Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Aplikace React Javascript – atomický design

Alina Altynbayeva

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Alina Altynbayeva

Informatika

Název práce

Aplikace React JavaScript – atomický design

Název anglicky

React JavaScript Application – atomic design

Cíle práce

V bakalářské práci bude vytvořena knihovna (design system), která se používá pro frontendový vývoj webových stránek. Cílem této bakalářské práce je: naučit se, jak knihovna funguje při tvorbě webových stránek a aplikací a jak se liší, vytvořit volně šiřitelný balíček opakovaně použitelných komponent zveřejněný v balíčkovacím systému yarn.

Metodika

Aplikace Storybook a JavaScript framework React JS budou použity jako základ pro vytvoření aplikace.

Bude proveden přehled literatury, abyste porozuměli Reactu, design systémům, atomickému designu a osvědčeným postupům při vývoji Reactu. Vypracujte výzkumné otázky a cíle.

Budou zvolené vhodné výzkumné metody (např. průzkumy, rozhovory, případové studie).

Bude vyvinut design systém založený na Reactu s využitím principů atomického designu(atomic design).

Bude se zkoušet účinnost design systému a osvědčených postupů (např. testování, optimalizace výkonu). Provede se analýza a vyvozené závěry.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

UI, React, React components, design-system, grafické rozhraní, web, webová stránka, uživatelské rozhraní, atomický design

---

Doporučené zdroje informací

GODBOLT, Micah. Frontend architecture for design systems: a modern blueprint for scalable and sustainable websites. Sebastopol, California: Oreilly Media, 2016. ISBN 978-1491926789.

SANTANA ROLDÁN, Carlos. Learning React Design Patterns. Packt Publishing, 2017. ISBN 978-1786464538.

STEFANOV, Stoyan. React: up & running: building web applications. Sebastopol, California: O'Reilly, 2016. ISBN 978-1491931820.

VERISSIMO, Michele Bertoli. React Design Patterns and Best Practices. Packt Publishing, 2017. ISBN 978-1786464538.

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. David Buchtela, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 9. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 07. 03. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Aplikace React Javascript – atomický design" jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 15.3.2024 _____

Poděkování

Ráda bych poděkovala vedoucímu mé bakalářské práce panu Ing. Davidu Buchtelovi,Ph.D. za odborné konzultace, velice cenné rady, věcné připomínky, vstřícnost a trpělivost při konzultacích. Děkuji rodině za podporu při studiu.

Aplikace React Javascript – atomický design

Abstrakt

Tato bakalářská práce se zaměřuje na vypracování atomického design systému v prostředí frameworku React.js s hlavním cílem srovnání výkonu aplikací a jejich optimalizace za použití dostupných nástrojů. Teoretická část práce analyzuje historický vývoj webu a postupně přechází k detailnímu popisu základních stavebních prvků webu, včetně HTML, CSS a JavaScript. Praktická část práce se zaměřuje na implementaci atomického design systému ve frameworku React.js s následnou optimalizaci a vyhodnocením. Výsledky této práce poskytují ucelený pohled na využití atomického designu a jeho dopad na výkonost webových aplikací v prostředí frameworku React.js.

Klíčová slova: design systém, atomický design, React.js, optimalizace, testování

Application React Javascript – atomic design

Abstract

This bachelor thesis aims to create an atomic design system within the React.js framework environment. The primary objective of this study is to compare the performance of applications and optimize them using available tools. The theoretical part of the thesis analyses the historical evolution of the web and provides a detailed description of its main elements, including HTML, CSS, and JavaScript. The practical part of the thesis focuses on implementing an atomic design system using React.js followed by the optimization and the evaluation. The results of the bachelor thesis offer a compacted perspective on the use of atomic design and its impact on the performance of web applications within the React.js development framework environment.

Keywords: design system, atomic design, React.js, optimization, testing

Obsah

1 Úvod

Při vývoji webových aplikací se aplikuje mnoho různých postupů a cest k dosažení tíženého cíle, vývojem takové aplikace se rozumí užívání rozličných nástrojů, zkušeností, praktik a již vytvořených pracovních schémat / postupů. S každou další takovou aplikací se, ale počítá jako s uživatelsky přívětivou, užívajících nejnovějších designových prvků a nejaktuálnějšího programovacího jazykového setu. K vývoji webových aplikací se nejčastěji užívá kombinace těchto programovacích jazyků – HTML, CSS a Javascript. HTML a CSS si do hloubky představíme v následujících kapitolách této práce.

Javascript je nejrozšířenějším programovacím jazykem klientského rozhraní webu, kde zajišťuje spojení mezi uživatelem a aplikací. Postupem času se na základech Javascriptu vyvinulo veliké množství frameworků a knihoven sloužící k snazšímu vývoji webových stránek a aplikací. Knihovny se zaměřují na řešení specifických problémů, jako jsou například validita, stylování a efekty, manipulace s DOM a state management. Frameworky se zabývají komplexním vývojem aplikace, užívají svoji vlastní architekturu kódu, kterou je nutné dodržovat. Jejich vývoj neustále pokračuje, a tak je pro každého vývojáře dobré vědět, co chce využívat, aby mohl co nejefektivněji využít dostupných technologií.

V této bakalářské práci se budu zabývat implementací a vývojem knihovny, kterou bude možné volně užívat, budeme postupovat po malých jednoduchých krocích a následně nabalovat vícero komplexních řešení.

2 Cíl práce a metodika

2.1 Cíl práce

V rámci této bakalářské práce bude vyvinuta a detailně popsaná knihovna, často označovaná jako „design systém," která slouží jako klíčový nástroj pro front end vývoj webových stránek a aplikací. Hlavním cílem této práce je nejen vytvořit tuto knihovnu, ale také důkladně porozumět jejímu fungování a způsobu, jakým ovlivňuje proces tvorby webových stránek a aplikací.

Tato knihovna bude vytvořena s ohledem na opakované použití a snadnou rozšiřitelnost, což umožní vývojářům využívat ji jako základní stavební kámen pro své projekty. Tím pádem tato bakalářská práce zahrnuje nejen samotný vývoj knihovny, ale také analýzu toho, jakým způsobem tato knihovna usnadňuje a urychluje proces vývoje webových stránek a aplikací.

Dalším důležitým aspektem této práce je zdůraznění rozdílů mezi vytvořenou knihovnou a běžnými postupy při tvorbě webových stránek a aplikací. Tímto způsobem bude možné identifikovat, jakým způsobem použití této knihovny přináší přidanou hodnotu a zlepšuje efektivitu vývojového procesu.

Kromě toho je v plánu zveřejnit vytvořenou knihovnu jako volně šiřitelný balíček, který bude dostupný v balíčkovacím systému Yarn. Tím bude knihovna přístupná široké veřejnosti a bude moci být využívána v různých projektech bez nutnosti opakovaného vytváření podobných komponent. Tímto způsobem bude knihovna přispívat k efektivnímu sdílení a znovupoužití komponent v rámci vývojové komunity.

Celkově lze tuto bakalářskou práci vnímat jako komplexní projekt, který spojuje vývojové dovednosti s analýzou a šířením znalostí o tvorbě webových stránek a aplikací prostřednictvím vytvořené knihovny a jejího následného zveřejnění v balíčkovacím systému Yarn.

2.2 Metodika

Aplikace Storybook a JavaScript framework React JS budou použity jako základ pro vytvoření aplikace.

Bude proveden přehled literatury, abyste porozuměli Reactu, design systémům, atomickému designu a osvědčeným postupům při vývoji Reactu. Vypracujte výzkumné otázky a cíle.

Budou zvolené vhodné výzkumné metody (např. průzkumy, rozhovory, případové studie). Bude vyvinut design systém založený na Reactu s využitím principů atomického designu (atomic design). Bude se zkoušet účinnost design systému a osvědčených postupů (např. testování, optimalizace výkonu). Provede se analýza a vyvozené závěry.

3 Evoluce webu

„Na počátku byl Web a ten byl dobrý. Ale, v tomto případě, počátek byl začátkem 90. let a "dobrý" znamenalo, že se web dostal na indexovou stránku Yahoo! a počítadlo návštěvníků se točilo ve spodní části stránky plné tabulek a animovaných GIFů." (Godbolt, 2016, str. 1. paragraf Origins) (překlad vlastní)

Web, slovo, které jsme již všichni slyšeli, ale málo z nás si uvědomí, jakým vývojem prošel od svého počátku. Jedním z podstatných, lze říci i klíčových milníků, bylo vytvoření WWW neboli World Wide Webu, což je připisováno Timu Berners-Leeovi, který tehdy pracoval v CERNu a snažil se vylepšit sdílení informací tak, aby nebylo třeba opouštět pracoviště a přesouvat se k jinému počítači, který měl k informaci přístup. Zároveň napsal tři fundamentální technologie webu HTML, http a URI nazývané též URL. (World Wide Web Foundation, 2020)

3.1 WEB 1.0

Web 1.0 byla první strukturou webu určenou pro veřejnost, která byla poprvé touto formou vytvořena v 90. letech 20. století. Stál na základních třech základech: http, URI a HTML. Uživatel měl pouze omezené možnosti, protože stránky byly pouhou statickou informací pro čtení a stahování; nemohl sdílet data či komentovat. Pro tento typ webu nebylo potřeba, aby uživatel měl znalosti o fungování a designu, vše bylo spravováno úzkou skupinou lidí. V případě nahrání nových dat administrátorem bylo třeba obnovit stránku, na základě toho lze konstatovat, že probíhal jednosměrný provoz od administrátora k uživateli. (Jacksi & Abass, 2019)

3.2 WEB 2.0

Druhá verze internetu, známá jako Web 2.0, byla formálně představena společností O'Reilly Media. Tato verze využívala nové metody pro užívání internetu, přičemž důraz byl kladen na uživatelskou účast a takzvanou obousměrnost toku dat. Uživatelé měli možnost komunikovat s ostatními uživateli a interagovat s obsahem, což představovalo značný posun od pasivního čtení k aktivní účasti. Jednou z nevýhod se stala hrozba hackerských útoků a s tím spojen únik osobních údajů. Technologická infrastruktura Webu 2.0 byla založena na pravidlech RSS, RDF, XML, DOM a REST. (Jacksi & Abass, 2019)

3.3 WEB 3.0

Web 3.0 je současnou verzí internetu, která začala kolem roku 2014 a je známá jako interaktivní web, poskytující uživatelům možnost dynamické interakce s aplikacemi. Tento web je občas označován také jako „sémantický web" s důrazem na personalizaci. Reprezentuje přestavbu internetu s cílem udělat počítače efektivnějšími v interpretaci nových dat a chápat je jako člověk. Jeho cílem je přeměnit web do jedné velké databáze, kde počítače analyzují a prezentují informace bez použití tradičních vyhledávacích metod. Úspěšným příkladem je zde Google. Technologie užité pro jeho vývin jsou HTML5, CSS, Javascript Web 3.0 má mnoho benefitů, ale i své nevýhody, mezi něž patří serverová náročnost, komplexita, identifikace uživatele a složitost vývoje. (Jacksi & Abass, 2019)

4 HTML

HTML neboli Hypertext Markup Language je značkovací jazyk pro web, který definuje strukturu webových stránek. Zbylé technologie mimo HTML jsou obecně užívány, buď k vzhledu a prezentaci webové stránky v podobě CSS nebo k jeho chování a funkcionalitě v tomto případě se jedná o JavaScript.

Slovo „Hypertext" odkazuje na spojení mezi webovými stránkami, které mohou být uvnitř jedné stránky nebo mezi různými stránkami na internetu. Odkaz je tím pádem základním principem webu. (HTML: HyperText Markup Language | MDN, 2023)

4.1 Verze HTML

4.1.1 HTML 1.0

Jedna z prvních verzí HTML byla vydána na počátku 90.let 20.století, stala se průkopníkem vytváření webových stránek. Tato verze HTML přinesla základní strukturální nástroje pro vytváření obsahu na webu, ale byla omezená ve svých možnostech stylizace a úprav. Ve své době nebyla prezentace obsahu prioritou, a proto se HTML 1.0 zaměřovalo spíše na strukturu a obsah.

Základní charakteristiky této verze zahrnovaly:

1. Struktura elementů: Zavedení elementů umožnilo přidávání obsahu, včetně nadpisů, odstavců, seznamů a obrázků.
2. Jednoduchost: HTML 1.0 neumožňovalo složité stylování nebo úpravy obsahu, což omezovalo možnosti kontroly vzhledu stránek.
3. Podpora fontů: Byla zavedena podpora různých fontů, avšak s velkými omezeními týkajícími se velikosti a stylů.
4. Priorita obsahu: Spojeno s technologickými omezeními, jako byla pomalá internetová připojení a omezené možnosti prohlížečů. (Pedamkar, 2023)

4.1.2 HTML 2.0

Vydaná verze přinesla podstatná vylepšení oproti předchozí verzi.

Základní charakteristiky této verze zahrnovaly:

1. Standardizace HTML: Proběhlo vytvoření pravidel a regulací, které webové prohlížeče museli následovat.
2. Formy: Uveden koncept formulářů, jež uživateli poskytoval možnost vložit data na webovou stránku. Jednalo se pouze o textová pole a tlačítka.
3. Tabulky: Zavedení elementu `<table>` umožnilo strukturovat a organizovat tabulková data, což je důležité pro lepší prezentaci informací na webových stránkách.
4. Vytvoření W3C: Vznik pro ochranu před nekonzistencí a vytváření vlastních tagů, díky tomu byla zajištěna konzistence. (Pedamkar, 2023)

4.1.3 **HTML 3.2**

HTML 3.2 byla dalším hlavním nástupcem HTML 2.0 a byla vyvinuta na konci 20.století.

Aktualizované funkce zahrnuté v této verzi jsou:

1. Aktualizované formuláře: HTML 3.2 přinesla lepší způsoby vytváření interaktivních formulářů. Vývojáři mohli vytvářet formuláře, které byly pro uživatele interaktivnější a dynamické.
2. Podpora CSS: Další důležitou funkcí zahrnutou v HTML 3.2 byla podpora CSS (kaskádových stylů). Pomohla designérům zlepšit vzhled webových stránek stylováním a přizpůsobením prvků HTML.
3. Rozšířené funkce pro obrázky: Umožnila lepší kontrolu nad velikostí, zarovnáním a popisy textu obrázků.
4. Rozšířená sada znaků: Rozšíření dostupnosti znaků pro webové stránky. Zahrnovala speciální symboly a mezinárodní znaky. (Pedamkar, 2023)

4.1.4 **HTML 4.01**

HTML 4.01, vydané v roce 1999, přineslo několik pokroků do jazyka HTML.

Zde jsou některé klíčové funkce a tagy přidané v HTML5:

1. Propojení s CSS: Dříve bylo nutné umístit CSS na každou stránku pro aplikaci stylů. S verzí 4.01 mohly být CSS soubory propojeny do každé HTML stránky pomocí tagu `<link>`. To pomohlo udržovat konzistentní bez opakování kódu CSS.
2. Tags: HTML 4.01 také představilo několik nových tagů jako `<fieldset>`, `<header>`, <footer> a `<legend>`. Tyto tagy zlepšily prezentaci obsahu.
3. Vylepšení tabulek: Vytvoření atributu jako `colspan` a `rowspan`. Tato úprava usnadnila vytváření složitějších tabulek. (Pedamkar, 2023)

4.1.5 **HTML5**

Vydán byl první návrh HTML5 v roce 2008, ale oficiálně se stal doporučením až roku 2014. Tato verze přinesla rozsáhlou podporu pro nové HTML tagy. Navíc HTML5 poskytoval podporu pro nové prvky formulářů, různých typů a tagy pro podporu geolokace.

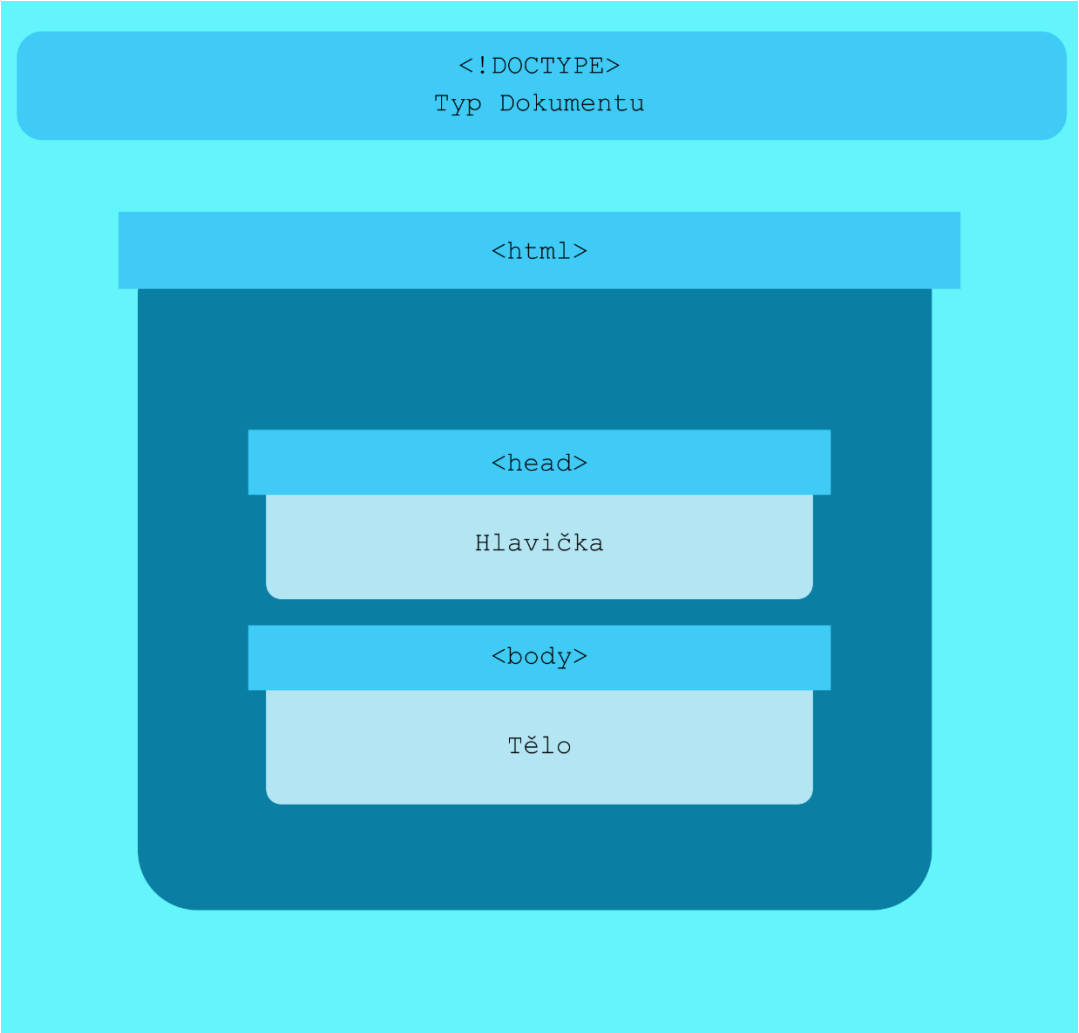
Zde jsou některé klíčové funkce a tagy přidané v HTML5:

1. Formulářové prvky: Jedním důležitým doplněním byl tag `<input type="email">`, který ověřuje, zda je uživatelský vstup platnou e-mailovou adresou. Stejně tak dalším formulářovým prvkem byl tag `<input type="password">`, který byl navržen k bezpečnému zachycení hesel. Prohlížeč zobrazil speciální symboly jako uživatelský vstup v poli pro heslo, čímž chránil heslo před odhalením.
2. Audio: Představen element `<audio>`, která umožňuje vývojářům vkládat zvukový obsah přímo do webových stránek. Tento tag umožnil bezproblémovou integraci zvukových klipů a umožnil přehrávání zvuku přímo na webové stránce.
3. Sémantické značky: Strukturální značky, poskytují organizaci a strukturu HTML stránek. Tyto značky poskytovaly jasnější hierarchii a význam různým částem webové stránky. Několik sémantických značek představených v HTML5 zahrnuje `<figcaption>`, `<header>`, `<footer>` atd. Tyto značky také pomohly zlepšit dostupnost a optimalizaci vyhledávání na webové stránce.
4. Tag section: Tag `<section>` definuje odlišnou sekci v rámci HTML stránky. To pomáhá organizovat a vymezovat obsah a tím poskytuje logické oddělení uvnitř webové stránky. Podpora CSS: Další důležitou funkcí zahrnutou v HTML 3.2 byla podpora CSS (kaskádových stylů). Pomohla designérům zlepšit vzhled webových stránek stylováním a přizpůsobením prvků HTML. (Pedamkar, 2023)

4.2 Struktura HTML

HTML stejně jako jiné jazyky má svá pravidla a ty je nutné dodržovat pro správné fungování. To jinými slovy znamená, že HTML dokument je tvořen určitými bloky, které se nazývají tagy. (Hél, 2022)

Zdroj: vlastní



Obrázek č. 1 Struktura HTML, 2024

4.2.1 DOCTYPE deklarace

V HTML dokumentu je potřeba začít s deklarací značenou `<!DOCTYPE>`. Navzdory podobné struktuře zápisu se nejedná o tag, jedná se o formální zápis, který slouží prohlížeči k identifikaci, o jaký dokument se jedná, a co má očekávat. V současné době se může setkat převážně setkat se zápisem v HTML 5, ale i jinými. (HTML doctype declaration, 2023)

HTML5 syntaxe:

Zdroj: vlastní

data.html

```
<!DOCTYPE html>
```

Obrázek č. 2 Doctype HTML5, 2024

4.2.2 HTML

Tag `<html>` obsahuje celý HTML dokument s jeho elementy. Měl by obsahovat atribut lang pro deklaraci jazyka webové stránky. (Hél, 2022)

Tag `<html>` obsahuje celý HTML dokument s všemi podřízenými elementy, jedinou výjimkou je `<!DOCTYPE>`. V tagu `<html>` je pravidlem zahrnout atribut `lang`, ta dává informaci, v jakém jazyce je webová stránka napsána, tento atribut pomáhá vyhledávacím enginům. (HTML html tag, 2023)

Zdroj: vlastní

```
<!DOCTYPE html>
<html>
  <head> </head>
  <body></body>
</html>
```

Obrázek č. 3 HTML tag, 2024

4.2.3 Hlavička

Tag `<head>` se užívá jako úložiště pro metadata, která jsou nepostradatelná pro HTML dokument, nachází se mezi `<html>` tagem a `<body>` tagem. Data, která obsahuje nejsou přímo zobrazena na stránce. Obsahuje metadata tag a další elementy např.: `<title>`, `<link>`, style nebo script. (Hél, 2022)

Zdroj: vlastní

```
  <head>
    <title>Title</title>
  </head>
```

Obrázek č. 4 Head příklad jednoduchého zápisu, 2024

4.2.4 Tělo

Tag `<body>` definuje viditelný obsah HTML dokumentu, vně tagu `<body>` dále najdeme základní elementy jako nadpisy, paragrafy, obrázky a další. V každém HTML dokumentu se může vyskytovat pouze jeden `<body>` element. (HTML body tag, 2023)

Zdroj: *vlastní*

```
<body>
  <p>This is a common paragraph.</p>
  <h3>This is a common heading.</h3>
</body>
```

Obrázek č. 5 Body příklad jednoduchého zápisu, 2024

4.3 Accessibility

„Síla webu je v jeho univerzalitě. Přístup pro všechny nezávisle na postižení je jeho hlavním aspektem.“ (Initiative, W. W. A., 2023)(překlad vlastní)

Web je od počátku navrhován tak, aby byl přístupný pro lidi, bez ohledu, jakým jazykem mluví, kde žijí, jaký užívají počítač či na jejich schopnostech. V případě, že web plní tento cíl, je dostupný i lidem s různými úrovněmi motoriky, sluchu, vizuálními a kognitivními schopnostmi. Accessibility neboli přístupnost je tedy klíčovým faktorem pro vývojáře a organizace, které chtějí vyvíjet vysoce kvalitní webové stránky, aplikace a webové nástroje. Jejím cílem je zajistit, aby někteří lidé nebyli odříznuti od možnosti využívat jejich produktů a služeb (Initiative, W. W. A., 2023)

4.3.1 Testování Accessibility

Testování přístupnosti webových stránek je klíčovou součástí vývoje, aby se zabránilo vzniku nedostupných webových stránek, které by mohly být problematické jak pro uživatele s různými typy omezení, tak i pro běžné uživatele. Při vývoji nebo redesignu je vhodné vyhodnotit přístupnost na začátku vývoje, abychom mohli identifikovat problémy s přístupností a vyřešit je.

Existuje mnoho pomůcek na vyhodnocení přístupnosti, ale žádná není schopna samostatně rozhodnout, zdali webová stránka či aplikace splňuje standardy přístupnosti. Konečné rozhodnutí je tedy na člověku. (Initiative, W. W. A., 2023)

4.3.1.1 Nástroje testování

Nástroje pro testování webové přístupnosti jsou softwarové programy a online služby, které nám pomáhá rozhodnout, zdali testovaný subjekt splňuje standardy přístupnosti. (Initiative, W. W. A., 2023)

5 CSS

Kaskádové styly (CSS) jsou jazykem stylů, který se používá k popisu prezentace dokumentu napsaného v jazyce HTML nebo XML. CSS popisuje, jak mají být prvky zobrazeny na obrazovce, na papíře, v řeči nebo na jiných médiích. Jazyk CSS patří mezi základní jazyky webu a je standardizován ve všech webových prohlížečích podle specifikací W3C. (CSS: Cascading Style Sheets, 2024)

5.1 Historie CSS

Roku 1994 webová stránka vypadala jako obyčejný list papíru o bílé a černé, protože v té době chyběla možnost měnit styl a zobrazení dokumentu. To si mnoho vývojářů a zástupců firem uvědomovalo, a tak probíhaly snahy o zavedení určitého standardu. Z důvodu zájmu mezi členy W3C byla na konci roku 1995 představena idea o cíli dát kaskádové styly jako doporučený standard k užívání. W3C organizace doporučila používání CSS následujícího roku 1996 a vytvořila pracovní skupinu, která měla určovat další směřování CSS a jeho standardů. (Bos Bert, 2016)

5.2 Výzvy spojené s údržbou a správou CSS kódu

1. Globální selektory:

Tím je myšleno, že selektory v CSS mají globální rozsah působení. To znamená, že stylování definované pomocí selektorů může ovlivnit více prvků na stránce, než je původně zamýšleno. Styly se mohou zdát dobře strukturované a organizované, ale stále mohou mít dopad na zbytek stránky, to může dále vést k neočekávanému chování nebo vzhledu. S tímto chováním selektorů se dostáváme do konfliktu se správou a udržitelností v kódu.

2. Definice závislostí:

Termínem je myšleno, že v CSS je obtížné jednoznačně určit, na čem přesně daný prvek závisí v rámci stylování. Tato nejednoznačnost znamená, že není snadné identifikovat, jaké konkrétní vlastnosti či styly jsou přiřazeny danému prvku a jakým způsobem se tento prvek váže na ostatní části kódu. Tento nedostatek jasnosti a přesnosti může vést k problémům při úpravách a správě kódu, zejména v rozsáhlých projektech s mnoha různými styly.

3. Eliminace mrtvého kódu:

Kaskádová povaha ztěžuje identifikaci a odstranění nepoužívaného CSS, které se vztahuje k dané komponentě. Nepoužívaný kód vzniká na základě pravidelného zasahování do struktury napsaného kódu z důvodů změn ve funkcionalitě a designu, které následně nebyly odstraněny ze souboru.

4. Problémy s miniaturizací:

Minifikace selektorů je náročná, dochází k odlehčení souboru se CSS. Při nevhodném použití se nám může neočekávaně zhoršit výkonost. Minifikací následně dochází ke zhoršení čitelnosti, udržitelnosti a konzistenci.

5. Sdílení konstant:

Dalším problémem, se kterým se setkáme je sdílení konstant. Jedná se například o sdílení barev, fontů a dalších vlastností, které jsou užity na různých místech aplikace. Jejich nedostatečnou správou dochází k redundanci vlastností, která ovlivňuje chování a design aplikace.

6. Nedeterministické rozlišení:

Tento problém vzniká, při načítání webové stránky, kdy se styly neaplikují postupně, a to může mít za následek aplikace špatného stylu. (Bertoli, 2017, stránky 151-152)

7. Nedostatečná izolace:

Nedostatek izolace v CSS komplikuje předvídání stylů kvůli globálním selektorům. V takovém případě může nastat situace, kdy styly elementu byly ovlivněny již napsaným CSS v jiném místě (Bertoli, 2017, stránky 151-152)

5.3 SASS

Sass je jazyk pro tvorbu stylů, který je zkompilován do jazyka CSS. Umožňuje používat proměnné, vnořená pravidla, mixiny, funkce a další prvky, a to vše s plně kompatibilní syntaxí CSS. Sass pomáhá udržovat velké soubory stylů přehledné a usnadňuje sdílení návrhu v rámci projektů i mezi nimi. (SASS: Documentation, 2023)

5.3.1 Předběžné zpracování nebo Preprocessing

Sass vezme váš předzpracovaný soubor a převede ho na běžný CSS, který lze použít na webových stránkách. Nejjednodušší způsob, jak to udělat, je použít terminál v editoru. Zadá se příkaz Sass, určí se vstupní a výstupní soubor. Sass zkompiluje vstupní soubor do výstupního souboru. Sass umožňuje kontrolovat změny ve zdrojových souborech a automaticky

aktualizovat, a to pomocí příkazu --watch. Tímto způsobem se váš kód stává více efektivním. (SASS: Basics, 2023)

Zdroj: vlastní

```
sass --watch priklad.scss vysledek.css
```

Obrázek č. 6 Příklad Sass kompilace souboru při watch módu, 2024

5.3.2 **Proměnné nebo Variables**

O proměnných se uvažuje jako o způsobu ukládání informací, které chcete opakovaně používat v celém souboru stylů. Můžete ukládat například barvy, typy písma nebo jakékoli jiné hodnoty CSS, o kterých si myslíte, že je budete chtít opakovaně používat. Sass využívá symbol $ k tomu, aby se hodnota stalo proměnnou. (SASS: Variables, 2023)

SCSS syntaxe:

Zdroj: vlastní

```
$font-stack: Helvetica, Arial, sans-serif;
$primary-color: #333;

    body{
        font: 100% $font-stack;
        color: $primary-color;
        }
```

Obrázek č. 7 Zápis SCSS Variables pro body, 2024

CSS syntaxe:

Zdroj: vlastní

```
body{
    font: 100% Helvetica, Arial, sans-serif;
    color: #333;
    }
```

Obrázek č. 8 Zápis obrázku č. 7 v CSS, 2024

Při zpracování Sass vezme proměnné, které jsme definovali pro $font-stack a $primary-color, a vypíše normální CSS s hodnotami našich proměnných umístěnými v CSS. (SASS: Variables, 2023)

5.3.3 **Hnízdění nebo Nesting**

Nesting je způsob zápisu CSS v preprocesoru SASS, který umožňuje vnořovat selektory CSS tak, aby dodržovaly vizuální hierarchii podobnou HTML. Je důležité si však uvědomit, že příliš hluboké vnořování pravidel může vést k nadměrné kvalifikaci CSS, což ztěžuje jeho udržování a čtení. Tento přístup je obecně považován za špatnou praxi, a proto je důležité používat hnízdění s rozvahou a zváženě. (SASS: Basics, 2023)

Hnízdění příklad:

Zdroj: vlastní

```
nav {
  ul {
    list-style: none;
    padding: 0;
    margin: 0;

    li {
      display: inline-block;

      a {
        text-decoration: none;
        color: #333;
        padding: 10px;
      }
    }
  }
}
```

Obrázek č. 9 Nesting, 2024

5.3.4 Částečné náhrady nebo Partials

S pomocí Partials můžete vytvářet soubory Sass, které obsahují malé úryvky CSS, jež lze následně zahrnout do jiných souborů Sass. Tento přístup usnadňuje modularitu CSS a zlepšuje údržbu. Partials jsou pojmenovány s podtržítkem na začátku, například _partial.scss. Tímto způsobem Sass ví, že se jedná o Partials soubor a neměl by být samostatně zkompilován do souboru CSS. K zahrnutí Partials do jiného souboru se používá pravidlo @use. (SASS: Basics, 2023)

5.3.5 Mixins

Mixiny jsou v preprocesorech CSS, jako je SASS, mocným nástrojem pro znovupoužití kódu a zabránění duplikací. Jsou to bloky kódu obsahující různé vlastnosti, které lze jednoduše znovu použít v různých částech projektu. Výhody mixinů tedy jsou zvovupoužitelnost, udržitelnost a prevence duplikace, jsou tak silným nástrojem pro organizaci a správu stylů, a tím mohou výrazně zlepšit efektivitu. (SASS: Basics, 2023)

5.3.5.1 Tvorba mixinu

Pro vytvoření mixinu v SASS je třeba použít syntaxi `@mixin` následovanou jménem mixinu s potřebnými vlastnostmi. (SASS: Basics, 2023)

Zdroj: vlastní

```
@mixin space {
/* Deklarace vlastností */
  padding: 0.5rem 1rem;
  margin: 1rem;
}
```

Obrázek č. 10 Vytvoření mixinu Space, 2024

5.3.5.2 Použití mixinu

Mixin lze poté jednoduše použít v jiných částech kódu pomocí syntaxe `@include` následované jménem mixinu. (SASS: Basics, 2023)

Zdroj: vlastní

```
    /* Použití mixinu */
    .selector {
    @include space;
}
```

Obrázek č. 11 Příklad reusability u mixinu, 2024

6 JavaScript

JavaScript (JS) je lehce interpretovatelný programovací jazyk s prvotřídními funkcemi. Je to skriptovací jazyk pro webové stránky, používá ho i mnoho jiných prostředí než prohlížeč, například Node.js, Apache CouchDB a Adobe Acrobat. JavaScript je prototypový, víceparadigmový, jednovláknový, dynamický jazyk podporující objektově orientované, imperativní i deklarativní styly psaní kódu. (JavaScript | MDN, 2023)

Potenciální nevýhodou JavaScriptu však může být jeho volná typizace, což v důsledku znamená sice vysokou flexibilitu při vývoji, avšak za vysokou cenu při opravách bugů. (Yang, 2018)

JavaScript také zahrnuje dynamické možnosti jazyka například konstrukci objektů za běhu, seznamy parametrů proměnných, proměnné funkcí, dynamické vytváření skript, introspekci objektů a obnovu zdrojového kódu. (JavaScript | MDN, 2023)

JavaScript dále zajišťuje dynamiku a interakci s webovými stránkami, kdy dokáže například měnit jejich obsah, vzhled, vytváří nové elementy nebo naslouchá jiným změnám v prohlížeči. Toho všeho je schopen na základě možnosti aktivně manipulovat s DOMem. (Ayebola, 2023)

Standardy pro JavaScript jsou specifikace jazyka ECMAScript (ECMA-262) a specifikace ECMAScript Internationalization API (ECMA-402). (JavaScript | MDN, 2023)

6.1 TypeScript

TypeScript je nadstavba jazyka JavaScript určená ke zjednodušení vývoje aplikací. To ve zkratce znamená, že můžeme používat všechny dílčí vlastnosti JavaScriptu, ale s možností typizace. Díky této vlastnosti předcházíme neočekávaným jevům, zajišťujeme kvalitní, čitelný a škálovatelný kód.

TypeScript je schopen inference neboli dedukce, to znamená, že je schopen přiřadit správný typ k proměnné. To znamená, že TypeScript rozumí tomu, jak se s proměnnou může dále pracovat. V případě nepovolené manipulace zakročí, upozorní nás na chybu a neproběhne kompilace.

Definování typů je jedna z dalších schopností TypeScriptu, díky ní je možné vytvářet vlastní struktury. Tato možnost je zejména užitečná v situacích, kdy inference neproběhne a nedokáže správně určit typ proměnné.

Kromě běžných primitivních typů, které jsou dostupné i v běžném Javascriptu. TypeScript přichází s novými typy, a to any, never, unknown a void. Tyto typy rozšiřují možnosti typizace a přináší další flexibilitu do TypeScriptu.

S použitím podporovaných rozšíření můžeme vytvářet vlastní typové struktury, což je užitečné v situacích, kdy standardní typy nevyhovují. Jedná se o kompozitní typy generik a unionů. (Documentation - TypeScript for JavaScript Programmers, 2024)

6.2 Framework

Framework je nástroj, který je vyvinutý nad konkrétním programovacím jazykem, který nám usnadňuje práci s kódem, přidává nám mnoho funkcionalit, konceptů a pomáhá psát jednodušší a čistší kód, který lze lehce testovat a opravovat chyby. Framework je vlastně konfigurovatelná šablona, kterou nastavíme podle našich požadavků. Můžeme je rozdělit podle našich potřeb na dva typy, a to Front-end Frameworky a Back-end Frameworky. Front-end Frameworky slouží k vývoji UI[1], které je zobrazeno uživateli. Front-end frameworky jsou React.js, Vue.js, a Angular. Back-end Frameworky jsou užívány jako server-side aplikace, mají na starosti requesty, komunikace s API[2] a databází. Back-end Frameworky jsou RUBY a Django. (What is a Framework? - GeeksforGeeks, 2023)

Je důležité si uvědomit, že neexistuje žádný univerzálně dokonalý JavaScriptový framework. Není důvod se zabývat otázkou, proč zvolit konkrétní framework, možná nebude vůbec potřeba žádný framework. Nejlepší přístup může být nejprve si promyslet a rozhodnout se, co přesně potřebujeme k dosažení našich cílů. (Godbolt, 2016, stránky 45-46)

[1] User Interface – uživatelské rozhraní

[2] Application Programming Interface – Aplikace programovacího prostředí

6.2.1 **React**

React je open-source Javascript knihovna, která byla vyvinuta společností Facebook, která chtěla zjednodušit komplikovaný proces vývoje webových stránek. A to tím způsobem, že webovou stránku složíme z malých kousků, které dělají určitou činnost, tak jako puzzle. (Herbert, 2023)

Běžná webová stránka před Reactem fungovala způsobem, že jsme měli otevřené své pracovní okno a na něm jsme vytvářeli, co potřebovali, a když jsme činnost dokončili zmáčkli jsme na tlačítko a stránka nás přesměrovala na jinou. Oficiálně se tento stav nazývá request neboli žádost. Na základě toho si dokážeme jednoduše představit, co vznikalo za problém, stránky byly pomalé a při změně dat se celý obsah stránky opět načítal. (Herbert, 2023)

React se k řešení tohoto problému postavil jinak, tím že navrhl koncept známý jako single-page application (SPA). SPA načte jeden dokument HTML při requestu, dále aktualizuje pouze část kódu, ve které skutečně proběhla změna. Tomuto způsobu prezentace obsahu se říká client-side routing, proto i v případě nějaké změny se stránka znovu nezačne načítat. To má za následek lepší výkonost a příjemnější uživatelskou zkušenost. (Herbert, 2023)

V jednoduchosti řečeno React je schopný zjistit, jak provést úpravu bez toho, abychom si povšimli nějakého zásahu do webové stránky. (Herbert, 2023)

6.2.1.1 **React Komponenta React Props**

Komponenty v Reactu jsou pro nás nástroje, pomocí nichž dokážeme vystavět složitou aplikaci. Jedná se izolované, znovupoužitelné a nezávislé kusy kódu, které mají stejnou podstatu tak jako funkce v Javascriptu. Komponenty se rozdělují na dva různé typy, a to Class komponenta a Function komponenta. Rozdíl mezi nimi se dá popsat takto: Class komponenta má složitější strukturu zápisu, a proto je méně používaná. Function komponenta je přehlednější a k zapsání je potřeba daleko méně kódu. (legacy.reactjs.org, 2022)

Props je způsob pomocí, kterého mezi sebou komunikují komponenty, rodič neboli parent předává data, která potomek neboli children přijme. Props nejsou narozdíl

od klasických HTML atributů omezeny typem. Jedná se o základní způsob šíření dat v Reactu. (Passing props to a component – react, 2023)

6.2.1.2 **Životního cyklus**

Komponenty mají životní cyklus, který se dělí na fáze. Metody se v komponentu v určitý specifický moment volají, a to je odvislé od fáze cyklu. Tyto metody nám poskytují kontrolu nad komponentem. (Gatwiri , 2023)

Metody Životního cyklu v Class Komponentě:

1. componentDidUpdate(): Provede se po dokončení metody render() a aplikaci nových změn na původní model DOM.
2. componentDidMount(): Metoda proběhne v moment, co se vloží do DOMu.
3. componentWillUnmount(): Tato metoda se provede těsně před odebráním komponenty z DOM. (Stefanov, 2016, str. 60)

6.2.1.3 **JSX**

JSX je syntaxe zápisu pro Javascript, který má podobnu strukturu jako HTML. (Writing Markup with JSX – React, 2023) Má otevírající a zavírající tag k tomu, aby lépe reprezentoval vnořenou strukturu elementů, která by byla nečitelnou a nespravovatelnou v čistém JavaScriptu. (Bertoli, 2017, str. 21) V současné době se jedná o nejběžnější styl zápisu, a to kvůli jeho přehlednosti. (Writing Markup with JSX – React, 2023)

7 Design systémy

Design systém je programová reprezentace vizuálního jazyka webových stránek. Vizuální jazyk vytvořený designéry vyjadřuje, jakým způsobem webové stránky vizuálně sdělují své poselství uživatelům. Je to soubor barev, fontů, tlačítek, stylů obrázků, typografie a vzorů uživatelského rozhraní, které se používají k vyjádření nálady, významu a záměru. Stejně jako mluvený jazyk lze rozložit na menší dílky, tak i vývojáři webových stránek rozloží vizuální jazyk na menší části. Díky tomu můžeme vytvořit dále pravidla, jak jej opět složit dohromady. Cílem této myšlenky je vytvořit kód, který je škálovatelný a udržitelný a který

věrně reprodukuje veškeré možnosti, které daný jazyk umožňuje vyjádřit. (Godbolt, 2016, str. 27)

7.1 Atomický design

Atomický design je metodologie využívaná k vytváření designových systému webových stránek. Místo designu celé webové stránky najednou, atomický design rozdělí na kousky frekventovaně užívané elementy do různě velikých částí, postupným procesem takto kombinuje stavební bloky, až se zpětně objeví a zformuje celá stránka. Důležité je, že nejmenší vytvořené elementy tímto způsobem jsou nedělitelné. (Godbolt, 2016, str. 155)

Jednotlivé úrovně atomického designu:

7.1.1 Atomy

Jsou to základní stavební prvky UI, jako jsou tlačítka, vstupní pole, nadpisy, ikony atd. Tyto prvky jsou nejmenšími stavebními bloky a jsou navrhovány tak, aby byly znovupoužitelné a nezávislé na kontextu. (Frost, © 2016)

7.1.2 Molekuly

Jsou kombinace atomů, které tvoří jednoduché funkční bloky. Například pole pro vyhledávání s tlačítkem, seznam položek s textem a ikonami, nebo navigační prvek s odkazy a ikonami. Molekuly mohou být znovupoužívány a kombinovány k vytvoření komplexnějších komponent. (Frost, © 2016)

7.1.3 Organismy

Jsou složitější struktury, které obsahují kombinace atomů a molekul. Organismy jsou obvykle komplexními oblastmi rozhraní, jako je záhlaví, zápatí, menu nebo karty obsahu. Tyto struktury se mohou skládat z různých molekul a atomů a často reprezentují samostatné funkční bloky. (Frost, © 2016)

7.1.4 Šablony

Jsou rámce, které definují strukturu stránek nebo rozložení určité oblasti rozhraní. Šablony určují, jak jsou organismy, molekuly a atomy uspořádány na konkrétní stránce nebo v konkrétním kontextu. (Frost, © 2016)

7.1.5 **Stránky**

Jsou konečné výsledky, které vzniknou kombinací šablon s reálnými daty. Stránky jsou konkrétními výsledky designu a reprezentují kompletní uživatelské rozhrání. (Frost, © 2016)

7.2 **Checklist CSS architektury**

Existuje seznam osvědčených postupů a pokynů, které musíme dodržovat, abychom zajistili škálovatelnost a udržovatelnost našeho systému. (Frantz, 2021)

7.2.1 **Organizovanost**

Musíme být organizovaní, to znamená, že potřebujeme mít pevnou strukturu kódu, pevné pokyny, kterými se řídíme, a konzistentní kód. (Frantz, 2021)

7.2.2 **Specifičnost CSS**

Vyřešit všechny problémy se specifičností CSS. Například CSS z jedné komponenty by nemělo být v rozporu s CSS v jiné komponentě. Musíme se ujistit, že naše kód správně odráží zásady atomického návrhu, i kdyby šlo jen o způsob pojmenování složek. (Frantz, 2021)

7.2.3 **Pochopitelnost**

Kód, který píšeme, v tomto případě zejména CSS, musí být snadno pochopitelný. A toho můžeme dosáhnout tím, že se ujistíme, že máme vhodné komentáře, proměnné a vysvětlení rozhodnutí, která jsme učinili při vývoji. (Frantz, 2021)

7.2.4 **Flexibilita**

Je nezbytné zajistit, aby náš kód CSS byl plně flexibilní a snadno upravitelný. To znamená, že pokud dojde ke změně barevného schématu nebo typografického fontu v rámci konkrétního produktu nebo projektu, naše styly CSS by měly být schopny se těmto změnám automaticky přizpůsobit. Tímto způsobem zůstane naše prezentace konzistentní a relevantní s aktuálními. (Frantz, 2021)

7.2.5 **Opakovaně užitelný kód**

Přepis CSS byl opakovaně použitelný napříč týmy a projekty. S největší pravděpodobností bychom měli mít naše CSS jako externí balíček NPM, který kdykoli stáhneme pro libovolné projekty nebo libovolný tým. (Frantz, 2021)

8 Testing

Testing je proces, při kterém se snažíme vyzkoušet funkcionality programu. V průběhu procesu testování se hledají chyby a mezery v kódu, proto abychom je mohli odstranit dříve, než se dostanou do produkce. (Yasar, 2022)

Důvody pro užívání testování jsou:

1. Brzké identifikování chyby: V případě, že vytváříme plnohodnotnou aplikaci může docházet k chybám, tomu se snažíme předcházet s pomocí testování, které slouží právě k detekci takových chyb.
2. Zlepšení kvality: Při testování zajišťujeme, že produkt bude splňovat stanovená kritéria a specifikace, tím máme kontrolu nad kvalitou produktu.
3. Úspora nákladů: Testováním můžeme zajistit to, abychom v budoucnu po uvedení produktu neměli dodatečné náklady a velikou ztrátu při řešení chyb, kterým jsme mohli předejít při vývoji.
4. Podpora škálovatelnosti: Testovaní nám pomáhá zjišťování výkonnostních limitů aplikace, jak pracuje při vyšší zátěži a objemu transakcí.
5. Bezpečnostní rizika: Pomocí testování dokážeme najít, identifikovat zranitelné části aplikace a odstranit je. (Yasar, 2022)

8.1 Typy testování

8.1.1 Unit Testing

Unit testing je způsob testování malých částí v aplikaci. Běžně jej provádějí developeři, dělají se s účelem kontroly jednotlivých částí kódu, jestli pracují, jak mají. Jedná se především o automatizované testy a jsou designované speciálně na testování kódu, jako jsou metody a funkce tedy malé bloky. Unit testy jsou prováděny na nejnižším stupni vývoje. (GfG, 2024)

Výhody unit testingu:

1. Změnou v kódu nevytvoříme nový bug.
2. Pomáhá nacházet a identifikovat bugy brzy při vývoji, tím se ulehčuje jejich oprava.
3. Kód je jednodušší, lépe se chápe a pracuje se s ním.
4. Výrazně zlepšuje kvalitu a spolehlivost vyvíjeného softwaru.

(GfG, 2024)

8.1.2 Performance Testing

Důvodem jakéhokoliv testu je ochranou uživatele před špatnou zkušenosti, web se slabým výkonem je jedním z nejvýraznějších způsobů, jak takovou zkušenost ukázat.

Performance testing měří nejdůležitější hodnoty, které ovlivňují schopnosti uživatele k aktivní práci s webovou stránkou, jedná se například o page weight, number of requests, time to first byte (TTFB). (Godbolt, 2016, str. 111)

8.1.2.1 Page Weight

Webové stránky postupně rostou, a to má neblahý vliv na výkon, mezi roky 2011 a 2015 se průměrná hmotnost stránky zvýšila přibližně o 2,7x násobek své původní váhy. To má za následek zvýšení doby načítání, proto je vhodné hledat způsoby snížení hmotnosti. (Godbolt, 2016, stránky 113-114)

8.1.2.2 Number of Request

Prohlížeč při načítání webové stránky potřebuje projít s HTTP requesty všechny složky, to může mít za následek vysoké množství requestů. V případě, že máme vysoké množství

složek, do kterých je třeba vstupovat dochází k zpomalení sítě. To v jednoduchosti znamená, že čím méně requestů potřebujeme, tím máme lepší výkon. (Godbolt, 2016, str. 115)

8.1.2.3 **TTFB neboli „čas do prvního bytu“**

TTFB je doba, která určuje, jak dlouho trvá, než se požadavek z prohlížeče dostane na webovou stránku a zpět v podobě prvního bytu dat. (Godbolt, 2016, str. 116)

8.1.2.4 **Time To Start Render**

Jedná se o naměřenou hodnotu, za kterou uživatel uvidí vykreslení obsahu na stránce. Volně řečeno to znamená, že soubory se načetly a prohlížeč je již dokáže je zobrazit v DOM. Výkon lze vylepšit například pomocí lazy-loadingu, s prací s obrázky a jejich převodem na URI či umístit CSS do hlavičky stránky. (Godbolt, 2016, str. 116)

9 Základní nástroje potřebné pro vývoj

9.1 Node

Node.js je open-source a multiplatformní prostředí pro běh JavaScriptu. Jedná se o populární nástroj pro širokou škálu projektů. Node.js používá běhové prostředí JavaScriptu enginu V8, a tak dosahuje vysoké výkonnosti.

Aplikace v Node.js běží v jednom procesu, aniž by vytvářela nové vlákno pro každý požadavek. Node.js poskytuje sadu asynchronních I/O primitiv ve své standardní knihovně, což zabrání blokování JavaScriptového kódu. Knihovny v Node.js jsou psány s využitím neblokujících paradigmatu. Při provádění I/O operace nedochází k blokování kódu. Po odeslání požadavku se kód opět spustí, až když přijme odpověď. To umožňuje zpracovávat vysoké množství souběžných připojení se serverem.

Node.js má faktickou výhodu, protože vývojáři front-endu, kteří píší JavaScript pro prohlížeč, jsou schopni psát kód na straně serveru stejně jako kód na straně klienta bez nutnosti přecházet na zcela odlišný jazyk. (Introduction to Node.js, 2023)

9.1.1 **Npm**

NPM (Node.js package manager) je nástroj pro správu balíčků v jazyce JavaScript, který je nezbytný pro vývoj aplikací v prostředí Node.js, kterého je součástí a není nutné jej instalovat separátně. Jeho hlavním účelem je usnadnit instalaci, aktualizaci a odstranění balíčků, které jsou potřebné pro vývoj a běh JS aplikací. (Node.js — An introduction to the npm package manager, 2023)

Funguje to tak, že uživatelé mohou prostřednictvím npm vyhledávat, stahovat a instalovat balíčky z veřejného npm registru, který obsahuje obrovské množství volně dostupných balíčků. Balíčky mohou obsahovat různé užitečné knihovny, frameworky, nástroje a další zdroje, které jsou k dispozici pro jazyk JavaScript. (Thompson, 2020)

Npm také umožňuje správu dependencí projektu, což znamená, že můžete definovat, které balíčky jsou potřebné pro vaši aplikaci a npm se postará o jejich automatickou instalaci a správu. To výrazně usnadňuje proces vývoje a udržování aplikací. Kromě toho npm poskytuje funkce pro správu verzí balíčků, sdílení balíčků s ostatními uživateli nebo týmy, vytváření a správu organizací a další pokročilé funkce, které usnadňují práci s balíčky. Npm je tedy klíčovým prvkem ekosystému Node.js a je nezbytný pro vývoj JS aplikací. (Luke, Loginov, & Thompson, 2023)

9.1.1.1 **Architektura**

V Node.js NPM ekosystému se dependencies ukládají do adresáře node_modules v projektu. Problém však může nastat se strukturou složek, kdy se duplikují dependencies v projektu. Npm instaluje balíčky nedetermisticky. To znamená, že struktura adresáře node_modules se může lišit projekt od projektu. Následkem toho se setkáváme s bugy, které nám může dlouho trvat vyřešit. (Nakazawa, McKenzie, & Kyle, 2018)

9.1.1.2 **Instalace npm**

Instalace npm do projektu je velice jednoduchá stačí napsat do CLI[3] v kódový editor.

Zápis příkazu má strukturu:

[3] Command Line Interface

`npm install` (Node.js — An introduction to the npm package manager, 2023)

9.1.1.3 **Instalace balíčku**

Instalace balíčku má strukturu velice intuitivní, kdy nejprve do CLI uvedete `npm` následované názvem balíčku, který chcete instalovat do svého projektu. Je přitom možné instalovat přímo i verzi balíčku, který chcete nainstalovat do projektu.

Zápis příkazu má strukturu:

```
npm install <package-name>
```

`npm install <package-name>@<version>` (Node.js — An introduction to the npm package manager, 2023)

9.1.1.4 **Update npm**

V případě potřeby je vhodné v projektu provést aktualizaci pomocí příkazu update. Příkaz update zkontroluje, zda nebyla vydána novější verze některého z používaných balíčků, a pokud ano, a nainstaluje ji.

Zápis příkazu má strukturu:

`npm update` (Node.js — An introduction to the npm package manager, 2023)

9.1.1.5 **Update balíčku**

V určitých případech je vhodné či nutné určitý balíček aktualizovat, a to z mnoha různých důvodů. Může se jednat o konec podpory určitých funkcí, oprava předchozích chyb a vylepšení.

Zápis příkazu má strukturu:

`npm update <package-name>` (Node.js — An introduction to the npm package manager, 2023)

9.1.2 **Yarn**

Yarn je balíčkovací systém vyvinutý v roce 2016 společností Facebook pro běhové prostředí programovacího jazyka JavaScript Node.js jako alternativa k npm. Mezi přednosti tohoto balíčkovacího systému patří rychlost, konzistence, stabilita, a bezpečnost.

Tento nástroj byl vyvinut společným úsilím společností Facebook (nyní Meta), Exponent, Google a Tilde s cílem vyřešit potíže s konzistencí, bezpečností a výkonem. (Nakazawa, McKenzie, & Kyle, 2018)

9.1.2.1 **Přidávání souborů**

Yarn přidáme:

```
npm install --global yarn
```

(Yarn, 2022)

Pro vygenerování základního souboru package.json, kde bude seznam všech balíčku, stačí napsat do CLI příkaz `init`.

Zápis příkazu má strukturu:

```
yarn init
```

(Yarn, 2022)

Pro případ, kdy chceme přidat balíček je potřeba napsat příkaz `add` následovaným jménem balíčku.

Zápis příkazu má strukturu:

```
yarn add vasBalicek
```

(Yarn, 2022)

Všechny balíčky, co chceme přidat a zároveň chceme použít jen ve vývoji přidáváme pomoci příkazů `add - D` nebo `add --dev`:

Zápis příkazu má strukturu:

```
yarn add -D vasBalicek
```

(Yarn, 2022)

9.1.2.2 **Architektura**

Yarn řeší problémy spojené s verzováním a nedeterminismem u npm pomocí uzamčených souborů a algoritmu, který je deterministický a spolehlivý. Tyto soubory uzamykají nainstalované dependencies na konkrétní verzi a zajistí, že každá instalace má na všech počítačích stejnou strukturu souborů v adresáři node_modules. Každý soubor používá formát s uspořádanými klíči, aby se zajistilo, že změny v adresáři jsou minimální. (Nakazawa, McKenzie, & Kyle, 2018)

Instalační proces lze rozdělit do tří kroků:

1. Rozlišení: Yarn začíná rozlišovat dependencies tím, že požaduje údaje z registru a rekurzivně prochází každou závislost.
2. Fetching: Yarn hledá v globálním adresáři s mezipamětí, jestli byl balíček už stažený. V případě, že nebyl stažený, tak Yarn stáhne balíček a umístí ho do mezipaměti. Díky tomu může pracovat i offline.
3. Propojení: Yarn propojí vše dohromady, tak že zkopíruje všechny potřebné soubory z mezipaměti do místního adresáře node_modules.

Pomocí rozložení do kroků a užívání deterministického postupu, je Yarn schopen provádět operace paralelně. Paralelní operace využívají maximálně zdroje, což zajišťuje rychlý proces instalace. A zároveň se užívá mechanismu mutex, který zajišťuje, aby se více souběžných akcí v příkazovém řádku nepomíchalo, což by poškodilo soubory. (Nakazawa, McKenzie, & Kyle, 2018)

Mutex jedná se o flag, který zajišťuje, že pouze jedna instance probíhá v jeden moment. (CLI Introduction | Yarn, 2022)

9.2 **Kódový editor**

Kódový editor je hlavním nástrojem každého softwarového vývojáře. Jeho význam spočívá v několika různých věcech. Kódový editor jako celek zpříjemňuje práci s kódem tím, že umožňuje programátorovi pracovat v přívětivém prostředí, kompilovat a interpretovat kód.

Kódový editor je schopen napovídat. Díky tomuto chování se zrychluje vývoj a zároveň snižuje šance na vyskytnutí se chyby v kódu. Významné ulehčení práce v editoru zajišťuje zvýrazňování syntaxe v kódu, to nám pomáhá v lepší orientaci v kódu.

Chyby a varovaní jsou také důležitým znakem v editoru. V případě, že vývojář použije například nesprávnou syntaxi v jazyce, který užívá. Editor nám chybu zvýrazní, bez této metody by hledání problému stálo mnoho úsilí a času.

Zvýraznění závorek je také jedna z důležitých forem pomoci editoru, v zásadě jde o to, že snáze nalezneme druhou závorku a díky tomu můžeme najít chyby, které mohou vzniknout jejich špatným umístěním či úplným postrádáním uzavírající závorky. Závorku si zvýrazníme tím, že na ni klikneme, na to se zobrazí a zvýrazní uzavírající závorka.

Další pomoc při programování je formátování textu do určitých bloků, které slouží ke zvýšení přehlednosti kódu. (Avnishsharamav, 2023)

9.3 Storybook

Storybook je nástroj pro vytváření UI komponent a webových stránek, přičemž jeho hlavní vlastností je izolovanost. (Storybook: Frontend workshop for UI development, 2023) Frameworky JavaScriptu rozložili UI do jednoduchých komponentů, ale to nutně nestačí v případech, kdy vyvíjíme skutečně objemnou aplikaci. Tyto projekty mohou mít tisíce různých variant, a to vede ke špatně a složitě udržovatelnému kódu. Řešením tomu, poté je vytváření vlastních UI komponent, se kterými pracujeme v izolovaném prostředí Storybook. (Why Storybook? • Storybook docs, 2023)

9.4 Rollup.js

Rollup.js je používán jako modulový balíček pro JavaScriptové aplikace, ten pomáhá sloučit kód do jednoho či více souborů. Tento postup usnadňuje správu dependencies a zlepšuje výkon. (Mandal, 2023)

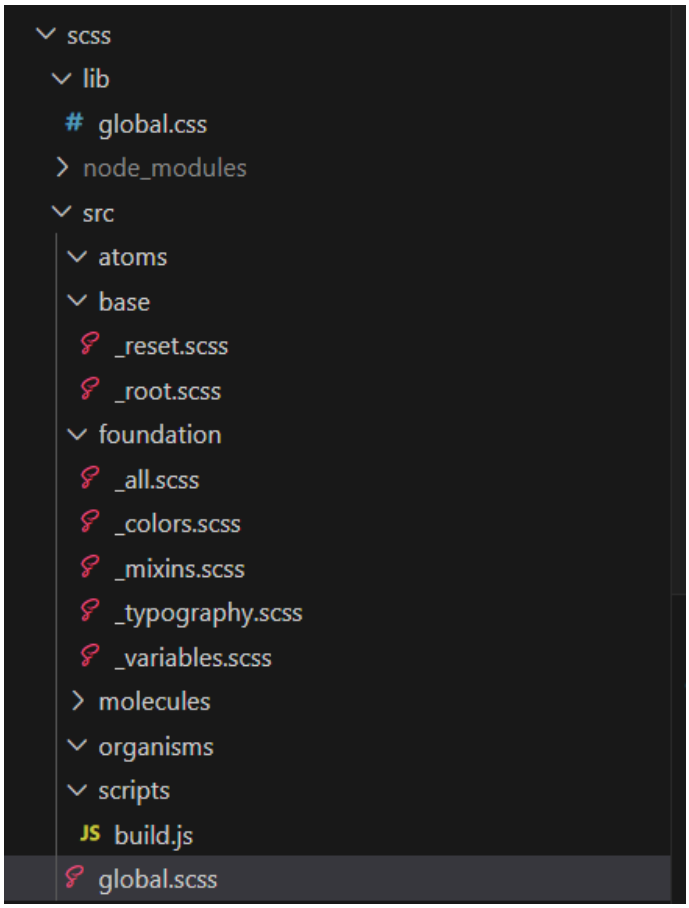
10 Vlastní práce

Praktická část práce pojednává o postupu vývoje atomického designu, seznamuje nás s použitými nástroji a provádí postupné kroky k dosažení cíle, které je vytvořit funkční designový systém konkrétně atomický design.

10.1 Organizace scss souboru

Naši práci vývoje započneme stažením Sass, s jeho pomocí postupně vytvoříme scss soubory podle našich požadavků.

Zdroj: vlastní



Obrázek č. 12 Obrázek složek scss, 2024

Vytvoříme základ pro další použití. Budeme používat soubory ze složky foundation k napsaní stylu pro naše komponenty. Tato složka obsahuje variables, typography, mixins a colors.

Ve složce scripts je soubor build.js, ten potřebujeme, abychom si nadefinovali, jakým způsobem se budou kompilovat naše sass soubory.

```
const Fs = require("fs")
const Path = require("path")
const Sass = require("node-sass")

const getComponents = () => {
  let allComponents = []

  const types = ["atoms", "molecules", "organisms", "templates"]

  types.forEach((type) => {
    const allF const Fs = require("fs")
const Path = require("path")
const Sass = require("node-sass")

const getComponents = () => {
  let allComponents = []

  const types = ["atoms", "molecules", "organisms", "templates"]
iles = Fs.readdirSync(`src/${type}`).map((file) => ({
      input: `src/${type}/${file}`,
      output: `lib/${file.slice(0, -4) + "css"}`,
    }))

    allComponents = [...allComponents, ...allFiles]
  })

  return allComponents
}

const compile = (inputPath, outputPath) => {
  const result = Sass.renderSync({
    data: Fs.readFileSync(Path.resolve(inputPath)).toString(),
    outputStyle: "expanded",
    includePaths: [Path.resolve("src")],
  })

  Fs.writeFileSync(Path.resolve(outputPath), result.css.toString())
}

try {
```

```
  Fs.mkdirSync(Path.resolve("lib"))
} catch (e) {}

compile("src/global.scss", "lib/global.css")

getComponents().forEach((component) => {
  compile(component.input, component.output)
})
```

10.2 Monorepositář

Monorepositář má za cíl konsolidovat a spravovat všechny balíčky, zejména ty, které mají vzájemné závislosti, v rámci jediného úložiště a pracovního prostoru.

Vytvoříme zvlášť package.json pro každý projekt. Naše projekty, které budeme chtít publikovat jsou v package.json souboru. Jedná se konkrétně o react, scss a foundation projekt.

Konfigurace lerna.json:

Zdroj: vlastní

lerna.json

```
{
  "$schema": "node_modules/lerna/schemas/lerna-schema.json",
  "version": "0.0.0",
  "packages": [
    "packages/*"
  ],
  "npmClient": "yarn",
  "stream": true
}
```

Obrázek č. 13 Lerna.json, 2024

Tímto způsobem můžeme z nástroje Lerna spustit pouze jeden skript, který spustí všechny naše balíčky v režimu dev. Následně můžeme automaticky provádět změny a vidět, jak se tyto změny odrážejí ve vývoji, pomoci nodemon.

Pro packages/react:

```
"dev": "yarn build --watch"
```

Pro packages/scss:

```
"dev": "nodemon --watch src --exec yarn build -e scss"
```

Pro packages/foundation:

```
"dev": "yarn build -w"
```

Všechny projekty v development modu spustíme pomoci scriptu:

```
"dev": "yarn lerna run dev"
```

Skripty pro spuštěni projektu v production modu:

Pro packages/react:

```
"build": "rollup -c --bundleConfigAsCjs"
```

Pro packages/scss:

```
"build": "node src/scripts/build.js"
```

Pro packages/foundation:

```
"build": "tsc"
```

Všechny projekty v production modu spustíme pomoci scriptu:

```
"build": "yarn lerna run build"
```

Následně soubory vygenerované build scriptem budeme používat k publikaci.

10.2.1 **Package.json**

Hlavní package.json:

```
{
  "name": "atomic-design-system",
  "version": "1.0.0",
  "main": "index.js",
  "repository": "https://github.com/altynali/atomic-design-system.git",
  "author": "Alina <alina.altynbaeva.00@mail.ru>",
  "license": "MIT",
  "devDependencies": {
    "lerna": "^7.1.4"
  },
  "private": true,
  "workspaces": {
    "packages": [
```

```
      "packages/*",
      "playgrounds/*"
    ],
    "nohoist": [
      "**/normalize-scss"
    ]
  },
  "scripts": {
    "build": "yarn lerna run build",
    "dev": "yarn lerna run dev",
    "test": "yarn lerna run test"
  }
}
```

10.3 Implementace Reactu

V dalším kroku zavedeme soubor react do package souboru a přidáme do package.json react, react-dom a typescript.

```
 "devDependencies": {
    "@types/react": "^18.2.23",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "typescript": "^5.2.2"
  },
```

tsconfig.json pro react package:

```
{
  "compilerOptions": {
    "outDir": "lib",
    "module": "esnext",
    "lib": ["DOM", "ESNext"],
    "jsx": "react",
    "allowSyntheticDefaultImports": true,
    "target": "esnext",
    "noImplicitAny": true,
    "strictNullChecks": true,
```

```
    "noImplicitReturns": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "declaration": true,
    "moduleResolution": "node"
  },
  "include": ["src"],
  "exclude": ["node_modules", "lib"]
}
```

Vytvoříme soubory atoms, molecules, organisms a templates. Začínáme vytvářet komponenty a nastavovat vývojový skript pro každý z našich balíčků.

V rollup.config.js musíme naimportovat všechny již hotové komponenty, které chceme přidat do naší knihovny.

rollup.config.js:

```
import Ts from "rollup-plugin-typescript2"

const config = {
  input: [
    "src/index.ts",
    "src/atoms/Color/index.ts",
    "src/atoms/Text/index.ts",
    "src/atoms/Margin/index.ts",
    "src/atoms/Button/index.ts",
    "src/atoms/Padding/index.ts",
    "src/atoms/Grid/index.ts",
    "src/atoms/Checkbox/index.ts",
    "src/molecules/Card/index.ts",
    "src/molecules/Select/index.ts",
    "src/molecules/Input/index.ts",
    "src/molecules/CardList/index.ts",
    "src/organisms/Form/index.ts",
    "src/templates/Layout/index.ts",
  ],
  output: [
    {
      dir: "lib",
```

```
      format: "esm",
      sourcemap: true,
      preserveModules: true,
    },
  ],
  plugins: [Ts()],
  external: ["react", "atomic-design-system-foundation"],
}

export default config
```

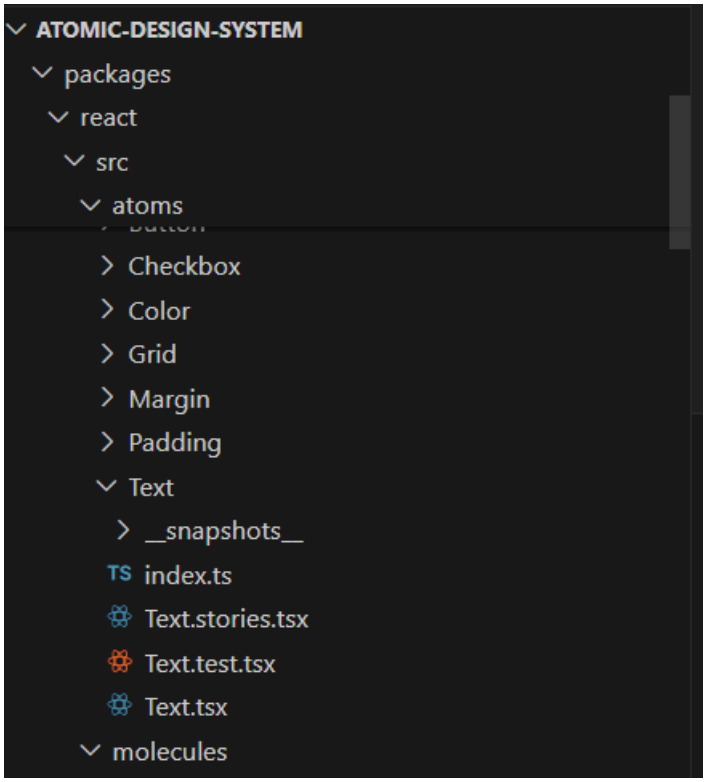
10.4 Implementace Komponenty

V této kapitole si ukážeme, jakým způsobem můžeme přidávat nové komponenty do našeho design systému.

10.4.1 Příklad vytváření komponenty

Packages/react:

Zdroj: vlastní



Obrázek č. 14 Detailní záběr na komponenty, 2024

Rozmístíme react komponentu do příslušného souboru s příslušným index.ts.

Index.ts je určen jako výchozí exportní mechanismus, aby se odstranily importy úplnou cestou.

Samotná komponenta Text.tsx:

```
import React, { FC, PropsWithChildren } from "react"
import { Sizes } from "atomic-design-system-foundation/lib/src"

export type TextProps = {
  className?: string
  size?: string
} & PropsWithChildren

const Text: FC<TextProps> = (props) => {
  const { size = Sizes.base, children, className } = props

  const cls = className + ` atds-text atds-text-${size}`

  return <p className={cls}>{children}</p>
}
```

```
export default Text
```

index.ts:

```
import Text from "./Text"

export { Text }
```

V index.ts v packages/react/src chceme všechny komponenty vyjmenovat.

```
…
import { Text } from "./atoms/Text"

export {
  Text,
  …
}
```

Následně v index.ts v packages/react modulu chceme všechny komponenty, co jsme napsali exportovat za účelem dalšího použití:

```
import {
  Color,
  Margin,
  Select,
  Text,
  Padding,
  Button,
  Checkbox,
  Card,
  Input,
  Form,
  Layout,
  Grid,
} from "./src/index"

export {
  Color,
  Text,
```

```
  Margin,
  Padding,
  Button,
  Checkbox,
  Card,
  Select,
  Input,
  Form,
  Layout,
  Grid,
}
```

10.4.2 **Příklad stylování Komponenty**

Text.scss:

```
@import "foundation/all";

.atds-text {
  margin: 0;
}

@each $size, $value in $fontSizes {
  .atds-text-#{$size} {
    font-size: $value;
  }
}
```

10.4.3 **Příklad testování Komponenty**

Všechny testy pro naši komponentu obalíme do funkce describe, abychom věděli, že se to týká jedné komponenty.

Příklad napsaného testu:

```
describe("Text component", () => {
  it("renders with default size", () => {
    const { getByText } = render(<Text>Hello World</Text>)
```

```
    const textElement = getByText("Hello World")
    expect(textElement).toBeInTheDocument()
    expect(textElement).toHaveClass("atds-text")
    expect(textElement).toHaveClass("atds-text-base")
  })
})
```

Výsledky testů pro tuto komponentu:

Zdroj: vlastní

```
yarn run v1.22.19
$ jest --verbose -u
 PASS  src/atoms/Text/Text.test.tsx
  Text component
    √ renders with default size (31 ms)
    √ renders with custom size (3 ms)
    √ snapshot of component (10 ms)

 › 1 snapshot written.
Snapshot Summary
 › 1 snapshot written from 1 test suite.

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   1 written, 1 total
Time:        2.819 s
Ran all test suites.
Done in 4.06s.
```

Obrázek č. 15 Unit Test komponenty, 2024

10.4.4 **Přístupnost**

Přístupnost webové stránky a komponentů zlepšíme tím, že budeme používat správné sémantické tagy, aria atributy, kontrast barev, přizpůsobené ovládání komponenty pomoci klávesnice a další.

V souboru molecules/Select můžeme najít utility funkci onButtonKeyDown:

```
export const KEYS = {
  ENTER: "Enter",
  SPACE: " ",
  DOWN_ARROW: "ArrowDown",
  ESC: "Escape",
  UP_ARROW: "ArrowUp",
}

export const onButtonKeyDown = (
  event: KeyboardEvent<HTMLButtonElement>,
```

```
  setIsOpen: (isOpen: boolean) => void,
  highlightOption: (highlightedOption: number) => void
) => {
  event.preventDefault()

  if ([KEYS.ENTER, KEYS.SPACE, KEYS.DOWN_ARROW].includes(event.key)) {
    setIsOpen(true)
    highlightOption(0)
  }
}
```

Následně využijeme v Select komponente ve funckci onKeyDown:

```
<button
        data-testid="AtdsSelectButton"
        aria-controls="atds-select-list"
        aria-haspopup={true}
        aria-expanded={isOpen ? true : undefined}
        className="atds-select__label"
        onClick={(event: MouseEvent) => onLabelClicked(event)}
        ref={labelRef}
        onKeyDown={(event: KeyboardEvent<HTMLButtonElement>) =>
          onButtonKeyDown(event, setIsOpen, highlightOption)
        }
>
```

Také se používají aria-controls, aria-haspopup a aria-expanded atributy. Ukazují, že mají vyskakovací okno, a pomocí tlačítka se ovládají, zda jsou možnosti v seznamu Selectu ukázané nebo ne.

A kvůli tomu musíme ukázat explicitně id pro komponentu kterou ovládáme:

```
      <ul
        role="menu"
        style={{ top: overlayTop }}
        className={`atds-select__overlay ${
          isOpen ? "atds-select__overlay--open" : ""
        }`}
        id="atds-select-list"
      >
```

10.4.5 **Storybook**

Chceme přidat do našeho projektu Storybook, abychom zdokumentovali a ukázali naše komponenty. Prezentujeme to, jak vypadají a jak fungují.

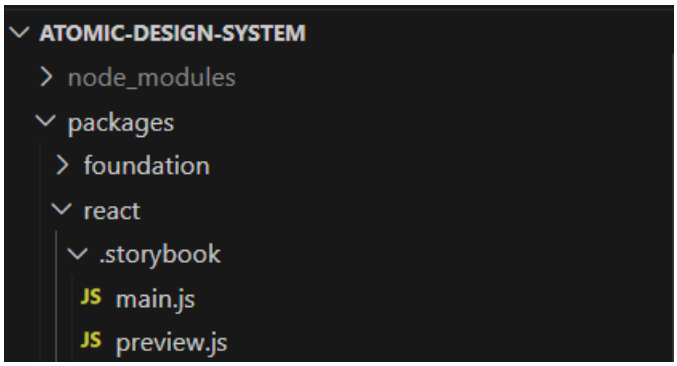
Storybook chceme přidat do packages/react.

Příkaz pro přidání storybooku:

```
npx storybook@latest init
```

Přidá se storybook konfigurace:

Zdroj: vlastní



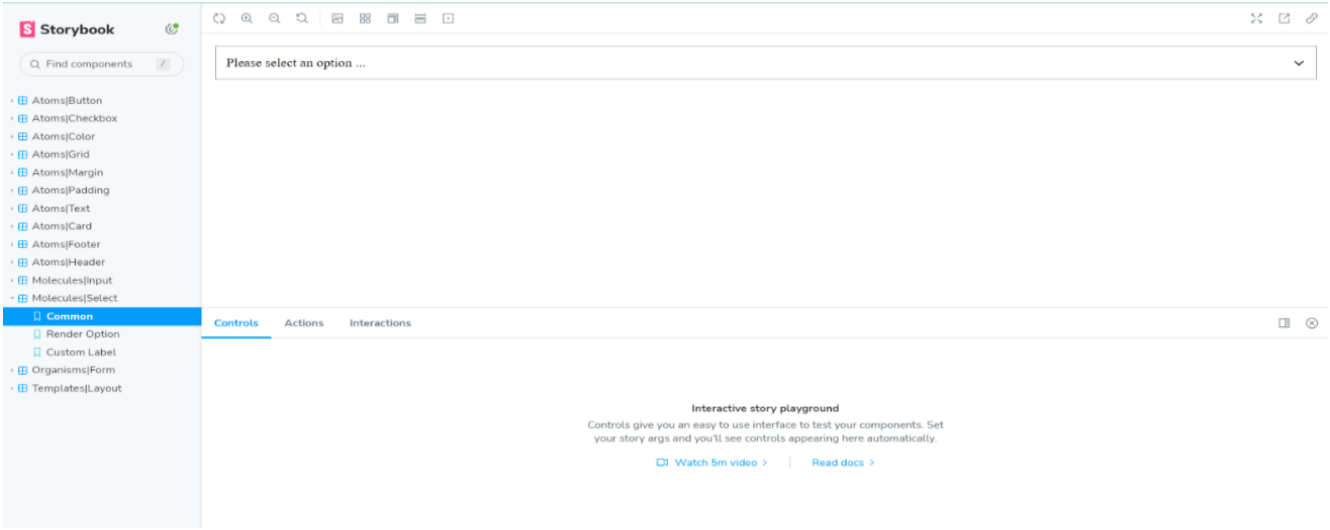
Obrázek č. 16 Storybook konfigurace, 2024

Text.stories.tsx:

```tsx
import React from "react"
import Text from "./Text"
import "atomic-design-system-scss/lib/Text.css"

export default {
  title: "Atoms|Text",
}

export const Common = () => <Text>text</Text>
```

A uvidíme naši knihovnu, když se spustí příkaz yarn storybook:

Zdroj: vlastní



Obrázek č. 17 Storybook playground, 2024

10.5 Výsledky Unit Testingu

Je nutné vyhodnotit výsledky testů. V případě nečekaných výsledků to bude znamenat, že naše komponenty se nechovají tak, jak jsme anticipovali. A to znamená, že chování naprogramovaných komponent musíme opravit.

Zdroj: vlastní

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  COMMENTS

    √ snapshot of component (25 ms)

 PASS  src/organisms/Form/Form.test.tsx (5.02 s)
  Form component
    √ renders form with label and children (82 ms)
    √ applies custom className (14 ms)
    √ calls onSubmit event handler when submitted (33 ms)
    √ snapshot of the base state (39 ms)

 PASS  src/molecules/Select/Select.test.tsx (5.174 s)
  Select
    √ renders all options passed to it (238 ms)
    √ renders options using custom renderOption method if passed as prop (21 ms)
    √ calls the onOptionSelected prop with the selected option and its index if passed (64 ms)
    √ the button label changes to the selected option label (38 ms)
    √ snapshot of the selected option state (68 ms)
    √ snapshot of the base state (13 ms)
    √ snapshot of the options menu open state (12 ms)
    √ can customize select label (7 ms)

 PASS  src/templates/Layout/Layout.test.tsx (5.208 s)
  Layout component
    √ renders layout with default label and children (57 ms)
    √ renders layout without header if noHeader prop is true (5 ms)
    √ renders layout without footer if noFooter prop is true (4 ms)
    √ applies custom className (8 ms)
    √ renders layout with custom label (6 ms)
    √ snapshot of the base state (22 ms)

 › 1 snapshot written.
Snapshot Summary
 › 1 snapshot written from 1 test suite.

Test Suites: 6 passed, 6 total
Tests:       29 passed, 29 total
Snapshots:   1 written, 6 passed, 7 total
Time:        7.11 s
Ran all test suites.
Done in 8.35s.
```

Obrázek č. 18 Výsledky Unit, 2024

10.6 **Publikace**

Musíme se přihlásit na webu https://www.npmjs.com/signup. Chceme přidat do package.json projektu, jehož chceme publikovat, několik změn. Specifikovat které chceme soubory publikovat a přístup k těmto souborům pomoci řádku `files` a `publishConfig`.

```
{
  "name": "atomic-design-system-react",
  "version": "1.0.0",
  "main": "lib/index.js",
  "license": "MIT",
  "files": [
    "lib/src"
  ],
  …,
  "publishConfig": {
    "access": "public"
  }
}
```

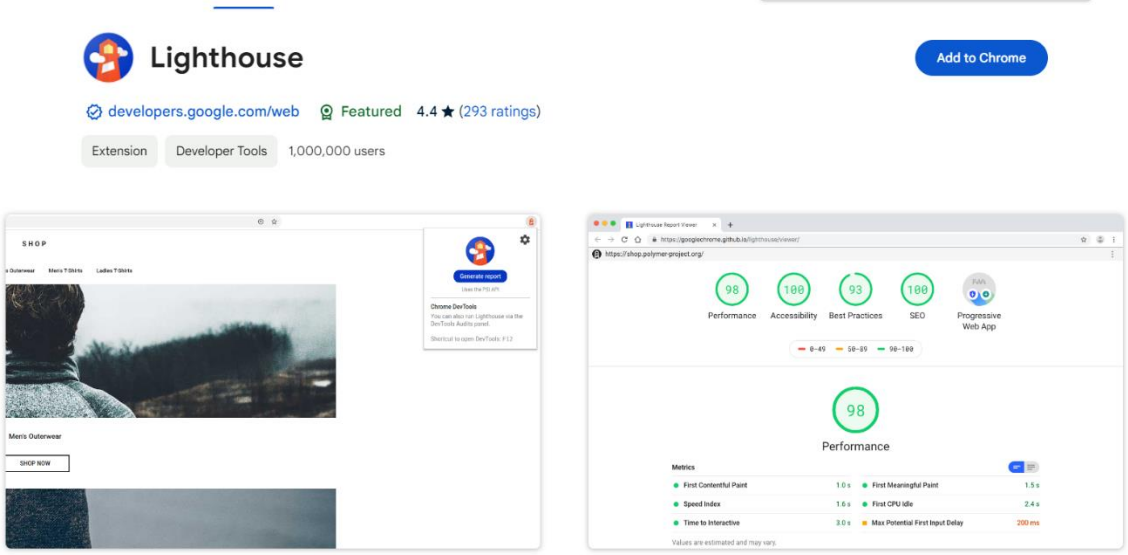
Pomocí skriptů se přihlásíme a vydáme naší práci. To s pomocí CLI, do kterého napíšeme tyto příkazy:

```
npm login
npm publish
```

11 Zhodnocení výsledků

Zvolila jsem napsat Todo aplikaci, abychom si komponenty otestovali a zanalyzovali. Pro účely analýzy webu byl zvolen nástroj LightHouse.

Obrázek č. 19 Lighthouse (Lighthouse, 2024)



Způsoby analýzy:

1. **Metoda Optimalizace:**

   Vyhodnocení účinnosti design systému pomocí LightHouse. Očekáváme, že budeme mít vysoký výkon. V případě, že výsledky nebudou takové, jaké očekáváme, tak provedeme optimalizaci výkonu aplikace. Cílem optimalizaci je zlepšení výkonu webové stránky, kde bude atomický design systém používán.

2. **Metoda způsob analýzy:**

   Provedeme srovnání dvou stejných aplikací napsané různými metodami. Jedna bude napsána s použitím atomického designu a druhá jej nebude využívat, v obou případech napsána v React.js. Cílem je ukázat klady a zápory využití atomického design systému.

Výzkumné otázky:

- Jaký bude výkon aplikace, na kterém je používán náš design systém?
- Jakým způsobem výkon můžeme zlepšit?
- Kolik na optimalizaci můžeme ušetřit?
- Jaký je rozdíl ve výkonu mezi aplikací, která využívá atomický designový systém a aplikací, která nevyužívá?

Cíle výzkumu:

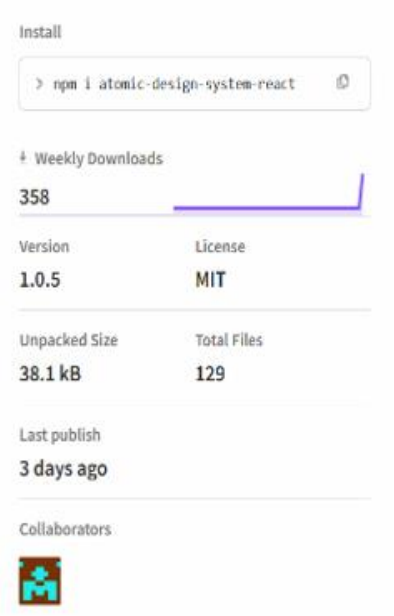
- Vyhodnotíme účinnost designového systému pomocí aplikace LightHouse.
- Budeme se snažit identifikovat určité nedostatky v designovém systému, abychom předešli snížení výkonu.
- Optimalizujeme výkon aplikace na základě zjištěných nedostatků.
- Budeme srovnávat výkon dvou identických aplikací, první bude vytvořena s pomocí atomického designu a druhou vytvoříme bez něj.
- Analyzujeme výsledky srovnání a vyvodíme závěry o vlivu atomického designu na chod aplikace.

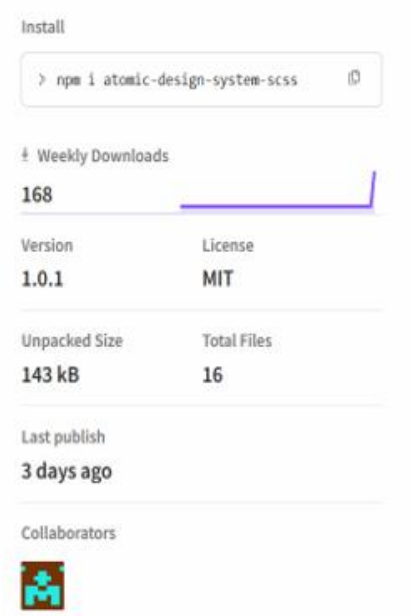
11.1 Optimalizace

Publikovaný balíček zkontrolujeme na npm registry v našem účtu, zde zjistíme verzi, velikost a datum zveřejnění, odtud postoupíme zpátky do našeho webu. Otevřeme doplňky a spustíme test výkonu.

Původní velikost našich balíčků, před optimalizací:

Obrázek č. 20 Publikovaný React – Atomic. *(npmjs.com, 2024)*



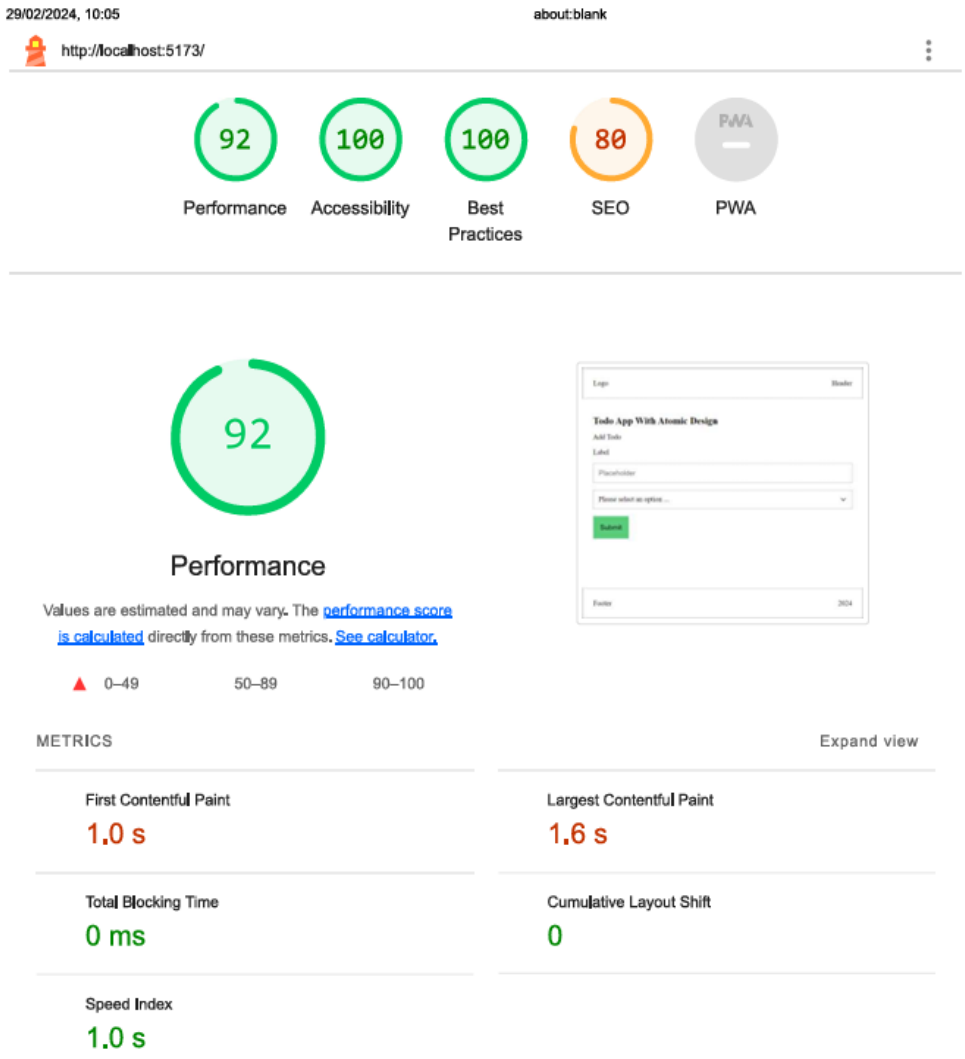


Obrázek č. 21 Publikovaný Sass – Atomic Design. *(npmjs.com, 2024)*

Spustíme analýzu v prohlížeči a počkáme na výsledky testu.

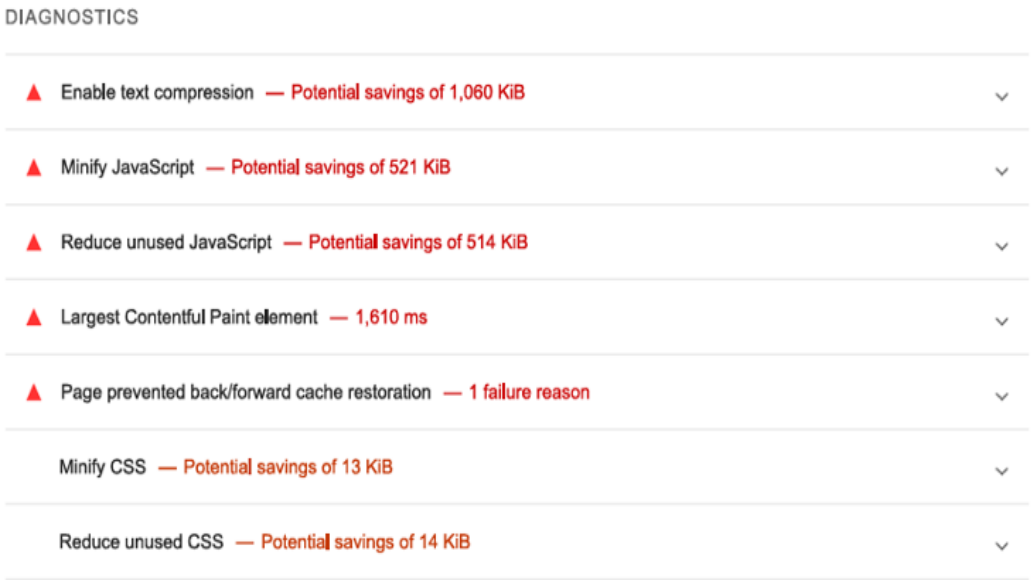
Výsledky analýzy naší stránky:

Zdroj: vlastní



Obrázek č. 22 Výsledky Performance Testu v.1, 29.2.2024

Zdroj: vlastní



Obrázek č. 23 Výsledek Performance Testu, 29.2.2024

Na základě výsledku si vyhodnotíme vhodnost optimalizace naší aplikace, a to za použití vhodných nástrojů, a to s pomocí komprese.

Změníme build.js script v packages/scss souboru, pro kompresi css souborů:

```
const compile = (inputPath, outputPath) => {
  const result = Sass.renderSync({
    data: Fs.readFileSync(Path.resolve(inputPath)).toString(),
    outputStyle: "compressed",
    includePaths: [Path.resolve("src")],
  })

  Fs.writeFileSync(Path.resolve(outputPath), result.css.toString())
}
```

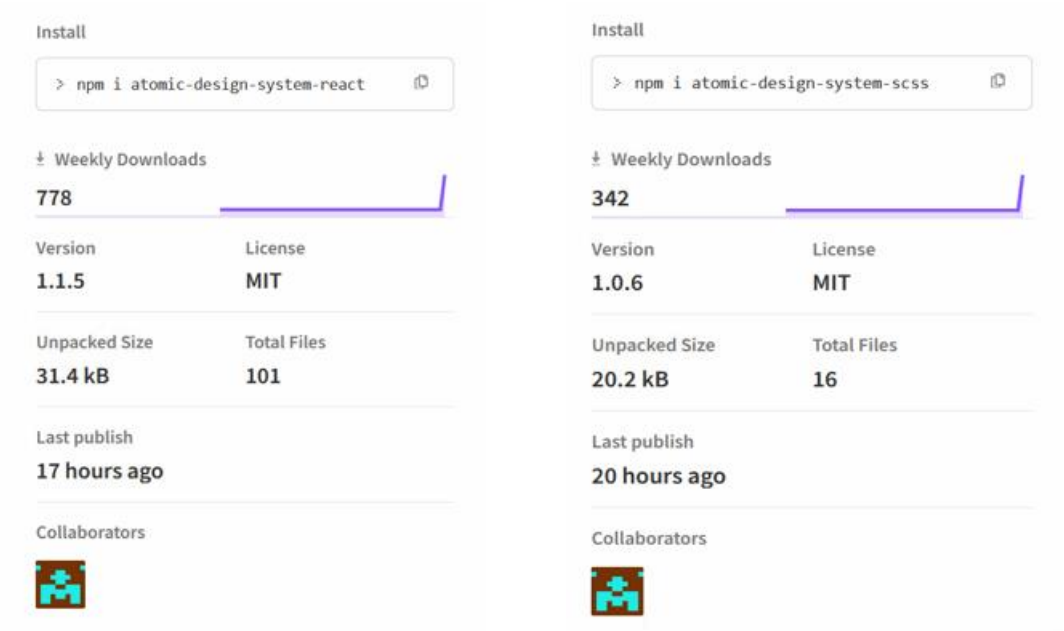
Přidáme taky do rollup.config.js v packages/react balíčky, které nám provedou kompresi existujícího kódu:

```
import terser from "@rollup/plugin-terser"
import { uglify } from "rollup-plugin-uglify"
const gzipPlugin = require("rollup-plugin-gzip").default

const config = {
  …,
  plugins: [
      …,
    terser(),
    uglify(),
    gzipPlugin(),

  ],
  …
}

export default config
```

Uděláme další verzi naše knihovny a provedeme její publikaci, abychom se podívali, jak se změnil výkon naší aplikace a velikost balíčků.

Velikost našich balíčků, po optimalizací:

Obrázek č. 24 Optimalizace React. (npmjs.com, 2024)

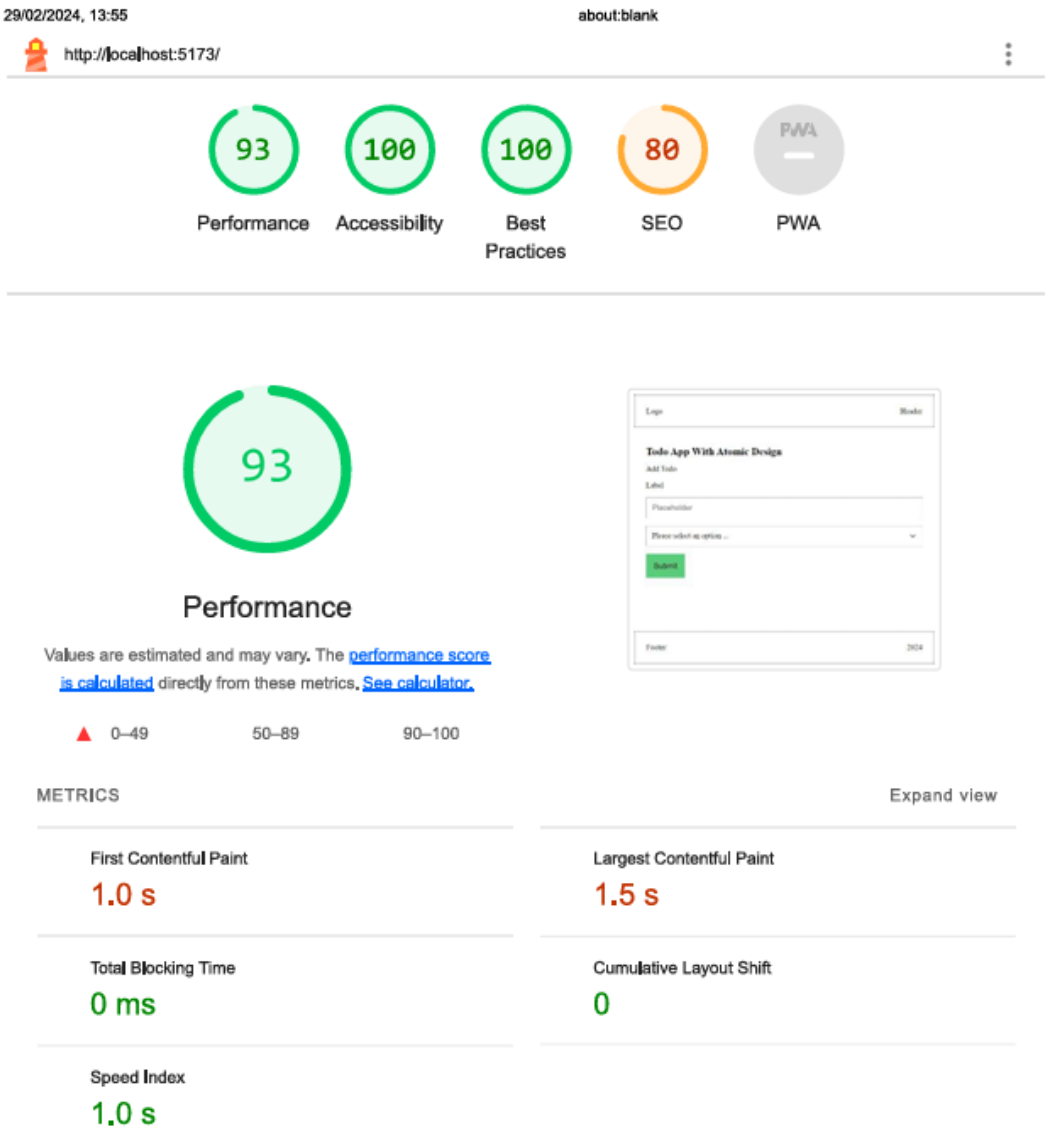


Obrázek č. 25 Optimalizace Scss. (npmjs.com, 2024)

Můžeme vidět, že velikost balíčku i složek se snížila. Z toho můžeme usuzovat, že se zlepší výkon aplikace. Tento předpoklad se pokusíme ověřit dalším testem výkonu.

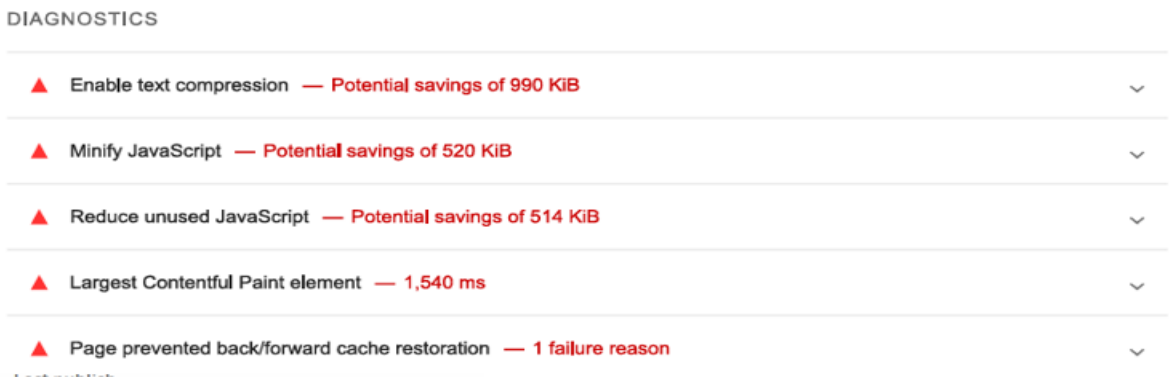
Výsledky analýzy naší stránky po optimalizací:

Zdroj: vlastní



Obrázek č. 26 Výsledky Performance Testů v.2, 29.2.2024

Zdroj: vlastní



Obrázek č. 27 Výsledek Performance Testu v.2, 29.2.2024

Pro přehlednost sestavíme tabulku z výsledných hodnot a porovnáme je.

Zdroj: vlastní

	Atomický design v. 1	Atomický design v. 2
Text komprese	1060 KiB	990 KiB
Minifikace Javascript	521 KiB	520 KiB
Redukce neužitého Javascriptu	514 KiB	514 KiB
Největší obsahový prvek vykreslení	1,610 ms	1,540 ms
Minifikace CSS	13 KiB	0
Redukce neužitého CSS	14 KiB	0

Tabulka č. 1 Porovnaní verzí atomického designu v.1 a atomického designu v. 2, 2024

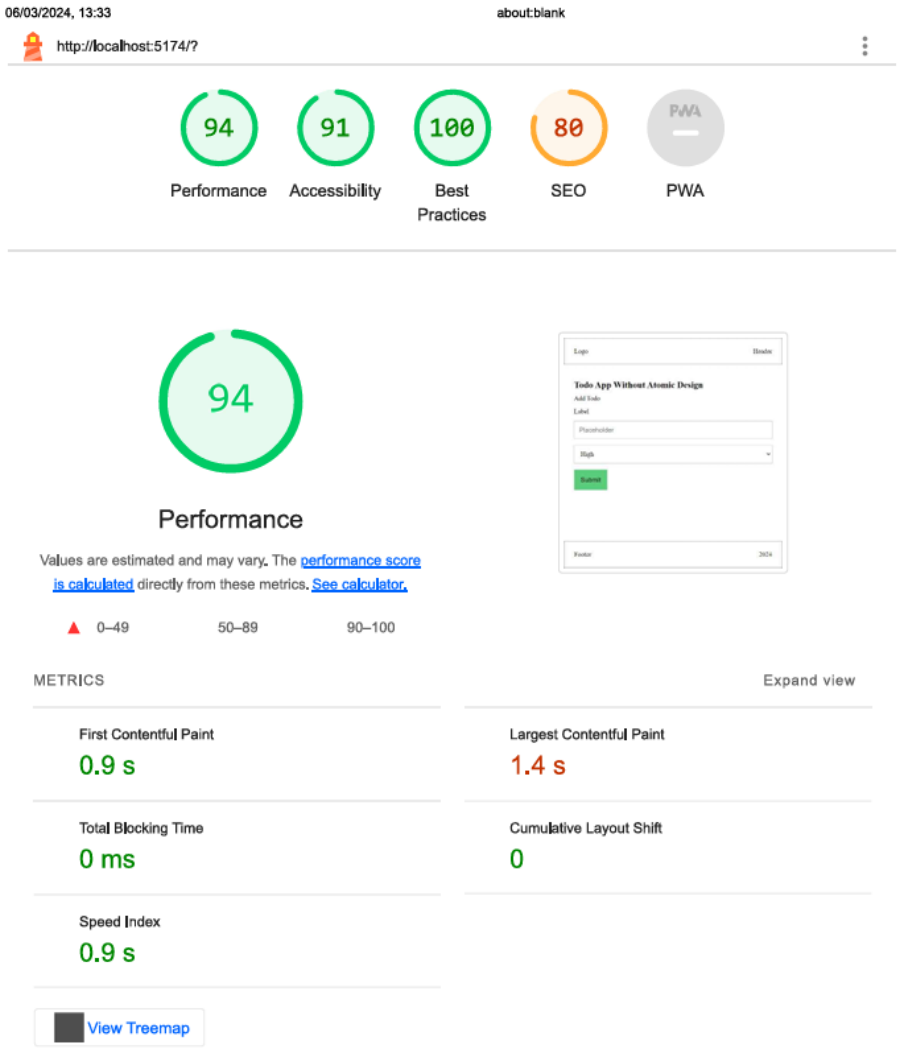
Z tabulky č.1 jsme vyvodili, že po provedení optimalizace aplikace s atomickým designem, zlepšili jsme text komprese o 70 KiB, což je zlepšení o 9.5 % v tomto parametru, minifikací JS ušetřil 1 KiB. Největší obsahový prvek vykreslení je rychlejší o 70 ms, cožzvýšilo rychlost vykreslení tohoto obsahu o 4.3 %. Na minifikaci CSS a neužitého kódu se podařilo tento problém zcela odstranit a ušetřit tím dalších 27 KiB.

11.2 Porovnání výkonu

Naším cílem je nyní vytvořit aplikaci React.js která bude mít stejnou funkcionalitu. Budeme postupovat stejným způsobem a po jejím dokončení proběhne performance test z výsledných hodnot vytvoříme tabulku a do ní vložíme hodnoty optimalizovaného modelu s atomickým designem. Hodnoty mezi sebou porovnáme a vyhodnotíme.

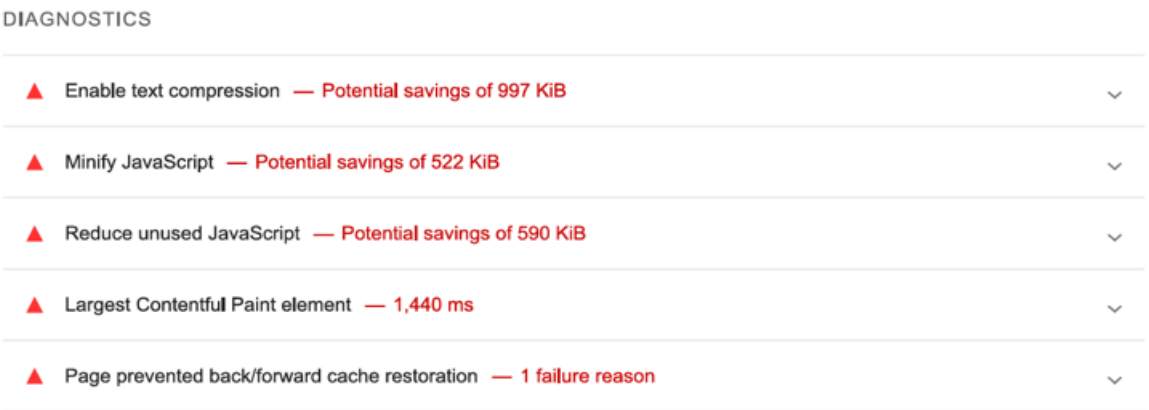
Výsledky analýzy:

Zdroj: vlastní



Obrázek č. 28 Výsledek bez atom. designu, 6.3.2024

Zdroj: vlastní



Obrázek č. 29 Výsledek bez atom. designu, 6.3.2024

Z výsledných hodnot optimalizace jsme si pro přehlednost vytvořili tabulku, z které vyhodnotíme závěry optimalizace.

Zdroj: vlastní

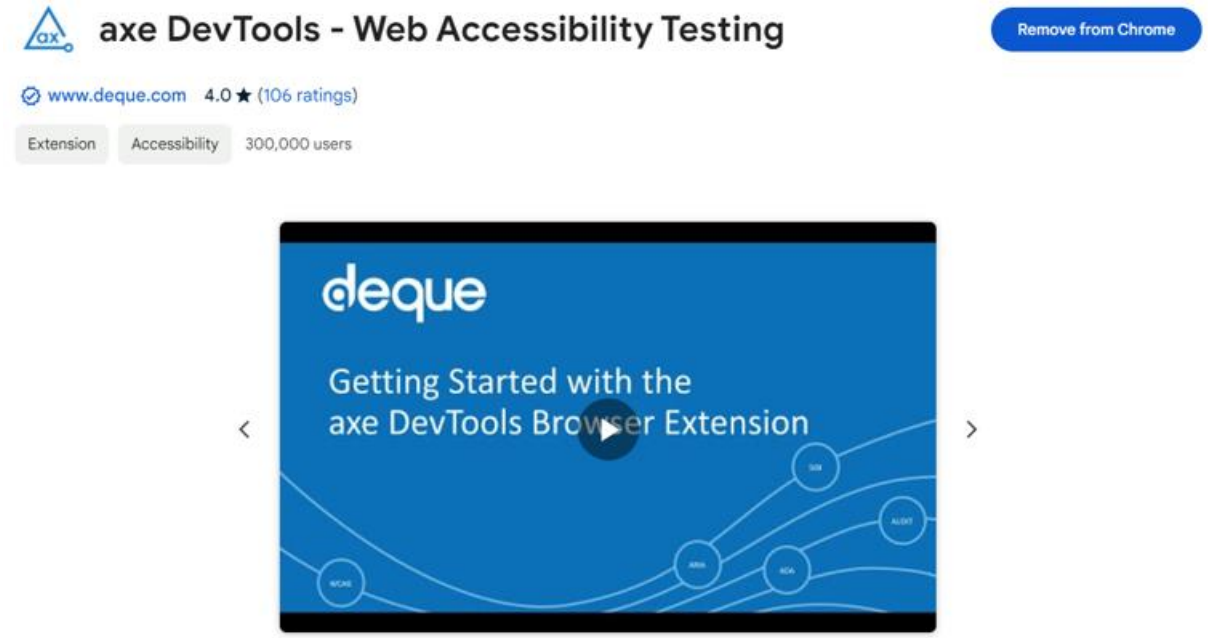
	Atomický design	Klasický způsob
Text komprese	990 KiB	997 KiB
Minifikace Javascript	520 KiB	522 KiB
Redukce neužitého Javascriptu	514 KiB	590 KiB
Největší obsahový prvek vykreslení	1,540 ms	1,440 ms

Tabulka č. 2 Porovnání aplikace s atomickým designem a.klasickým vývojem ., 2024

Na základě zkoumání hodnot v tabulce č. 2 jsme došli k těmto závěrům. U atomického designu je možné k optimálnímu stavu odstranit o 7 KiB méně, což znamená, že je postup s atomickým designem v tomto parametru o 0.7% lepší. Potenciální JavaScript kód, který jde minifikovat je o 2 KiB nižší, a ten který se dá redukovat je o 76 KiB nižší. Největší obsahový prvek vykreslení má delší čas vykreslování, a to o 100 ms.

11.3 Testování Přístupnosti

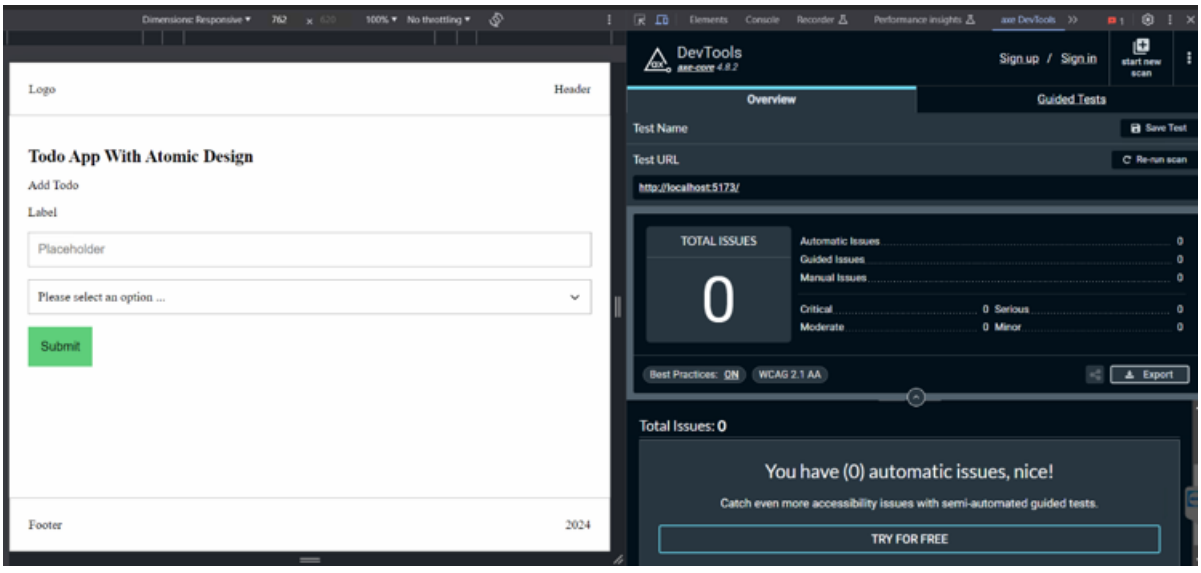
Testování přístupnosti aplikace bylo provedeno pomocí axe DevTools, rozšíření Chrome prohlížeči.



Obrázek č. 30 Axe DevTools (Axe DevTools, 2024)

Výsledky testování webu, který byl napsaný komponenty z atomického designu:

Zdroj: vlastní



Obrázek č. 31 Test přístupnosti s axeDevTools, 2024

Díky tomu, že při vývoji našeho design systému jsme se zaměřili na přístupnost, test prošel bez nedostatků.

12 Závěr

Vývoj vlastního design systému je náročný proces, při kterém je vhodné si rozvrhnout své úsilí do dílčích etap, jež si vytyčíme a pomalu se s nimi budeme měřit a zdolávat je. Atomický design přinesl nový vhled a představu o vývoji webu, jako takového, dokázal to jednoduchým přirovnáním, které zná každý, kdo seděl ve školních lavicích.

Mnozí se mohou ptát, v čem spočívá přelomová myšlenka tohoto konceptu, když je již delší dobu používán. Nicméně, jeho síla spočívá v tom, že nabízí efektivnější a zjednodušenější řešení pro vývoj webových projektů. Atomický design je zejména vhodný pro velké společnosti, které mají více projektů a potřebují jednotný vzhled a rychlý vývoj komponent.

Výhody vlastního atomického designového systému jsou patrné i u menších aplikací. V našem případě jsme dělali porovnání dvou jednoduchých aplikací na React.js, každou vytvořenou jiným způsobem. Výsledky tohoto porovnání je třeba brát v úvahu s ohledem na omezenou vzorkovou velikost. Výsledky mého snažení tvrdí, že se při porovnání mezi aplikací s atomickým designem a aplikací vyvíjenou klasickým způsobem tyto dvě aplikace o mnoho neliší, avšak to nemůžeme tvrdit pro všechny případy.

Domnívám se, že při obsáhlejší analýze by došlo k výraznějším rozdílům v testech výkonnosti. Má tvrzení jsou postavena nad ideou atomického designu, který je koncipován pro bohaté aplikace, kde se dobře uplatní jeho silné vlastnosti. Toto své tvrzení, ale nemám v současné chvíli podloženo daty a bylo nutné jej dále studovat.

Ráda bych nakonec odpověděla na výzkumné otázky.

- Jaký bude výkon aplikace, na kterém je používán náš design systém?

Souhrn výsledků naznačuje, že aplikace s atomickým designovým systémem dosahuje výkonu 93 ze 100, což je velmi solidní výsledek.

- Jakým způsobem výkon můžeme zlepšit?

Pro další zlepšení výkonu je navrženo využití balíčků pro kompresi a využití kompresního výstupu CSS.

- Kolik na optimalizaci můžeme ušetřit?

Celkově pomoci optimalizace jsme ušetřili 93 KiB a urychlili jsme Největší obsahový prvek vykreslení o 70 ms.

- Jaký je rozdíl ve výkonu mezi aplikací, která využívá atomický designový systém a aplikací, která nevyužívá?

Můžeme vidět, že atomicky design měl lepší výsledky ve všech parametrech kromě Největší ho obsahového prvku vykreslení. Můžeme předpokládat, že se to stalo kvůli tomu, že náš design systém má složitější strukturu HTML a víc komplexní CSS.

Celkově lze konstatovat, že aplikace s atomickým designem vykazuje nepatrně lepších výsledků oproti aplikaci bez tohoto designového systému.

13 Seznam použitých zdrojů

Avnishsharamav. (30. 2023). *Code Editor | VWO*. Získáno 12. Únor 2024, z VWO Glossary: https://vwo.com/glossary/code-editor/

Ayebola, J. (23. Listopad 2023). *DOM manipulation in JavaScript – a comprehensive guide for beginners*. Získáno 28. Únor 2024, z freeCodeCamp.org: https://www.freecodecamp.org/news/dom-manipulation-in-javascript/

Bertoli, M. (2017). *React Design Patterns and Best Practices.* Birmingham: Packt Publishing.

Bos Bert. (17. Prosinec 2016). *A brief history of CSS until 2016*. Získáno Únor 2024, z w3.org: https://www.w3.org/Style/CSS20/history.html

CLI Introduction | Yarn. (2022). Získáno Únor 2024, z yarnpkg.com: https://classic.yarnpkg.com/lang/en/docs/cli/

CSS: Cascading Style Sheets. (30. Leden 2024). Získáno Únor 2024, z MDN: https://developer.mozilla.org/en-US/docs/Web/CSS

Documentation - TypeScript for JavaScript Programmers. (1. Březen 2024). Získáno 28. Únor 2024, z Typescript: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Frantz, K. (červen 2021). *CSS architecture checklist*. Získáno Listopad 2023, z udemy: https://www.udemy.com/course/react-for-senior-engineers/

Frost, B. (© 2016). *Atomic Design Methodology*. Získáno 14. Leden 2024, z Atomic Design by Brad Frost: https://atomicdesign.bradfrost.com/chapter-2/

Gatwiri , V. (25. Květen 2023). *React Component Lifecycle Methods – Explained with Examples*. Získáno 23. Únor 2024, z freeCodeCamp.org: https://www.freecodecamp.org/news/react-component-lifecycle-methods/

GfG. (17. Leden 2024). *Types of software testing*. Získáno 29. Únor 2024, z GeeksforGeeks: https://www.geeksforgeeks.org/types-software-testing/

Godbolt, M. (2016). *Frontend architecture for design systems: a modern blueprint for scalable and sustainable websites.* Sebastopol, California: Oreilly Media: O'Reilly Media, Inc.

Hél, S. (2022). *Lekce 1 - Základní struktura HTML*. Získáno Únor 2024, z itnetwork.cz: https://www.itnetwork.cz/html-css/html5/zakladni-struktura-html

Herbert, D. (13. Listopad 2023). *What is React.js? Uses, Examples, & More*. Získáno 29. Únor 2024, z HubSpot: https://blog.hubspot.com/website/react-js

HTML body tag. (2023). Získáno 7. Březen 2024, z w3schools.com: https://www.w3schools.com/tags/tag_body.as

HTML doctype declaration. (2023). Získáno Únor 2024, z W3Schools: https://www.w3schools.com/tags/tag_doctype.ASP

HTML html tag. (2023). Získáno Únor 2024, z w3schools: https://www.w3schools.com/tags/tag_html.asp

HTML: HyperText Markup Language | MDN. (17. Červenec 2023b). Získáno Únor 2024, z MDN: https://developer.mozilla.org/en-US/docs/Web/HTML

Initiative, W. W. A. (1. Srpen 2023). *Evaluating web accessibility Overview*. Získáno Únor 2024, z Web Accessibility Initiative (WAI): c

Initiative, W. W. A. (20. Listopad 2023). *Introduction to web accessibility*. Získáno 25. Únor 2024, z Web Accessibility Initiative (WAI): https://www.w3.org/WAI/fundamentals/accessibility-intro/

Introduction to Node.js. (2023). Získáno 14. Leden 2024, z Node.js: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Jacksi, K., & Abass, S. M. (Září 2019). Development history of the world wide web. *International Journal of Scientific & Technology Research*, stránky 75–79. Získáno 14. Leden 2024,z

https://www.researchgate.net/publication/336073851_Development_History_Of_The_World_Wide_Web.

JavaScript | MDN. (25. Září 2023). Získáno 27. Únor 2024, z MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript

legacy.reactjs.org. (2022). *Components and Props - React*. Získáno 22. Únor 2024, z legacy.reactjs.org: https://legacy.reactjs.org/docs/components-and-props.html

Luke, K., Loginov, N., & Thompson, E. (23. Říjen 2023). *Creating and publishing unscoped public packages | npm*. Získáno 2. Březen 2024, z npm Docs: https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages

Mandal, B. (8. Prosinec 2023). *Javascript module bundler: rollup.js (Essential to learn)*. Získáno 1. Března 2024, z Medium: https://medium.com/@artbindu/javascript-module-bundler-rollup-js-essential-to-learn-615495cd2ead

Nakazawa, C., McKenzie, S., & Kyle, J. (26. Červen 2018). *Yarn: A new package manager for JavaScript*. Získáno 7. Únor 2024, z Engineering at Meta: https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/

Node.js — An introduction to the npm package manager. (2023). Získáno 6. Únor 2024, z Node.js: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

Node.js — An introduction to the npm package manager. (2023ne). Získáno 6. Únor 2024, z Node.js: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

Passing props to a component – react. (2023). Získáno 22. Únor 2024, z react.dev: https://react.dev/learn/passing-props-to-a-component

Pedamkar, P. (3. Červenec 2023). *Versions of Html*. Získáno Únor 2024, z EDUCBA: https://www.educba.com/versions-of-html/

Pittet, Sten. (2023). *The different types of testing in software*.). Získáno 27. Únor 2024, z Atlassian: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

SASS: Basics. (2023). Získáno Únor 2024, z Sass: https://sass-lang.com/guide/

SASS: Documentation. (2023). Získáno 22. Únor 2024, z Sass: https://sass-lang.com/documentation/

SASS: variables. (2023). Získáno 22. Únor 2024, z Sass: https://sass-lang.com/documentation/variables/

Stefanov, S. (2016). *React : Up & running: Building Web Applications.* Sebastopol: O'Reilly Media.

Storybook: Frontend workshop for UI development. (2023). Získáno 1. Březen 2024, z https://storybook.js.org/

Thompson, E. (23. Září 2020). *About the public npm registry*. Získáno 6. Únor 2024, z npm Docs: https://docs.npmjs.com/about-the-public-npm-registry

What is a Framework? - GeeksforGeeks. (2023). Získáno 26. Únor 2024, z GeeksforGeeks: https://www.geeksforgeeks.org/what-is-a-framework/amp/

Why Storybook? • Storybook docs. (2023). Získáno 1. Březen 2024, z https://storybook.js.org/docs/get-started/why-storybook

World Wide Web Foundation. (8. Říjen 2020b). *History of the Web - World Wide Web Foundation*. Získáno Únor 2024, z World Wide Web Foundation - Founded by Tim Berners-Lee, Inventor of the Web, the World Wide Web Foundation Empowers People to Bring About Positive Change: https://webfoundation.org/about/vision/history-of-the-web/

Writing Markup with JSX – React. (2023). Získáno 24. Únor 2024, z React.dev: https://react.dev/learn/writing-markup-with-jsx

Yang, X. (21. Červen 2018). *JavaScript is a loosely typed language, meaning you don't have to specify what type of information.* Získáno Únor 2024, z Medium: https://xiaoyunyang.medium.com/javascript-is-a-loosely-typed-language-meaning-you-dont-have-to-specify-what-type-of-information-137408d54fc7

Yarn. (2022). Získáno 7. Únor 2024 z Yarn: https://classic.yarnpkg.com/en/docs/cli/add

Yarn. (2022). Získáno 7. Únor 2024, z Yarn: https://classic.yarnpkg.com/en/docs/cli/init

Yarn. (2022). Získáno 7. Únor 2024, z Yarn: https://classic.yarnpkg.com/lang/en/docs/install/#windows-stable

Yasar, K. (12. Srpen 2022). *software testing*. Získáno Únor 2024, z WhatIs: https://www.techtarget.com/whatis/definition/software-testing

Lighthouse. (2024). chromewebstore.google.com. Získáno [online] Únor 2024 z https://chromewebstore.google.com/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl

AxeDevtools. (2024). chromewebstore.google.com. Získáno [online] Únor 2024 z https://chromewebstore.google.com/detail/axe-devtools-web-accessib/lhdoppojpmngadmnindnejefpokejbdd

npmjs.com. (2024). npmjs.com. Získáno [online] Únor 2024 z npmjs.com. https://www.npmjs.com/package/atomic-design-system-scss

npmjs.com. (2024). npmjs.com. Získáno [online] Únor 2024 z npmjs.com. https://www.npmjs.com/package/atomic-design-system-react

Seznam obrázků, tabulek, grafů a zkratek

13.1 Seznam obrázků

Odkazovaný seznam obrázků

13.2 Seznam tabulek

Přílohy