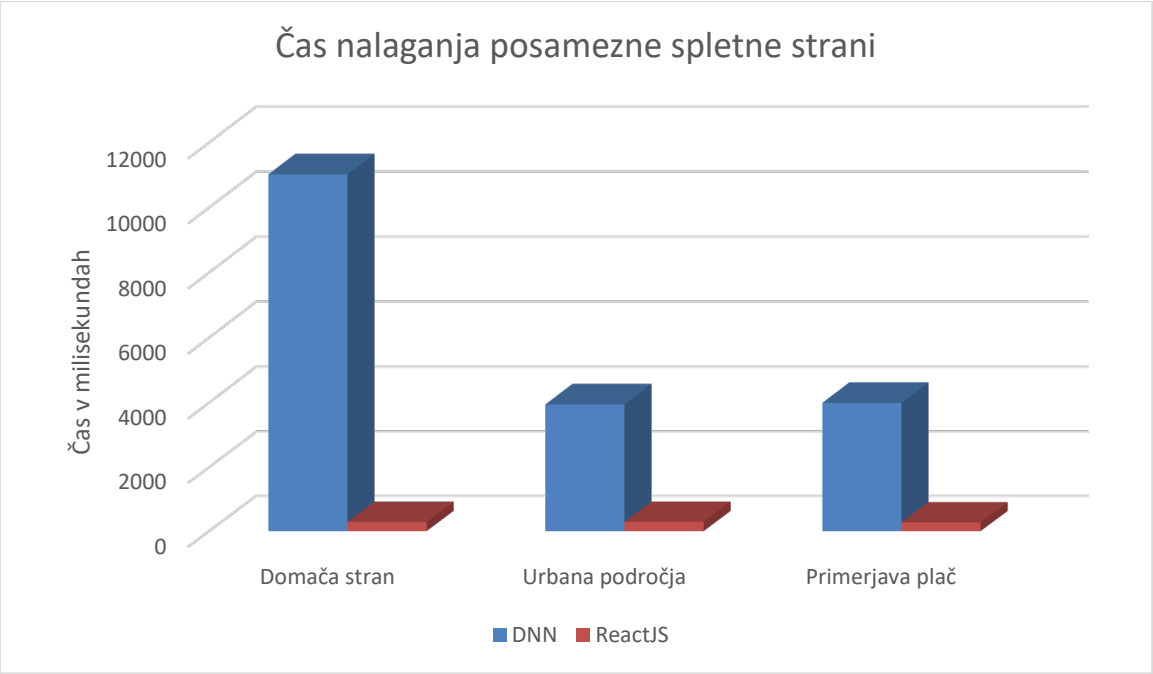
Izbrali smo obe celini in obe državi ter ob kliku na gumb za prikaz primerjave pričeli z meritvami, ki smo jih kot pri urbanih področjih zaključili, ko se je primerjava do konca naložila v brskalniku. V obeh aplikacijah smo meritve izvedli za prikaz primerjave plač v Avstriji in Kanadi, na angleških straneh.

Tabela 6: Čas nalaganja do zaznanega klika ali prikaza spremembe na upor. Vmesniku.

	DNN	ReactJS
Čas od pričetka meritve do zaznanega klika.	673,1 ms	534,6 ms
Čas od pričetka meritve do prikaza spremembe na uporabniškem vmesniku.	5648,36 ms	852,3 ms
Čas od zaznanega klika do prikaza spremembe na uporabniškem vmesniku.	**4975,26 ms**	**317,7 ms**

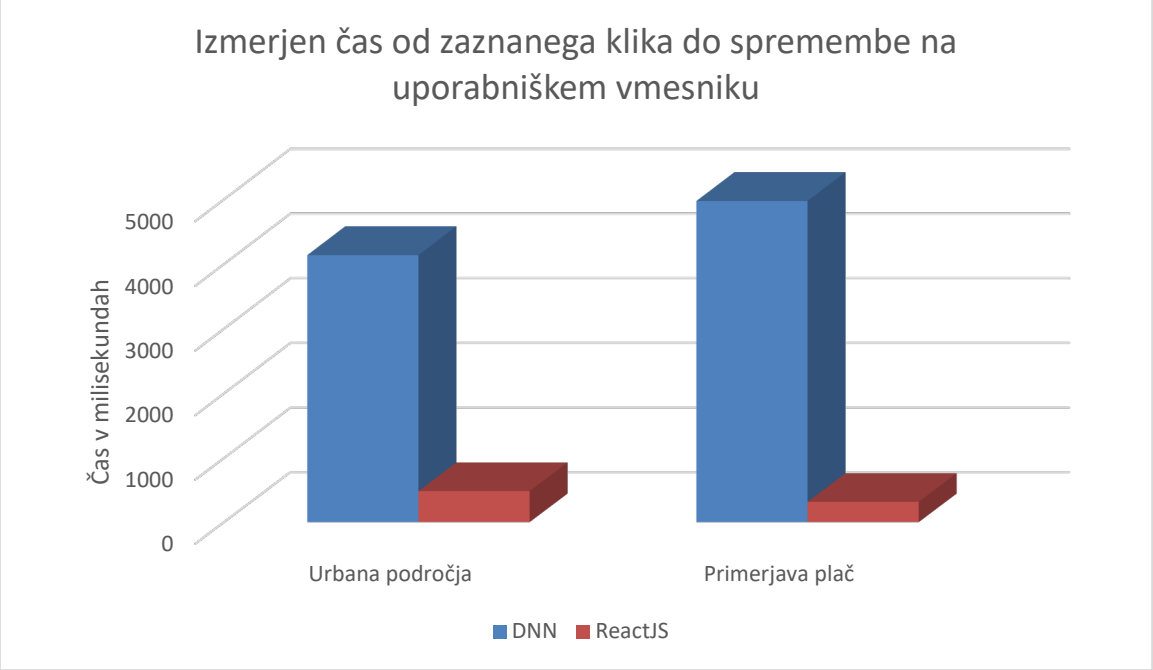
Vir: Lastni vir, 2022.

Glede na naše meritve se hitrost in odzivnost v spletnih aplikacijah QoL v ReactJS in QoL v DNN zelo razlikujeta. Spletne strani v QoL v ReactJS so se v brskalniku Chrome naložile od štirinajst do skoraj štirideset krat hitreje kot spletne strani v DNN in so prikazovale vse elemente v manj kot eni sekundi. Meritve odzivnosti posameznih uporabniških interakcij na strani za prikaz podrobnosti urbanih področij in strani za primerjavo plač med državami so razkrile podobne rezultate. Med tem ko je QoL v DNN od klika za prikaz podrobnosti do dejanskega prikaza podrobnosti o Pekingu potreboval več kot štiri sekunde, je enak postopek v QoL v ReactJS trajal manj kot pol sekunde. Razlika je bila še očitnejša pri prikazu primerjave plač med Avstrijo in Kanado, kjer so se izvedli štirje API klici in kompleksnejša manipulacija prejetih podatkov ter kompleksnejše generiranje uporabniškega vmesnika kot na strani urbanih področij. QoL v DNN je za prikaz primerjave plač od zaznanega klika potreboval skoraj pet sekund, med tem ko je QoL v ReactJS potreboval 317 milisekund, kar je več kot petnajstkrat hitreje.



Grafikon 1: Čas nalaganja spletne strani na obeh implementacijah aplikacije QoL.

Vir: Lastni vir, 2022.



Grafikon 2: Izmerjen čas od klika do izrisa v brskalniku v obeh implementacijah aplikacije QoL.

Vir: Lastni vir, 2022.

3.5 Pridobitve tranzicije iz DNN na ReactJS

Skozi proces tranzicije smo prišli do spoznanja, da je vzdrževanje in prenos posodobitev na drugo okolje bistveno lažji kot v DNN. Vse vsebine in prevodi (razen podatki od spletnega APIja) se v aplikaciji ReactJS nahajajo v JSON datotekah, kar nam omogoča lažji pregled in lažjo izvedbo sprememb kot v DNN (JSON, b.d.), kjer so nekateri prevodi (za HTML module) dostopni samo na uporabniškem vmesniku, drugi (npr. za QoL modul) pa v datotekah za prevode, ki so dostopne le razvijalcem.

Skupna velikost datotek spletne aplikacije QoL v DNN, ki jih je za publikacijo potrebno prenesti na strežnik, znaša 366 MB, med tem ko je skupna velikost datotek za publikacijo QoL v ReactJS le 6,88 MB.

Največja pridobitev prehoda QoL iz DNN na ReactJS je neizpodbitna razlika v hitrosti nalaganja in odzivnosti. Spletne strani se nalagajo štirinajst do osemintridesetkrat hitreje kot na DNN, kompleksnejše vsebine pa se ob uporabniški interakciji dinamično generirajo osem do petnajstkrat hitreje kot na DNN.

4 SMOTRNOST PREHODA IZ DNN V REACTJS

Različne spletne tehnologije služijo različnim namenom in med tem ko nekatere izvajajo ene naloge boljše, so ostale boljše na drugih področjih. Enako velja tudi za ogrodje DNN in ReactJS. Smotrnost procesa tranzicije spletne aplikacije iz DNN na ReactJS je odvisna od potreb projekta, zmogljivost podjetja in težavnosti vzdrževanja. S potrebami projekta so mišljene predvidene funkcionalnosti spletne aplikacije kot podpora za uporabniške račune in uporabniške vloge, CMS in večjezičnost. Zmogljivosti podjetja zajemajo znanje razvijalcev, čas, ki si podjetje lahko privošči za morebitno izobraževanje razvijalcev in čas, ki si ga podjetje lahko privošči za implementacijo. Težavnost vzdrževanja spletnih aplikacij je pomemben dejavnik pri katerikoli tranziciji med spletnimi tehnologijami, saj lahko v večjih spletnih aplikacijah na dolgi rok prihrani več časa, kot ga je bilo potrebnega za samo tranzicijo. V poglavju 3.2.2 smo potrdili, da je spletno aplikacijo v ReactJS lažje vzdrževati kot ekvivaletno aplikacijo v DNN.

4.1 Smotrnost glede na spletne aplikacije

DNN je CMS in služi izgradnji spletnih aplikacij, ki omogočajo uporabnikom brez znanja o spletnem razvoju, da urejajo svoje spletne strani. DNN temu namenu služi dobro, saj omogoča intuitiven in širok uporabniški vmesnik, na katerem lahko administrator ureja preproste vsebine in nastavitve ali napredne sistemske in strežniške konfiguracije, lokalizacijo in upravljanje z uporabniškimi računi, uporabniškimi vlogami in pravicami posamezne uporabniške vloge. Aplikacija v ReactJS brez dodatne implementacije ne vsebuje in ne omogoča nič od naštetega. QoL nima potrebe po CMS, uporabniških računih, uporabniških vlogah ali naprednih konfiguracijah, zajema pa lokalizacijo. Slednjo smo v ReactJS implementirali ročno s pomočjo knjižnice i18next. Skozi proces tranzicije QoL smo močno zmanjšali čas nalaganja spletnih strani in vsebin, izjemno zmanjšali potreben prostor za gostenje spletne aplikacije na strežniku ter olajšali vzdrževanje. Glede na potrebe spletne aplikacije in namen procesa je zadano bilo doseženo in ocenjujemo, da je bil proces tranzicije aplikacije v tehnoligijo ReactJS smiselni. Implementacija lastnega CMS je bistveno težji in daljši proces kot implementacija lokalizacije in v primeru, da je CMS zahtevan za spletno aplikacijo, tranzicija na ReactJS ni smiselna. Podobno velja za podporo za uporabniške račune in uporabniške vloge. Za te je potrebno razviti bazo podatkov in zaledne načine komunikacije kot spletne storitve (ang. web service) ali API, ročno ustvariti uporabniške vloge in pravice ter razviti ves uporabniški vmesnik za upravljanje

administracije na uporabniških računih, uporabniški vmesnik za vpis, pozabljeno geslo in ponastavitev gesla, logiko za pošiljanje povratne e-pošte za obnovitev pozabljenega gesla in mnogo več. Tudi v tem primeru proces ni smiselni.
Spletne aplikacije, ki ne potrebujejo CMS, naprednih konfiguracij in uporabniških računov, je smiselno prenesti iz DNN v ReactJS, saj ne izgubijo ključnih funkcionalnosti, katere bi bilo težko nadomestiti in pridobijo na odzivnosti, hitrosti nalaganja in olajšanem vzdrževanju.

4.2 Smotrnost glede na zmogljivost podjetja

Tudi če proces tranzicije zagotavlja lažje vzdrževanje in spletna aplikacija ne potrebuje CMS ter ostalih DNN funkcionalnosti, je smiselnost procesa v veliki meri odvisna od zmogljivosti podjetja. V primeru, da v podjetju ni potrebnega znanja za izvedbo tranzicije, je potrebno izobraževanje. Upoštevati je treba dejstvo, da za uspešno tranzicijo ni potrebno le znanje ReactJS, vendar tudi znanje DNN. Potrebno znanje smo podrobneje opisali v poglavju 3.3.
Tudi če razvijalci nimajo potrebnega znanja, vendar si podjetje lahko privošči izobraževanje le-teh, je proces smiselni, saj je poleg pridobitev procesa velika pridobitev tudi novo znanje, ki se lahko v podjetju uporabi na sedanjih ali bodočih projektih.
V primeru, da podjetju primanjkujejo sredstva ali potrebni razvijalci za izvedbo tranzicije ali izobraževanje, proces ni smisleni, saj gre za zahtevnejši proces, ki terja znanje in čas.
Končna ocena smotrnosti procesa je v prvi vrsti odvisna glede na potrebe aplikacije, v drugi od zmogljivosti podjetja in v tretji od težavnosti vzdrževanja. Pri oceni smotrnosti za proces tranzicije QoL iz DNN v ReactJS nismo zajeli zmogljivosti dejanskega podjetja, saj smo proces izvedli v okviru diplomskega dela. Ocena je pozitivna, saj proces dosega namen, kateremu služi.

5 SKLEP

Za izdelavo diplomskega dela smo implementirali spletno aplikacijo v ogrodju DNN ter izvedli proces tranzicije v knjižnico ReactJS. Proces smo dokumentirali in na koncu postavili primerjavo med aplikacijama v obeh tehnologijah. Namen diplomskega dela je bila izdelava ocene smotrnosti zadanega procesa glede na potrebe spletne aplikacije in zmogljivost podjetja. V uvodu smo določili naše osnovne trditve, si zastavili raziskovalna vprašanja in podrobneje opisali naš namen. V nadaljevanju smo opisali analizo spletnih strani, funkcionalnosti in vsebine spletne aplikacije QoL v DNN. Analiza je razkrila, da se spletne aplikacije v DNN in v ReactJS v implementaciji, strukturi datotek in drugih kritičnih točkah med seboj tako razlikujejo, da je za tranzicijo na ReactJS potrebna nova implementacija (H2). Na podlagi analize smo izdelali načrt procesa tranzicije, v katerem smo določili katero orodno verigo bomo uporabili za implementacijo ReactJS aplikacije in katere knjižnice bomo potrebovali za razvoj funkcionalnosti, kot so lokalizacija in klici na spletne APIje. Naslednji korak je bila vzpostavitev okolja, ki je zajemala namestitev Microsoftovega integriranega razvojnega okolja Visual Studio Code, namestitev npm in Node.js za poganjanje JavaScript programske kode zunaj brskalnika, implementacijo ReactJS aplikacije z orodno verigo Create React App in namestitev vseh potrebnih knjižnic.

Sledila je implementacija spletnih strani in vsebin v obliki komponent, implementacija lokalizacije s pomočjo i18next in implementacija klicev na končne točke APIja. Proces implementacije je podrobno opisan v poglavjih 2.3 – 2.5. Skozi proces tranzicije smo prišli do ugotovitve, da je proces tranzicije v celoti izvedljiv z delom od doma (H5). V nadaljevanju smo našteli in opisali funkcionalnosti, ki smo jih izgubili ob prehodu iz CMS, saj so za določene vrste spletnih aplikacij potrebne.

Po zaključenem razvoju spletnih aplikacij QoL v DNN in ReactJS smo podrobneje opisali načine samih implementacij in primerjali vzdrževanje v obeh tehnologijah pri čemer smo prišli do ugotovitve, da je vzdrževanje spletne aplikacije ReactJS veliko lažje kot vzdrževanje ekvivalentne spletne aplikacije v DNN (H4). Razlog za to je, da je v DNN besedila vsebinskih modulov potrebno urejati na vsaki jezikovni različici spletne strani posebej ter delo ponavljati na vsakem razvojnem okolju, medtem ko je pri ReactJS aplikaciji potrebno le zamenjati datoteke.

V nadaljevanju smo našteli potrebne skriptne in programske jezike ter ostale tehnologije, ki so potrebne za učinkovit razvoj QoL v DNN in v ReactJS ter opisali zakaj in v kolikšni meri so potrebne. Te smo prikazali v pregledni tabeli.

Naslednji korak je bila primerjava hitrosti in odzivnosti spletne aplikacije DNN in ReactJS. Meritve smo za obe implementaciji aplikacije QoL izvedli na istem računalniku z orodjem Chrome DevTools (Prokopets, 2022). Spletna aplikacija v ReactJS je v vseh naših primerjavah imela veliko boljše rezultate kot DNN (H3). Pridobitve tranzicije so za spletno aplikacijo QoL pomenile lažje vzdrževanje ter mnogo krajši čas nalaganja in dinamičnega generiranja vsebin na uporabniškem vmesniku. Na podlagi težavnosti procesa, rezultatov naših meritev in potreb spletne aplikacije smo izdelali oceno smotrnosti procesa tranzicije spletne aplikacije iz DNN v ReactJS. Oceno smotrnosti smo izdelali tudi glede na omejitve podjetja in potrebno znanje razvijalcev. Proces je neizpodbitno smiselni v primeru, ko spletna aplikacija razvita v DNN nima potrebe po CMS in uporabniških računih ter ima podjetje razvijalce s potrebnim znanjem ali sredstva in čas, da jih izobrazi (H1).

Na vsa zastavljena raziskovalna vprašanja smo našli odgovore in vse naše osnovne trditve smo potrdili. Naša ocena smotrnosti procesa lahko pomaga razvijalcem, ki imajo spletno aplikacijo v DNN in ne potrebujejo CMS, lažje sprejeti odločitev o izvedbi procesa tranzicije. Diplomsko delo ne poda ocene smotrnosti za izvedbo tranzicije za spletno aplikacijo, ki CMS potrebuje, saj bi razvoj lastnega CMS v ReactJS presegel obseg naše raziskave.

VIRI IN LITERATURA

API explorer. (brez datuma). Pridobljeno iz Teleport public APIs: https://developers.teleport.org/api/reference/

Cleary, R. (6. julij 2020). *The Pros and Cons of Functional Components in React*. Pridobljeno iz betterprogramming.pub: betterprogramming.pub

Code Academy. (b.d.). Pridobljeno iz www.codeacademy.com: https://www.codeacademy.com/

DNN Software. (brez datuma). Pridobljeno iz dnnsoftware.com: https://www.dnnsoftware.com/docs/developers/setup/index.html

DNN urejevalnik spletnih vsebin. (brez datuma). Pridobljeno iz dnn.si: http://dnn.si/

Download. (brez datuma). Pridobljeno iz DNN: https://www.dnnsoftware.com/community/download

ECMAScript 6. (brez datuma). Pridobljeno iz es6-features.org: http://es6-features.org/

Getting Started with React Redux. (brez datuma). Pridobljeno iz React Redux: react-redux.js.org/introduction/getting-started

Getting Started with Redux Toolkit. (brez datuma). Pridobljeno iz Redux Toolkit: redux-toolkit.js.org/introduction/getting-started

Google Search Central. (brez datuma). Pridobljeno iz developers.google.com: https://developers.google.com/

Grid System. (b.d.). Pridobljeno iz getbootstrap.com: https://getbootstrap.com/

i18next ducumentation. (brez datuma). Pridobljeno iz i18next.com: https://www.i18next.com/

jQuery.com. (brez datuma). Pridobljeno iz www.jquery.com: https://jquery.com/

JSON. (b.d.). Pridobljeno iz www.json.org: https://www.json.org/

MOJWEB solutions. (brez datuma). Pridobljeno iz mojweb.si: https://www.mojeweb.com/

Ocvirk, V. (2010). Ustreznost odprtokodnih sistemov za upravljanje vsebin za načrtovanje in izvedbo kompleksnih spletnih mest. Diplomsko delo. Ljubljana.

Prokopets, M. (2022). *The Beginner's Guide to Chrome Developer Tools*. Pridobljeno iz nira Blog: https://nira.com/chrome-developer-tools/

Rapidapi. (brez datuma). Pridobljeno iz rapidapi.com: https://rapidapi.com/

React, a JavaScript library. (b.d.). Pridobljeno iz reactjs.org: https://reactjs.org/

Red Hat. (brez datuma). Pridobljeno iz www.redhat.com: https://www.redhat.com/

Rožman, A. (2016). Primerjava razvojnih modelov .NET MVC in .NET Web Forms. Diplomsko delo. Ljubljana.

Sass. (brez datuma). Pridobljeno iz sass-lang.com: https://sass-lang.com/

SPA. (brez datuma). Pridobljeno iz developer.mozilla.org: https://developer.mozilla.org/

SSKJ Slovar slovenskega knjižnega jezika. (2014). Pridobljeno iz fran.si: fran.si

techopedia. (31. avgust 2020). Pridobljeno iz Pixel: https://www.techopedia.com/

Teleport public APIs. (brez datuma). Pridobljeno iz developers.teleport.org: https://developers.teleport.org/

TypeScript. (brez datuma). Pridobljeno iz typescriptlang.org: https://www.typescriptlang.org/

w3schools. (brez datuma). Pridobljeno iz w3schools.com: https://www.w3schools.com