Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Usnadnění přístupu ve webových aplikacích

Plzeň 2019 Josef Lexa

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. května 2019

Josef Lexa

Abstract

Accessibility in web applications

The aim of this thesis is to explain to a reader the issue of web applications accessibility, which is used to ease the access to users with different types of disabilities. At first, this work handles the analysis of standards and specifications defining procedures for making a web accessible. Based on this analysis, it verifies the accessibility of some Czech public administration websites. The main part concerns with design of techniques and creation of a web application that demonstrates accessibility implementation procedures.

This thesis mainly aims to analyse and design techniques to achieve accessibility in web applications, implement a demonstration web application based on these findings and show how to verify web accessibility.

Abstrakt

Cílem této práce je čtenáři vysvětlit problematiku přístupnosti webových aplikací, která slouží k usnadnění přístupu pro uživatele s různým typem postižení. Práce se nejdříve zabývá analýzou standardů a specifikací definujících postupy pro zpřístupnění webu. Na základě této analýzy ověřuje přístupnost některých webů veřejné správy. Hlavní část se zabývá návrhem technik a vytvořením webové aplikace, jež demonstruje postupy pro implementaci přístupnosti.

Hlavním cílem diplomové práce je tedy analýza a návrh technik pro dosažení přístupnosti ve webových aplikacích. Na základě těchto zjištění implementovat demonstrační webovou aplikaci a ukázat postupy, jak přístupnost webu ověřovat.

Obsah

1 Úvod

Cílem práce je analýza problematiky přístupnosti webových aplikací a průzkum souvisejících specifikací a standardů. Přístupnost webových aplikací je důležitá pro zajištění přístupu handicapovým uživatelů a poskytuje sémantické informace, které jsou významné pro kvalitu webu včetně zpracování vyhledávači. Veřejné správě navíc přístupnost ukládá zákon.

Vzhledem ke stále většímu využití webu a vzniku webových aplikací, které disponují komplikovanějšími ovládacími prvky, je také důležitější brát ohled na usnadnění přístupu pro uživatele s různým postižením. Abychom přístup na web pro tyto uživatele usnadnili, musíme dodržovat specifikace a standardy webové přístupnosti.

Práce bude nejdříve zkoumat problematiku usnadnění přístupu z pohledu existujících standardů. Detailněji se pak zaměří na specifikaci WAI-ARIA, jež se zabývá definováním postupů pro přístupnost webu a speciálně webových aplikací. V ČR a EU existuje legislativa upravující přístupnost webů veřejné správy. Na základě ní budou některé z těchto webů analyzovány a zhodnoceny, zda zákon skutečně dodržují. V případě nedodržení přístupnosti budou identifikovány hlavní nedostatky a navrženy konkrétní úpravy.

Poté budou analyzovány techniky pro implementaci přístupnosti webových aplikací a s jejich využitím bude implementována demonstrační přístupná webová aplikace. Tato aplikace bude na závěr otestována a zhodnocena.

Mezi hlavní výstupy této diplomové práce bude patřit:

- analýza standardů a specifikací pro usnadnění přístupu na web,
- doporučení postupů při tvorbě přístupné webové aplikace,
- demonstrační webová aplikace, která může sloužit jako vzor například pro dodavatele webů veřejné správy.

Demonstrační webová aplikace bude nakonec otestována a bude zmíněno několik nástrojů pro ověření přístupnosti na webu včetně vytvořeného rozšíření prohlížeče Google Chrome, které testuje hlavní parametry přístupnosti v reálném čase.

2 Přístupnost webu

V této kapitole jsou zmíněny základní postupy pro tvorbu přístupného webu. Základní rámec, standardy a pomocné materiály pro tvorbu přístupného webu jsou vydávány konsorciem W3C (World Wide Web Consortium) v rámci jeho podskupiny WAI (Web Accessibility Initiative).

2.1 Definice přístupnosti webu

Web je navržen tak, aby uspokojoval potřebu všech uživatelů bez ohledu na jejich software, hardware, jazyk, umístění nebo schopnosti [1]. Často ho využívají lidé s různými typy postižení, pro které by měl být plně přístupný. Tito lidé mohou mít např. poruchy sluchu, zraku, pohybu či kognitivních schopností.

Web tedy může odstranit překážky komunikace a interakce s ním pro handicapované uživatele. Nicméně, pokud nejsou webové stránky, aplikace či různé technologie dobře navrženy, tyto překážky neodstraňují a vylučují tak využití těmito lidmi.

Přístupností webových aplikací tedy rozumíme to, že jsou navrženy a vytvořeny tak, aby je mohli používat lidé s různými postiženími [1]. Konkrétně by měli mít možnost:

- vnímat a porozumět webu,
- navigovat a interagovat s webem,
- přispívat na web.

Usnadnění přístupu na web zahrnuje všechna postižení, která ovlivňují přístup k webu, včetně postižení:

- sluchového,
- poznávacího,
- neurologického,
- fyzického,

- psychického,
- vyjadřovacího,
- zrakového [1].

Přístupnost webu však zvyšuje jeho kvalitu obecně a přináší výhody i pro uživatele bez postižení, jako jsou:

- uživatelé používající mobilní zařízení, chytré hodinky, televize a další zařízení s různými velikostmi displeje, různými typy vstupů a interakce,
- uživatelé s horšími schopnostmi v důsledku stárnutí,
- uživatelé s dočasnými problémy, jako je zlomenina ruky nebo ztracené brýle,
- uživatelé s omezeními vzhledem k situaci, jako je sluneční světlo na displej nebo prostředí, ve kterém nemohou poslouchat zvuk,
- uživatelé s pomalým připojením k internetu [2].

Web je stále důležitějším zdrojem ve spoustě aspektů života: vzdělání, zaměstnání, obchod, vláda, zdravotnictví, bankovnictví a další. I proto je nezbytné, aby byla zajištěna přístupnost, a tou docílena stejná příležitost pro všechny uživatele s různými schopnostmi. Přístup k informačním a komunikačním technologiím, včetně internetu, je definován jako základní lidské právo v Úmluvě OSN o právech osob s postižením [2].

Tím, že web nabízí možnost dobrého přístupu pro uživatele s handicapem, stává se tak možností a cestou k překonání bariér. Toho mohou využít jak weby veřejné správy, které musí poskytovat informace přístupné bez ohledu na zdravotní stav uživatele, tak např. různé společnosti k propagaci, tvoření nástrojů či online obchodů.

Přístupnost se překrývá s dalšími postupy, jako je optimalizace webů pro mobilní zařízení, nezávislost webu na zařízení, použitelnost či optimalizace pro vyhledávače (SEO). Dobře přístupné weby jsou lépe indexovány vyhledávači, čímž může stoupnout jejich návštěvnost, mohou se snížit náklady na jejich údržbu a prokazují sociální odpovědnost autora webu [2].

2.2 Základní komponenty přístupnosti webu

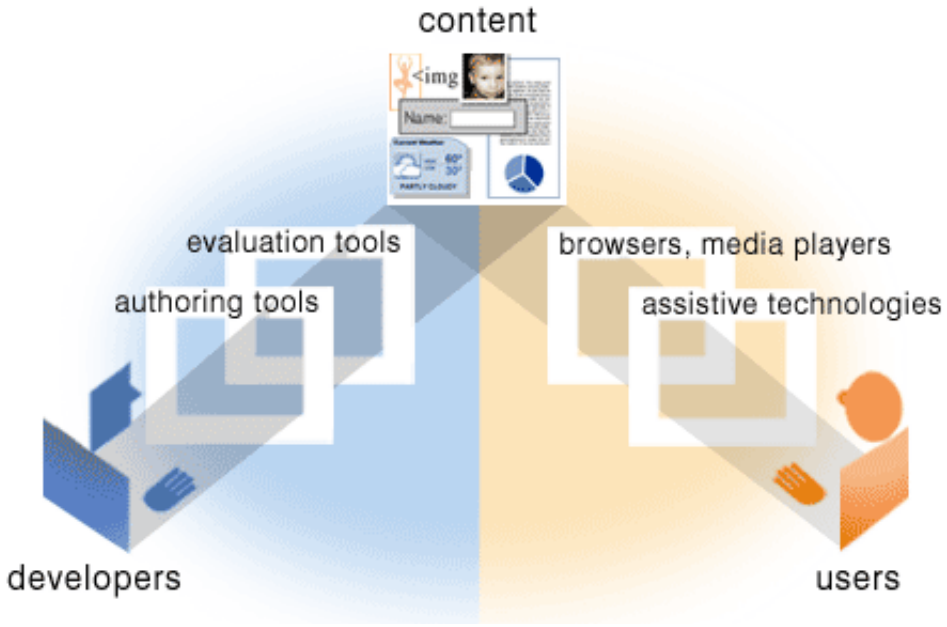
Aby byl web přístupný, závisí na vzájemné spolupráci několika komponent, které se týkají jak vývoje webu, tak jeho následné interakce s uživatelem [3]. Komponentou je zde myšlen nástroj, subjekt a další části, které jsou pro přístupnost webu důležité. Tyto komponenty se dají rozdělit do tří kategorií.

1. **Vývojáři** - jsou to designéři, kodéři a další, kteří tvoří obsah webu a využívají následující komponenty:
   - **vývojové nástroje** pro tvorbu webu a webových aplikací,
   - **nástroje pro kontrolu** - kontrola webové přístupnosti (např. Chrome DevTools), HTML a CSS validátory.
2. **Obsah** - informace na webové stránce obsahující:
   - základní informace jako je text, obrázky a zvuk
   - kód nebo značení, které definuje strukturu, vzhled apod.
3. **Uživatelé** - využívající pro přístupn na web následující komponenty:
   - **webové prohlížeče, přehrávače médií**, a další uživatelské nástroje
   - **asistenční nástroje** jako jsou čtečky obrazovky, alternativní klávesnice, čtečky s převodem na braillovo písmo a další
   - **znalosti** uživatelů a zkušenosti s asistenčními nástroji

Na obrázku 2.1 je vidět, jak komponenty souvisejí. Vývojáři vytváří webový obsah za použití vývojových nástrojů a nástrojů pro kontrolu. Uživatelé pak využívají webové prohlížeče, přehrávače médií, asistenční nástroje a další pro jeho prohlížení.

Nutnost vzájemné spolupráce těchto komponent je ukázáno na základním příkladu přístupnosti webu, kterým je textová alternativa u obrázků:

- **technická specifikace** definuje alternativní text (např. pro HTML ho definuje v atributu `alt` elementu pro obrázek `img`)
- **WAI standardy** definují, jak implementovat alternativní text pro přístupnost v jednotlivých komponentách
- **vývojáři** poskytují odpovídající alternativní text



Obrázek 2.1: Souvislost komponent webové přístupnosti [3]

- **vývojové nástroje** umožňují a napomáhají k vytvoření alternativního textu vývojářem
- **nástroje pro kontrolu** se využívají k ověření existence alternativního textu
- **webové prohlížeče** a další nástroje poskytují lidsky a strojově čitelný alternativní text

Pokud je funkcionalita přístupnosti efektivně implementována v jedné komponentě, v ostatních komponentách je pak implementace jednodušší. Příkladem může být vývojové prostředí, kde vývojáři chtějí implementovat postupy usnadnění. Budou pravděpodobně požadovat, aby jejich nástroj tvorbu přístupného webu usnadňoval [3].

2.3 Nástroje pro usnadnění přístupu

Nástroje pro usnadnění přístupu webu resp. asistenční nástroje jsou nástroje, které používají lidé s různým postižením k interakci s webem [5].

Tito lidé využívají web různými cestami, v závislosti na jejich individuálních potřebách a preferencích. Někdy jim postačuje pouze úprava nebo rozšíření softwaru (lupa ve webovém prohlížeči), se kterým pracují a někdy

využívají specializovaný software a hardware, jenž jim pomáhá provádět určité úkony (čtečka obrazovky) [4].

Mezi základní asistenční nástroje patří:

- **Čtečky obrazovky (Screen Reader):** Software využívaný nevidomými nebo jinak zrakově postiženými uživateli ke čtení obsahu na obrazovce počítače nebo jiného zařízení. Z nejznámějších je to JAWS pro Windows, NVDA nebo Voiceover pro Mac.
- **Software pro zvětšení obrazovky:** Umožňuje uživatelům kontrolovat a měnit velikost textu i grafických prvků na obrazovce. Na rozdíl od klasických nástrojů typu „lupa“, tento software umožňuje zvětšení pouze textu ve vztahu ke zbylým prvkům na obrazovce.
- **Software pro vstup řeči:** Pro uživatele s motorickými problémy přináší způsob ovládání a psaní textu pomocí řeči. Uživatel tak pomocí řeči může například vyplňovat formulář, kliknout na tlačítko nebo vybrat položku z nabídky. Pro platformu Windows a Mac je to nástroj Dragon NaturallySpeaking.
- **Alternativní vstupní zařízení:** Někteří uživatelé nemohou využívat klávesnici nebo myš k ovládání počítače. Tito uživatelé mohou využít různá zařízení jako:
  - ukazovátko přidělané na hlavě,
  - sledování pohybu nebo očí,
  - jednoduchá vstupní zařízení, jako např. jednotlačítkový ovladač, který může být využíván v kombinaci s některým z předchozích nástrojů.

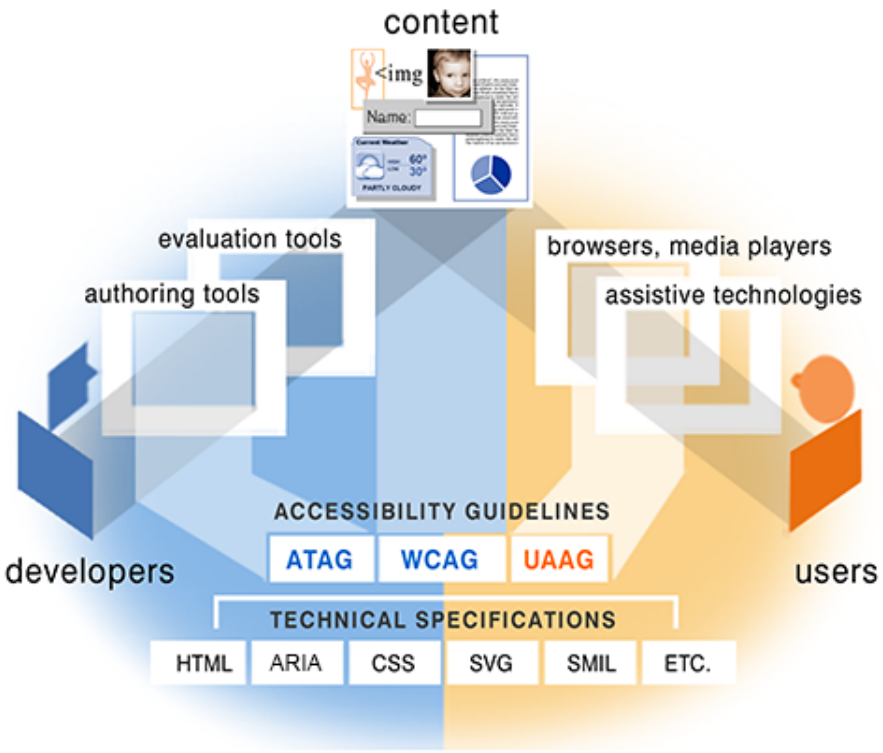
2.4 Standardy a specifikace pro přístupnost webu

Standardy pro vývoj přístupného webu pro všechny uživatele vydává konsorcium W3C resp. jeho podskupina WAI. Standardy se rozdělují na průvodce přístupností a technické specifikace pro přístupnost.

Průvodci přístupností se rozdělují do tří skupin podle komponent, kterých se týkají. Jsou to:

- **Authoring Tool Accessibility Guidelines (ATAG)** zabývající se nástroji pro vývoj přístupných webů,
- **Web Content Accessibility Guidelines (WCAG)** definující standardy a specifikace pro vývoj přístupného webu,
- **User Agent Accessibility Guidelines (UAAG)** popisující zpřístupnění asistenčních nástrojů.

Rozdělení těchto průvodců je na obrázku 2.2. Na obrázku je vidět, kterými komponentami se jednotlivý průvodci zabývají.



Obrázek 2.2: Rozdělení průvodců přístupnosti webu mezi komponenty [3]

2.4.1 Průvodce ATAG

Nástroje pro tvorbu obsahu jsou vývojové nástroje a služby, které vývojáři, designéři a další autoři webového obsahu používají k jeho tvorbě [9]. Může se jednat jak o tvorbu statických webových stránek, tak dynamických webových aplikací.

Dokumenty průvodce ATAG popisují jak:

- zpřístupnit nástroje pro tvorbu webového obsahu, aby je mohly používat osoby s postižením,
- pomáhat autorům vytvářet přístupnější webový obsah, tedy umožnit a podporovat produkování webového obsahu, který je v souladu s WCAG průvodcem zmíněným dále [9].

Průvodce ATAG je tedy primárně určen vývojářům nástrojů pro tvorbu webového obsahu, které zahrnují kromě klasických HTML editorů také například software pro generování webových stránek, konvertor textového obsahu do HTML, web jenž umožňuje uživateli vytvářet obsah, jako je blog nebo wiki a další [9].

2.4.2 Průvodce WCAG

Z pohledu vývoje přístupného webu a webových aplikací je důležité se zaměřit na standardy a postupy definované v průvodci WCAG, které se týkají samotného obsahu webu.

Standardy popisují principy pro tvorbu přístupného webu rozdělené do 4 okruhů. Každý okruh představuje vlastnost webu, kterou by měl splňovat. Jelikož je průvodce standardy velmi rozsáhlý, je dobré si alespoň vysvětlit, čeho se jednotlivé okruhy týkají. Weby podle WCAG standardu by tedy měly splňovat následující vlastnosti.

- **Vnímatelnost** - informace a komponenty uživatelského rozhraní musí být uživatelům prezentovány ve formě, které uživatelé porozumí.
  - poskytnutí textové alternativy pro netextový obsah
  - poskytnutí titulků a jiných alternativ pro multimediální obsah
  - vytvoření obsahu, který může být prezentován v různých formách, včetně asistenčních nástrojů, bez ztráty jeho významu
  - ulehčit uživatelům vidět a slyšet obsah
- **Ovladatelnost** - prvky uživatelského rozhraní musí být ovladatelné pro všechny uživatele bez ohledu na jejich schopnosti.
  - veškerá funkcionalita by měla být dostupná pomocí klávesnice
  - uživatel by měl mít dostatek času na přečtení a porozumění obsahu, což se týká hlavně časových limitů na interakci

  - usnadnění navigace a nalezení požadovaného obsahu
  - snažit se dělat web jednoduchý také pro další vstupní zařízení, které mohou lidé s postižením využívat

- **Srozumitelnost** - informace a ovládání uživatelského rozhraní musí být srozumitelné.
  - vytváření srozumitelného a čitelného obsahu
  - vytváření obsahu a operací, které mohou být předvídatelné
  - pomoci uživatelům vyhnout se chybám a případně je opravovat či kontrolovat

- **Robustnost** - maximalizace kompatibility s aktuálními i budoucími asistenčními nástroji, tvořit tedy obsah tak, aby mu porozuměly jak aktuální asistenční nástroje, tak ty, které vzniknou v budoucnu.

Pokud je některý z výše zmíněných postupů zanedbán nebo špatně implementován, uživatelé s postižením nebudou schopni takto vytvořený web plně využívat [10]. V některých případech, kdy např. není poskytnuta ovladatelnost webu klávesnicí, nejsou někteří uživatelé schopni takto vytvořený web vůbec používat.

2.4.3 Průvodce UAAG

Postupy definované v UAAG vysvětlují, jak zpřístupnit uživatelské nástroje lidem s postižením [11]. V jeho kontextu jsou uživatelskými nástroji myšleny webové prohlížeče, rozšíření webových prohlížečů, přehrávače multimédií, čtečky a jiné nástroje zobrazující webový obsah. Uživatelský nástroj, jenž splňuje UAAG by měl být přístupný lidem s postižením a zároveň by měl být otevřený pro komunikaci s dalšími technologiemi jakými jsou např. asistenční nástroje [11].

UAAG je tedy primárně určen vývojářům uživatelských nástrojů. Mimo jiné specifikuje také postup, jak hodnotit míru přístupnosti uživatelských nástrojů. To může být využito např. při volbě nástroje v podniku nebo pro hlášení nedostatků nástroje jeho autorovi.

2.4.4 Technické specifikace

Technické specifikace řeší aplikování zmíněných standardů přístupnosti pro konkrétní využití.

Jednou z nejznámějších a nejvyužívanějších je ARIA (Accessible Rich Internet Applications). Poskytuje autorům webového obsahu sémantiku pro zprostředkování chování jednotlivých prvků webu. Tato sémantika slouží jako informace pro asistenční nástroje. Této specifikaci se detailně věnuje kapitola 3.

Dalším příkladem jsou specifikace pro zvuk a video. Definují textové alternativy zvuku a videa. Nejznámějším je WebVTT (The Web Video Text Tracks Format) - formát pro titulky, textový popis videa a další metadata, které jsou časově spojená se zvukem a videem [12].

V neposlední řadě jsou to specifikace pro kontrolu a hodnocení přístupnosti. Existují následující dva základní zdroje, které slouží pro podporu vývoje nástrojů kontroly a vyhodnocování přístupnosti.

- **Accessibility Conformance Testing (ACT)** - vytváří a dokumentuje pravidla pro testování souladu webového obsahu se standardy přístupnosti.
- **Evaluation and Report Language (EARL)** - strojově čitelný formát pro popis výsledků testování přístupnosti.

2.5 Zákony a normy v ČR a EU

Jak bylo již zmíněno, přístupnost není jen otázkou uživatelů s různým postižením. Je to jeden ze základních aspektů dobře vytvořeného webu. Weby veřejné správy jsou na základě zákona nuceny publikovat informace s využitím pravidel přístupného webu. Jejich povinností je poskytovat informace všem občanům bez rozdílu.

Povinnost zveřejnění informací v přístupné formě všem uživatelům byla stanovena zákonem 365/2000 sb., zákon byl pak novelizován a upraven zákonem 81/2006 sb. Tato zákonná úprava nařizuje všem webům veřejné správy, aby byly vytvořeny s využitím pravidel přístupného webu [6]. Tato pravidla se nachází v metodickém pokynu vyhlášky č. 64/2008 Sb, který vychází z již zmíněného standardu WCAG [7].

V rámci Evropské unie spadá problematika přístupnosti webu pod dlouhodobý program označovaný jako eAccessibility, zaměřující se obecně na práva nějakým způsobem postižených osob s důrazem na elektronickou výměnu dat. Tento program považuje za svojí výchozí metodiku WCAG [6]. Pro členské státy EU to znamená, že by měly vytvářet veřejné webové stránky,

které jsou v souladu s metodikou WCAG resp. mají svou metodiku, která z WCAG vychází, jako je to u nás v ČR.

Metodika Web Content Accessibility Guidelines - WCAG 2.0 je platnou ISO normou: ISO/IEC 40500:2012 [8]. Norma ISO 40500 je stejná jako původní standard WCAG 2.0 od W3C. Od roku 2008, kdy tato norma byla vydána, ji přijala spousta zemí jako svůj standard. Výhoda ISO 40500 je v tom, že některé státy využívající pouze ISO standardy, mohou přijmout WCAG jako svůj standard pro přístupnost webů.

3 Specifikace WAI-ARIA

Accessible Rich Internet Applications (ARIA) je sada postupů ke zpřístupnění webu od skupiny Web Accessibility Initiative (WAI) konsorcia W3C. Definuje způsob, jak zpřístupnit webový obsah a webové aplikace lidem s různým zdravotním postižením. Umožňuje zejména zpřístupnění dynamického obsahu s pokročilými ovládacími prvky uživatelského rozhraní, dnes hojně vyvíjenými především v JavaScript, HTML a souvisejících technologiích. Aktuální verze specifikace ARIA je 1.1, která byla také využita při tvorbě této diplomové práce.

3.1 Využití ARIA

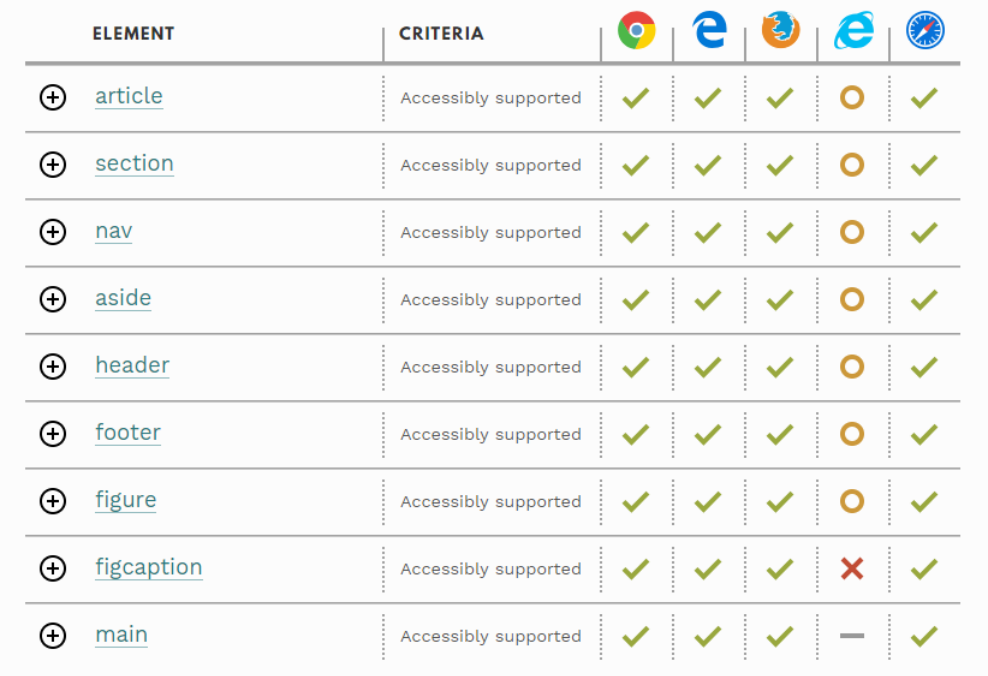
Vzhledem ke stále většímu spektru využití webových technologií, se také více využívají ovládací prvky, které nejsou standardními elementy HTML. Takto vytvořené ovládací prvky jsou pak těžko rozpoznatelné asistenčními nástroji.

Kromě ovládacích prvků jsou to také oblasti stránky. Pro ty již existují v HTML5 standardní elementy. Jak je vidět na obrázku 3.1, tyto elementy mají v nejnovějších verzích prohlížečů z pohledu přístupnosti plnou podporu. I přesto je stále dobré k jejich popisu používat také ARIA, neboť někteří uživatelé mohou mít starší verze prohlížečů bez této podpory.

Z technického hlediska ARIA poskytuje standard pro přidávání atributů HTML elementům, které slouží pro identifikaci jejich významu a stavu. Těmto atributům jsou pak schopny porozumět asistenční nástroje, a díky nim interpretovat obsah webové stránky v čitelné formě jejich uživatelům.

ARIA tedy přináší postupy k řešení problémů s přístupností tam, kde samotné HTML nestačí. Vývojářům webu umožňuje vytvářet:

- Role, pomocí nichž je možné popsat jednotlivé typy elementů, například `menu`, `treeitem` a další.
- Role, pomocí nichž lze popsat strukturu resp. oblasti webové stránky, jako jsou nadpisy, sekce či tabulky.
- Vlastnosti, kterými lze popsat v jakém stavu se element nachází, pro představu například `checked` u zaškrtávacího políčka.



ELEMENT	CRITERIA					
article	Accessibly supported	✓	✓	✓	○	✓
section	Accessibly supported	✓	✓	✓	○	✓
nav	Accessibly supported	✓	✓	✓	○	✓
aside	Accessibly supported	✓	✓	✓	○	✓
header	Accessibly supported	✓	✓	✓	○	✓
footer	Accessibly supported	✓	✓	✓	○	✓
figure	Accessibly supported	✓	✓	✓	○	✓
figcaption	Accessibly supported	✓	✓	✓	✗	✓
main	Accessibly supported	✓	✓	✓	–	✓

Obrázek 3.1: Podpora standardních HTML5 elementů v prohlížečích z pohledu přístupnosti [13]

- Definování tzv. živých oblastí stránky, které se nějakým způsobem aktualizují nebo objevují v závislosti na jiných akcích. Typickým příkladem je vyskakovací okno (alert).
- Způsob, jak ošetřit přístupnost z klávesnici pro již zmíněné prvky webové stránky [14].

3.2 ARIA role

Role je hlavním indikátorem typu HTML elementu. Nastavuje se pomocí atributu `role` a umožňuje vývojářům prezentovat, a tím podporovat interakci s elementem takovým způsobem, který je uživatelem, jenž využívá asistenční nástroj očekáván. V ARIA specifikaci je ke každé roli definováno několik informací. Zde je výčet těch nejdůležitějších:

- abstraktní resp. rodičovské role, ze které definovaná role vychází,
- povinné vlastnosti a stavy,
- další podporované vlastnosti a stavy,

- zděděné vlastnosti a stavy z abstraktních resp. rodičovských rolí
- požadované prvky, které musí být vytvořeny vnořením do aktuálně definované role, například role `list` musí vlastnit alespoň jeden prvek s rolí `listitem` nebo `group`,
- role požadovaného předka, což je opakem předchozího bodu, například role `listitem` musí být vnitřním prvkem role `list`,
- jakým způsobem se získává popisná informace o prvku - buď je definována autorem například skrze atribut `aria-label` nebo je zděděna z textové hodnoty prvku.

3.2.1 Kategorie rolí

Jak bylo již zmíněno, specifikace primárně rozděluje role podle jejich využití na definování prvků a definování struktury webové stránky. Kromě toho určuje šest kategorií, podle kterých jsou role detailněji členěny. Toto členění může být nápomocné pro správné pochopení a pozdější využívání rolí. Role jsou členěny následujícím způsobem.

1. **Abstraktní role (Abstract Roles)**: Tyto role se nevyužívají pro zpřístupnění prvků, slouží pouze k udržení hierarchie rolí v rámci standardu. Je to například role `input`, která je rodičem rolí `checkbox`, `textbox` a dalších.
2. **Komponentové role (Widget Roles)**: Používají se pro popis uživatelských prvků resp. komponent. Můžou to být buď samostatně stojící komponenty nebo části větších složených komponent. Příkladem role pro samostatně stojící komponentu je `button` nebo `link`. Složitější komponenta je typicky nějaký kontejner, který udržuje další obsažené komponenty. Příkladem je `menu`, které se skládá z jednotlivých položek označených rolí `menuitem`.
3. **Role pro strukturu dokumentu (Document Structure Roles)**: Definují popis struktury obsahu na dané webové stránce. Komponenty popisované těmito rolemi obvykle nejsou interaktivní. Struktura dokumentu může opět obsahovat nějaké seskupující komponenty, v tomto případě je to `list` s položkami označenými rolí `listitem`. Dalšími zástupci těchto rolí jsou například `img` či `table`.

4. **Role pro oblasti dokumentu (Landmark Roles)**: Používají se pro rozdělení webové stránky na oblasti. Každá oblast seskupuje prvky v ní definované do jednoho celku. Typickými zástupci této kategorie jsou `main` pro definování hlavní oblasti dokumentu, `navigation` pro seskupení navigace a také například `search` pro oblast s vyhledáváním.

5. **Role pro živé oblasti (Live Region Roles)**: Jedná se živé oblasti, tedy prvky, ve kterých se mění obsah. V kombinaci se stavovými atributy, které jsou zmíněné v následující podkapitole, tvoří mechanismus, jenž skrze asistenční nástroje dokáže při změně stavu upozornit uživatele. Jsou to např. role `alert` či `status`.

6. **Role pro vyskakovací okna (Window Roles)**: Poslední kategorie rolí slouží k označení vyskakovacích oken resp. dialogů v rámci webového obsahu. Nejedná se o živé oblasti, protože takto zobrazená okna obsahují aktivní prvek stránky, čímž asistenční nástroje získávají informaci o jejich stavu. Jsou to dvě role `alertdialog` pro dialog s upozorněním a `dialog` pro dialog k interakci s uživatelem.

3.2.2 Ukázka použití rolí

Na první ukázce kódu 3.1 je vidět základní využití role. V tomto případě se jedná o roli `group` z kategorie rolí pro strukturu dokumentu. Tato role zajistí, že asistenční nástroj bude uživateli prezentovat, že se jedná o nějakou skupinu tlačítek.

Kód 3.1: Ukázka využití ARIA role group

```
<div class="btn-group" role="group">
  <button type="button" class="btn">Prvni akce</button>
  <button type="button" class="btn">Druha akce</button>
</div>
```

Druhou ukázkou je kód 3.2 prezentující využití rolí pro definování oblastí dokumentu. Jak bylo zmíněno, pro definování oblastí dokumentu je dobré využívat kromě standardních HTML5 elementů také ARIA role.

Kód 3.2: Ukázka využití ARIA rolí pro oblasti

```
<header role="banner">
  Nazev webu
  <form role="search">
```

```
    <!-- Vyhledavani -->
  </form>
</header>
<nav role="navigation">
  <!-- Navigace -->
</nav>
<main role="main">
  <!-- Hlavni obsah -->
</main>
```

3.3 ARIA vlastnosti a stavy

ARIA kromě rolí poskytuje také kolekci vlastností a stavů, které jsou využívány pro podporu zpřístupnění obsahu webů. Společně s rolemi poskytují kompletní mechanismus zpřístupnění webu pro asistenční nástroje. Změny vlastností a stavů vyvolávají znamení pro asistenční nástroje, které mohou uživatele o změnách upozornit.

Vlastnosti a stavy přinášejí podobné rozšíření pro popis prvků webového obsahu, jsou mezi nimi ovšem určité rozdíly. Všechny vlastnosti a stavy se definují jako atribut začínající prefixem `aria-*`. Obojí poskytuje specifické informace o prvcích a tvoří součást definice vycházející z role prvku, nicméně koncepčně jsou odlišné. Jedním z hlavních rozdílu je to, že hodnoty vlastností se většinou v průběhu životního cyklu webové stránky resp. aplikace nemění. Pro příklad může být zmíněna vlastnost `aria-label` označující popis prvku. Je velmi pravděpodobné, že tento popis se měnit nebude. Naproti tomu `aria-checked` označující stav výběrových resp. přepínacích polí, se bude zřejmě během životního cyklu stránky měnit. Je dobré podotknout, že tento rozdíl není pravidlem. Například vlastnost `aria-valuetext`, nesoucí textovou alternativu pro vlastnost `aria-valuenow`, která udává aktuální hodnotu rozsahových nebo průběhových ukazatelů, se bude měnit často. Vzhledem k tomu, že rozdíl mezi vlastnostmi a stavy je velmi malý a často těžko rozlišitelný, obecně se jako celek označují za ARIA atributy. Stejně tak jsou označovány i dále v textu.

V ARIA specifikaci jsou ke každému atributu definovány tyto informace:

- definice atributu, která může pomoci pochopit jeho význam, občas je vysvětleno na standardním využití v HTML,
- role, se kterými může být atribut použit,

- role, které zdědí daný atribut z rolí definovaných jako jejich předchůdci viz kategorie abstraktních rolí,
- typ hodnot, kterých může atribut nabývat.

3.3.1 Kategorie atributů

Atributy se stejně jako role dělí do několika kategorií. Rozdělení může přinést snazší porozumění a orientaci při pozdější implementaci přístupných komponent. Atributy jsou děleny do následujících čtyř kategorií podle jejich využití.

1. **Atributy pro komponenty (Widget Attributes)**: Tato kategorie obsahuje atributy specifické pro uživatelské prvky, které slouží pro interakci s uživateli a přijímají uživatelské vstupy. Slouží k podpoře elementů popsaných pomocí komponentových rolí. Příkladem takového atributu je například `aria-pressed` určující stav stisknutí přepínacího tlačítka.
2. **Atributy pro živé oblasti (Live Region Attributes)**: Tyto atributy jsou specifické pro živé oblasti ve webových aplikacích. Jsou to atributy, které mohou být aplikovány na libovolný prvek. Přinášejí mechanismus, pomocí kterého se upozorňují asistenční nástroje, že obsah označených prvků těmito atributy se může měnit, aniž by se prvek stal aktivním prvkem stránky. Samotné atributy pak určují, jak by měly asistenční prvky na změny reagovat. Asi nejpoužívanějším zástupcem těchto atributů je `aria-live`.
3. **Atributy pro drag-and-drop (Drag-and-Drop Attributes)**: v této kategorii jsou řešeny atributy, které slouží k popsání oblastí pro drag-and-drop. Může se jednat o oblasti sloužící k nahrání souborů nebo o prvky webu, které lze na webové stránce přetahovat nebo s nimi jinak pohybovat pomocí myši. Příkladem je atribut `aria-dropeffect` popisující, co se s přesouvaným prvkem stane, když bude přesunut do cílové oblasti.
4. **Atributy pro vztahy (Relationship Attributes)**: Poslední kategorie obsahuje atributy, které označují vztahy nebo asociace mezi prvky, které nelze jednoznačně určit ze struktury dokumentu. Nejjednodušším příkladem je atribut `aria-labelledby` nesoucí identifikátor prvku, který poskytuje textový popis pro prvek na kterém je tento atribut uveden.

3.3.2 Ukázka použití atributů

První příklad poukazuje na využití ARIA atributu `aria-describedby`. Je to příklad z kategorie atributů určujících vztahy. Na výpisu kódu 3.3 je vidět jak se atribut přidává k referenci na jiný prvek, jenž slouží jako jeho popis.

Kód 3.3: Využití ARIA atributu aria-describedby

```
<label for="username">Uzivatelske jmeno</label>
<input type="text" id="username" aria-describedby="
    username-tip" required>
<div role="tooltip" id="username-tip">Uzivatelske jmeno je
    emailova adresa</div>
```

Za zmínku stojí vysvětlit rozdíl mezi `aria-describedby` a jemu podobnému `aria-labelledby`. Z pohledu asistenčních nástrojů je funkcionalita atributů stejná, rozdíl je pouze v doporučeném využití. První zmíněný by se měl používat tam, kde se prvek popisuje nějakým delším textem. Druhý atribut by se pak měl používat na krátký popisek prvku [15].

Dalším příkladem je kód 3.4 obsahující jednoduchou komponentu pro řazení seznamu pomocí přepínacích tlačítek. V tomto příkladu se objevují hned tři ARIA atributy. První atribut `aria-label` poskytuje popisek komponenty. Atribut `aria-controls` identifikuje prvek, jehož obsah je tímto prvkem obsluhován. Posledním atributem je `aria-pressed`, jenž určuje stav přepínacího tlačítka.

Kód 3.4: Využití ARIA pro řazení v seznamu

```
<div role="toolbar" aria-label="nastaveni razeni"
    aria-controls="sortable">
  <button aria-pressed="true">A-Z</button>
  <button aria-pressed="false">Z-A</button>
</div>
<ul id="sortable" tabindex="-1">
  <li>Brno</li>
  <li>Liberec</li>
  <li>Praha</li>
</ul>
```

Zde je také dobře rozpoznatelný rozdíl mezi vlastnostmi a stavy zmíněný výše. Titulek a reference na kontrolovaný prvek jsou vlastnosti neboť se zcela jistě nastaví pouze na začátku životního cyklu stránky, kdežto stav přepínacího tlačítka se bude po každém kliknutí dynamicky měnit.

3.4 Aktivní prvek stránky

Webové stránky, včetně webových aplikací, by měly mít vždy element, který je aktivní resp. má focus. Samotná specifikace ARIA v souvislosti s touto problematikou vyzývá autory k několika postupům:

- aktivní prvek stránky by neměl být za žádných situací ze stránky resp. Z jejího DOM (Document Object Model) odstraněn,
- aktivní prvek by neměl být posunut nebo skryt mimo obrazovku,
- všechny interaktivní prvky by měly mít možnost stát se aktivním prvek,
- všechny prvky kompozitních komponent by měly mít možnost stát se aktivním prvek nebo musí mít zdokumentovanou alternativní metodu pro jejich dosažení, například pomocí klávesové zkratky,
- mělo by se zachovávat zřejmé pořadí prvků, které je v logickém uskupení a je dosažitelné pomocí klávesy tabulátoru pro uživatele používající pouze klávesnici [16].

Specifikace ARIA také definuje pro jednotlivé kompozitní komponenty, jak se mají obsluhovat z pohledu aktivního prvku, při pohybu uživatele pouze pomocí klávesnice. Složitější komponenty totiž vyžadují, aby jejich autor obsluhu pomocí klávesnice včetně určování aktivního prvku sám implementoval. Pro příklad může být uvedena vyskakovací nabídka vyvolaná pomocí tlačítka. To není standardní HTML komponenta a její autor musí například vyřešit to, aby se při otevření nabídky přesunul aktivní prvek stránky na první položku nabídky.

Výhoda obsluhy aktivního prvku autorem komponenty může být také v případě, kdy je potřeba si pamatovat poslední aktivní prvek a při vrácení focus na komponentu ho označit [16]. Příkladem může být stromová struktura nebo formulář během kterého vyvoláme dialog pro zadání komplexnější položky (datum, čas).

Posledním problémem, na který ARIA specifikace poukazuje, je obsluha vlastní klávesy pro změnu aktivního prvku v rámci kompozitních komponent. To je užitečné hlavně u komplexnějších webových aplikací. Ty obsahují spoustu takových komponent a je dobré pomocí klávesy Tab (standardní klávesa pro změnu aktivního prvku) posouvat focus mezi komponentami

a změnu aktivního prvku v rámci komponenty řešit například pomocí kláves šipek.

3.5 Průvodce implementací ARIA

Konsorcium W3C kromě samotného specifikace vydává také dokument WAI-ARIA Authoring Practises [17]. Ten poskytuje pochopení a popisuje správné postupy k vytváření přístupných webových aplikací. Oproti standardu, který se zabývá teoretickým popisem jednotlivých ARIA rolí a atributů, tento dokument popisuje jednotlivé uživatelské komponenty a jak tyto komponenty pomocí ARIA zpřístupnit. Dokument cílí hlavně na vývojáře webových aplikací.

Dokument se skládá z několika částí. Nejprve čtenáře seznamuje obecně s problematikou přístupnosti a uvádí odkazy na jednotlivé části specifikace ARIA. Zajímavým bodem je pak část označená „No ARIA is better then Bad ARIA“. Poukazuje na fakt, že špatná implementace přístupnosti může mít fatální následky pro uživatele spoléhající na asistenční nástroje. Z funkčního hlediska jsou ARIA role a atributy pro asistenční nástroje stejné jako CSS pro vizuální vzhled. Nevhodně vytvořený vizuální styl prvku v CSS může také změnit pochopení jeho účelu.

Z pohledu implementace přístupnosti webu je přidání role na prvek resp. na vytvářenou komponentu v podstatě slib vývojáře, že poskytne veškerou funkcionalitu definovanou ve výše zmiňovaném dokumentu. Konkrétně v sekci „Design Patterns and Widgets“, kde pro každou komponentu je její implementace detailně popsána z několika pohledů:

- vymezení komponenty, její využití a obecná funkcionalita,
- interakce pomocí klávesnice včetně klávesových zkratek,
- popis pro její zpřístupnění a definice role a atributů, které jsou k tomu určeny,
- uvedení příkladu implementace komponenty.

Kromě popisu komponent je v dokumentu také detailně vysvětleno rozdělení webových aplikací na Landmark Regions, které byly zmíněny v kapitole 3.2.1.

Díky tomuto dokumentu mohou tedy weboví vývojáři při implementaci nových nebo zpřístupňování již existujících webových komponent využívat standardizovaných postupů, které zaručují správné zpřístupnění pro uživatele s různým typem postižení.

4 Analýza webů veřejné správy

Tato kapitola se nejprve zabývá průzkumem dostupných čteček obrazovek a poté analýzou přístupnosti některých webů veřejné správy. Jak bylo zmíněno v kapitole 2.5, weby veřejné správy mají za povinnost být v přístupné formě pro všechny uživatele. Weby budou ověřeny s pomocí čtečky obrazovky a analýzou jejich kódu pro identifikaci problémů s přístupností.

4.1 Čtečky obrazovky

Čtečka obrazovky je software, který rozpoznává obsah na ploše a ve webovém prohlížeči a dokáže ho převést na řeč nebo Braillovo písmo. Čtečky většinou poskytují další funkcionalitu, jako jsou klávesové zkratky, různé režimy pro rozpoznání obsahu a interakce s ním nebo zvýraznění právě čteného textu [4].

Existuje množství čteček obrazovek, které se liší jak jejich funkcionalitou, kterou přináší, tak operačním systémem a platformou využití. Nejznámější čtečky obrazovek jsou zmíněny dále.

- **VoiceOver**: vestavěná čtečka operačního systému macOS a iOS od firmy Apple.
- **JAWS**: čtečka pro operační systém Windows vyvíjena firmou Freedom Scientific, je ovšem dostupná pouze v placené verzi.
- **NVDA**: open source čtečka pro Windows vyvíjna společností NV Access.

Společnost NV Access se kromě samotného vývoje čtečky NVDA podílí charitativní formou na rozvoji přístupnosti zrakově postižených uživatelů k počítači. Čtečka NVDA byla použita v rámci experimentu přístupnosti webů veřejné správy a později pro testování přístupnosti vyvíjené aplikace.

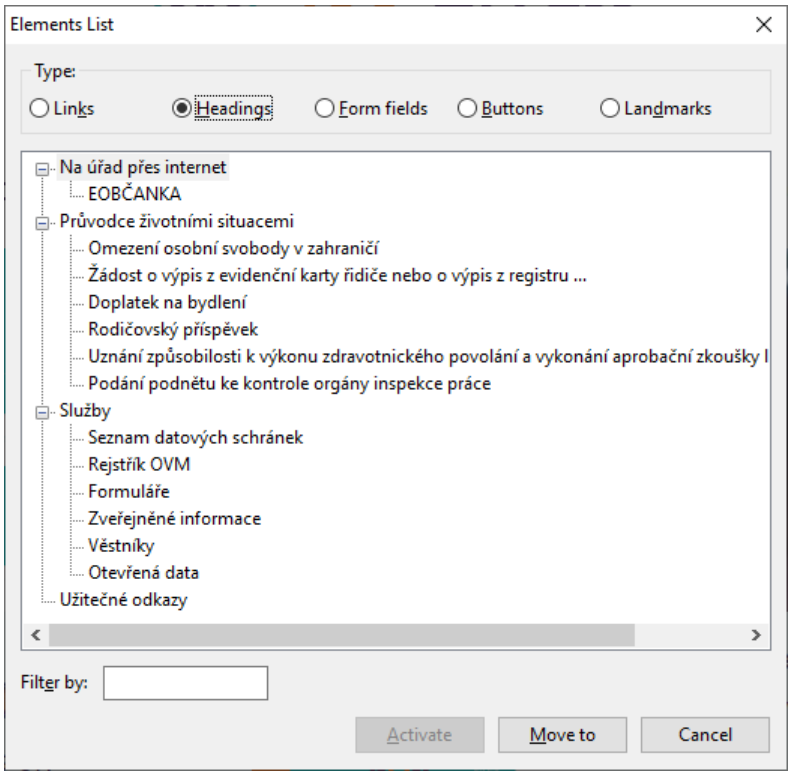
NVDA je modální čtečka obrazovky. To znamená, že disponuje dvěma rozdílnými režimy. Režimem pro předčítání obsahu a režimem pro interakci

s ovládacími prvky. To čtečce umožňuje využívat znakových kláves a klávesových zkratek v obou režimech rozdílným způsobem. Mezi jednotlivými režimy se přepíná klávesovou zkratkou Caps Lock + Mezerník. Prvním režimem je „browse mode“ neboli režim procházení, který slouží pro procházení obsahu včetně možných přeskoků například na následující nadpis nebo na následující prvek formuláře. Ke zmiňovaným přeskokům se používají znakové klávesy. Některé z nich jsou pro ukázku uvedeny v tabulce 4.1. Pro skok na předchozí prvek se znakové klávesy používají v kombinaci s klávesou Shift. V tomto režimu tedy není možné psát a jinak interagovat s prvky na webové stránce. K tomu slouží druhý režim, kterým je „forms mode“ neboli režim formuláře. Ten slouží pro vyplňování formulářových polí a jsou při něm zrušené všechny operace, které je možné vyvolat v režimu procházení přes znakové klávesy. V některých dokumentacích je také nazýván jako „focus mode“, což jak z názvu napovídá, jedná se o režim, při kterém čtečka předčítá obsah podle toho, kde se aktuálně na stránce nachází aktivní prvek. Tento režim je standardním režimem všech čteček obrazovek, které nejsou modální.

Klávesa	**Provede skok na následující**
H	nadpis
B	tlačítko
k	odkaz
F	pole formuláře
E	editovací pole formuláře
C	combobox

Tabulka 4.1: Klávesy pro navigaci obsahem v režimu procházení

Zajímavou funkcionalitou NVDA je také „Elements list“. Vyvolá se klávesovou zkratkou Caps Lock + F7 a slouží k zobrazení seznamu prvků webové stránky. Jak je vidět na obrázku 4.1, seznam poskytuje několik typů prvků, na které je možno nahlížet. Jsou to odkazy, nadpisy, formulářové prvky, tlačítka a oblasti. Zde se potvrzuje důležitost struktury a správného popisování jednotlivých prvků webové stránky pro uživatele s postižením, kteří využívají asistenční nástroje. Například díky využívání různých úrovní nadpisů se nevidomí uživatel může snáze zorientovat v obsahu stránky a přeskočit na její určitou část, která ho zajímá.



Obrázek 4.1: Ukázka struktury nadpisů v seznamu prvků čtečky NVDA

4.2 Test přístupnosti webů veřejné správy

Na základě znalosti zákona z kapitoly 2.5, který ukládá povinnost zveřejnění informací v přístupné formě pro všechny uživatele na všech webech veřejné správy, byl proveden experiment přístupnosti. Týkal se několika webů veřejné správy a na základě výše analyzovaných znalostí byla k hodnocení využita jak analýza samotného HTML kódu stránky, tak srozumitelnost při využití NVDA čtečky a její možnost zobrazení seznamu prvků stránky. Na základě výsledků experimentu byly vybrány dva weby, Portál veřejné správy a Správa základních registrů, které budou detailněji popsány.

4.2.1 Portál veřejné správy

Tento web se nachází na adrese `https://seznam.gov.cz`. Jedná se o portál, který informuje o službách veřejné správy. Z tohoto důvodu byl také do testu vybrán, neboť skrze něj se uživatel může dostat na další weby a důležité informace veřejné správy.

Na první pohled vypadá web velmi dobře, poskytuje responzivní design

pro zařízení s menším displejem a vizuálně působí jednoduše a přehledně. Ovšem vizuální stránka není kritériem přístupného webu pro všechny uživatele.

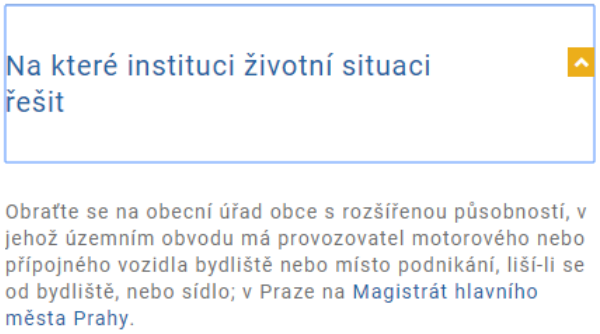
Pro zhodnocení webu z pohledu jeho přístupnosti je dobré nejprve vypsat seznam prvků stránky přes čtečku NVDA. Nadpisy jsou členěny logickým způsobem, to uživateli s čtečkou může pomoci při skocích mezi jednotlivými sekcemi. Z pohledu oblastí (landmarks) je web špatně implementován. Obsahuje pouze dvě oblasti, konkrétně je to hlavička (banner) a patička (contentinfo). Uživatel se čtečkou tedy nemůže provést skok rovnou na navigaci nebo další oblasti stránky. To zároveň znamená, že pokud se do těchto oblastí dostane pomocí procházení webu, čtečka nerozpozná, že se jedná o danou oblast. Což bylo ostatně vyzkoušeno, po dosažení prvního odkazu navigace čtečka informuje uživatele slovy: „List with four items, Občan, link“. O tom, že se jedná o hlavní navigaci webu nemá čtečka žádnou informaci. Stejná situace nastává při dosažení vstupního pole pro vyhledávání. Čtečka ho interpretuje následujícím způsobem: „Zajímá mě, edit“. Uživatel by mohl usoudit, že se možná jedná o vyhledávání, ale explicitní informaci o tom nemá.

Na dalším příkladu nepřístupné části webu je vidět rozdíl mezi vizuálně pochopitelným odkazem, který ovšem pro nevidomého uživatele pochopitelný není. Jedná se o odkazy na více informací z obrázku 4.2. Běžný uživatel si odkaz spojí s textovým obsahem nad ním a rozumí, k čemu odkaz patří. Nevidomí uživatel, jenž se po stránce pohybuje klávesnicí dojde na první odkaz a čtečka mu ho interpretuje následovně: „Více, link“. Tuto informaci dostane také u druhého odkazu, k čemu odkaz patří se ovšem nedozví. Propojení odkazu s jeho popisem by se mělo podle ARIA definovat pomocí atributu `aria-labelledby` [18].

Obrázek 4.2: Odkazy pro zobrazení více informací[1]

[1] `https://seznam.gov.cz/obcan/`

Úvodní stránka portálu další výraznější nedostatky z pohledu přístupnosti neobsahuje. Po přesunutí na stránku informující o Registru vozidel lze narazit na další těžko přístupný obsah pro uživatele se čtečkou obrazovky. Konkrétně se jedná o odkazy, které po kliknutí rozbalí svůj obsah viz obrázek 4.3. Modrým rámečkem je na obrázku zvýrazněn odkaz, na který lze kliknout pro zobrazení resp. schování obsahu pod ním. Vizuálně opět jasná funkcionalita. Při použití čtečky už to ovšem úplně jasné není. Při dosažení odkazu pro rozbalení obsahu ho čtečka interpretuje následujícím způsobem: „Na které instituci životní situaci řešit, heading level 3, link“. Žádná informace o tom, že odkaz slouží pouze pro rozbalení obsahu není v kódu uveden. Uživatel spoléhající pouze na čtečku obrazovky by tedy čekal, že po kliknutí na odkaz bude přesměrován na jinou stránku. Místo toho se ale nic nestane, obsah je rozbalen pouze přidáním CSS stylů na jeho HTML element. To čtečka nemůže zaznamenat a informovat jejího uživatele. Odkaz by měl disponovat ARIA atributem `aria-expanded` a případně by měl nést informaci, který obsah ovládá [19].



Obrázek 4.3: Odkazy s rozbalovacím obsahem[2]

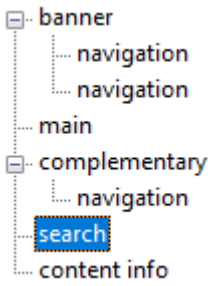
Přesto, že web působí z vizuálního pohledu velmi dobře, obsahuje několik úskalí, která uživatelům s asistenčními nástroji komplikují čitelnost webu.

4.2.2 Správa základních registrů

Druhý web se nachází na adrese `http://www.szrcr.cz/`. Je to web, který slouží pro informování občanů, orgánů veřejné moci a vývojářů o správě základních registrů, včetně poskytnutí informací k jejich přístupu. Jedná se o dobře přístupný web a do experimentu byl zahrnut, aby na něm bylo ukázáno několik dobrých postupů a technik, které dělají web přístupným.

[2] `https://seznam.gov.cz/obcan/zivotni-situace`

Analýzu webu byla opět zahájena vypsáním seznamu prvků přes NVDA čtečku. Web je velmi dobře rozdělen do oblastí, které jsou správně popsány pomocí ARIA rolí. Jak bylo již zmíněno, oblasti (landmarks) přináší uživatelům se čtečkou obrazovky možnost přeskakovat přímo do vybrané oblasti. Struktura oblastí je vidět na obrázku 4.4. Web obsahuje hlavičku (banner) s dvěma navigacemi (navigation), oblast s hlavním obsahem (main), boční panel s další navigací (complmentary), vyhledávání (search) a patičku s informacemi o webu (content info). Na totožné části je web rozdělen také z pohledu vizuální stránky. Nevidomý uživatel si tedy může představit rozdělení webu na stejné oblasti a pomocí klávesnice se v něm rychle pohybovat.



Obrázek 4.4: Rozdělení oblastí na webu správy základních registrů

Dalším přínosem pro uživatele využívající pouze klávesnici, mezi které patří i uživatelé se čtečkou obrazovky, je skrytá navigace na začátku webové stránky. Je to první dosažitelný prvek pomocí klávesnice. Pokud není aktivním prvkem stránky je vizuálně schovaná. Zobrazí se až poté, co získá jeden z jejích odkazů focus. Ukázka HTML navigačního odkazu je v kódu 4.1.

Kód 4.1: Odkaz skryté navigace

```
<a href="#content" class="hdn-nav" title="Prejit na obsah [
   klavesova zkratka Alt+S]" accesskey="s">
  Skoc na obsah
</a>
```

Odkazuje se na hlavní element s obsahem stránky. Pro detailnější popis používá atribut `title`. Tento popis se standardně zobrazuje po najetí myši na prvek a je předčítán pomocí čtečky. Posledním atributem je `accesskey`. Ten specifikuje klávesu, která v kombinaci s klávesou Alt slouží jako klávesová zkratka pro tento odkaz [20].

Tento web je dobře přístupný pro všechny uživatele a zároveň přináší techniku skryté navigace, které rozšiřuje usnadnění přístupu.

5 Techniky pro přístupnost webových aplikací

Tato kapitola se zabývá technikami, které jsou určeny pro docílení přístupnosti webových aplikací.

Dříve byl web místem pro prezentaci pouze statického obsahu. Změna nastala s příchodem Web 2.0. Tím je označována vývojová etapa internetu, kdy uživatel začíná s webem interagovat, podílí se na jeho obsahu či ovládá aplikace velmi podobné desktopovým programům. S příchodem této etapy se začalo používat spoustu technologií pro vývoj webových aplikací společně s JavaScriptem a výsledný kód začal být komplikovanější a pro uživatele s různým postižením těžko ovladatelný. To dokázalo konsorcium W3C zaznamenat a začalo vydávat již zmíněný standard WCAG.

V dnešní době se na webu stále více využívá dynamicky generovaného obsahu. To znamená, že je obsah pomocí JavaScriptu průběžně přidáván resp. odebírán a není třeba odesílat na server požadavek pro načtení celé stránky znovu. To je ovšem další problém pro jeho přístupnost. Takto generovaný obsah musí být dobře popsán, aby mu rozuměly asistenční nástroje a na jeho změny dokázaly reagovat.

K podpoře a udržitelnosti přístupnosti webu slouží několik dále zmíněných technik.

5.1 Struktura HTML5

Jednou ze základních technik pro vybudování přístupné webové aplikace je její správné navržení struktury oblastí. Důležitost těchto oblastí byla ověřena při experimentu s přístupností webů veřejné správy. K vytvoření oblastí slouží HTML5 sekční elementy a ARIA role z kategorie Landmark Roles (viy kapitola 3.2.1). Jak bylo zmíněno na obrázku 3.1, aktuálně nejnovější verze prohlížečů podporují sekční elementy i z pohledu přístupnosti. Starší verze prohlížečů a některé z čteček obrazovky stále vyžadují také označení pomocí ARIA rolí. Z čehož vyplývá, že sekční elementy je dobré používat, ale je nutné je vždy doplnit o ARIA role. Výjimkou může být použití některých sekčních elementů bez atributu role, význam oblasti se pak změní. Toto

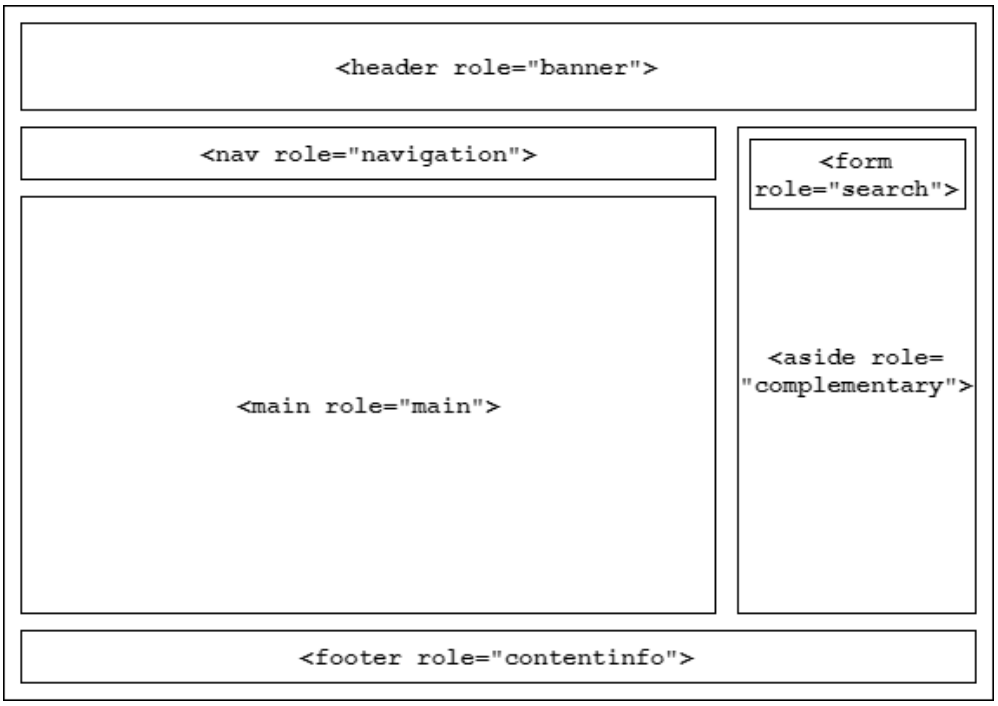
použití je objasněno v následujícím popisu šesti hlavních oblastí.

- `<header role="banner">` - jde o hlavičku stránky nebo sekce, specifikace ARIA definuje, že rolí `banner` by měla být označena pouze jedna oblast, a to hlavička celého webu [23].
- `<nav role="navigation">` - hlavní případně sekční navigace webu, v případě využití více navigací v rámci jedné stránky, navigace by měly být popsány alternativním vysvětlujícím textem.
- `<form role="search">` - oblast, která udržuje prvky sloužící pro vyhledávání, v případě více vyhledávacích polí v rámci jedné stránky, je třeba opět přidat detailnější popis, pro tuto oblast neexistuje HTML5 sekční element.
- `<main role="main">` - jde o hlavní oblast stránky nebo místo, kde beží hlavní část aplikace.
- `<aside role="complementary">` - většinou boční lišta s doplňujícími informacemi pro hlavní oblast stránky, může být ale využita i v rámci jiné sekce.
- `<footer role="contentinfo">` - patička stránky nebo sekce, která obsahuje informace o ní, ARIA definuje, že by rolí `contentinfo` měla být označena právě jedna hlavní patička webu [25].

Základní rozdělení oblastí webu je znázorněno na obrázku 5.1. Mezi další HTML5 elementy definující oblasti, které je dobré pro strukturu webu využít patří `<article>`, `<section>`, `<figure>` a `<h1>` až `<h6>`. Element `<article>` definuje nezávislou, většinou opakující se část webu, pro představu to může být položka výrobku na eshopu. Pro seskupení nějaké určité části obsahu se využívá element `<section>`, například je to sekce s komentáři článku. Element `<figure>` rozšiřuje textový obsah videem, obrázkem nebo ukázkou kódu. Je možné ho volitelně doplnit o popis pomocí elementu `<figcaption>`. Nadpisy jsou jediným způsobem tvorby strukturované osnovy webu. Jejich výhodu procházení webu pomocí čtečky obrazovky bylo již zmíněno.

Vzhledem ke složitosti struktury dnešních webových aplikací je dobré při jejich návrhu a implementaci dodržovat následující postup:

1. Navržení logické struktury - rozdělení webové aplikace do vnímatelných oblastí, které se pak ještě dále mohou strukturovat.



Obrázek 5.1: Rozdělení webové aplikace na základních šest oblastí

2. Přidání ARIA rolí jednotlivým oblastem - přiřazení rolí podle typu oblasti, role `banner`, `main`, `complementary` a `contentinfo` by měly být na nejvyšší úrovni struktury [21].

3. Alternativní popis jednotlivých oblastí - popisovat alternativním textem by se měly oblasti, které:

   (a) vyskytují se vícekrát v rámci webové aplikace,

   (b) začínají nadpisem (`<h1>` až `<h6>`), ten by měl být použit jako popis oblasti pomocí atributu `aria-labelledby`,

   (c) neobsahují žádný nadpis, který by je mohl popisovat, měly by být popsány pomocí atributu `aria-label` [21].

Během návrhu a implementace webových aplikací je dobré strukturu z pohledu přístupnosti testovat. Pro to existuje několik rozšíření prohlížečů a Google Chrome má ve svých DevTools záložku "Accessibility" pomocí které lze vypsat všechny dostupné informace, které jsou předávány čtečkám a dalším asistenčním nástrojům. Nejlepší a nejrychlejší je však testování pomocí samotné čtečky obrazovky.

5.2 Kontrast barev

Tato technika se netýká uživatelů využívajících čtečky obrazovky neboť jde o techniku zkoumající obsah webu z vizuálního hlediska. Zaměřuje se na uživatele se zhoršeným zrakem a také na běžné uživatele. Čím menší je kontrast barev v popředí a pozadí, tím složitější je obsah rozpoznat. Technika se týká jak psaného textu, tak obrázků s textem a jiného grafického obsahu.

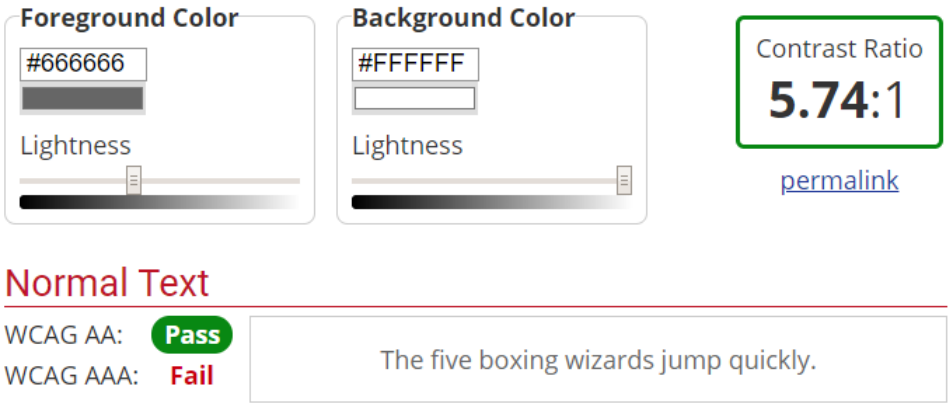
Technika se opírá o definici ze standardu WCAG 2.0. Ten definuje dvě úrovně pro kontrast barev (Contrast Ratio):

- Minimální poměr (Level AA) - text a obrázky obsahující text by měly splňovat kontrastní poměr 4,5:1, pro text velikosti alespoň 18px je to 3:1 [27].
- Doporučený poměr (Level AAA) - zde je základní poměr navýšen na 7:1, pro text velikosti alespoň 18px je to 4,5:1 [27].

Z definice pro dodržování poměru jsou vyjmuty texty a obrázky s textem, které jsou dekorativního charakteru nebo jsou součástí loga či jiné identity vydavatele webu [27].

Poměr kontrastu barev je během implementace webových aplikací složité počítat ručně. Samotný WCAG standard doporučuje několik nástrojů, které lze pro zjištění poměru dvou barev použít. Poměr se také dá analyzovat automatickými nástroji jako je například axe. Ten slouží k celkovému testování přístupnosti webu a je popsán v kapitole 8. Jedním z doporučených nástrojů pro kontrolu kontrastu je WebAIM Color Contrast Checker. Jeho použití je vidět na obrázku 5.2. Z obrázku je patrné, že kontrast zadaných barev splňuje pro standardní velikost textu pouze minimálně poměr definovaný podle WCAG (Level AA).

Z pohledu vývoje webových aplikací je dobré si všechny barvy navrhnout a analyzovat z pohledu kontrastu před jejich použitím. Tyto barvy je také dobré při stylování pomocí CSS definovat v proměnných, se kterými lze poté ovlivňovat jejich použití v rámci celé aplikace. To také umožňuje změnit barevné schéma aplikace v reálném čase, což může být použito například pro uživatelskou volbu, která změní web do hodně kontrastních barev. To většinou není z pohledu vizuální stránky aplikací primárně vyžadováno, ale některým uživatelům to může zpřístupnit cestu aplikace.



Obrázek 5.2: Analýza kontrastu dvou barev v nástroji WebAIM Color Contrast

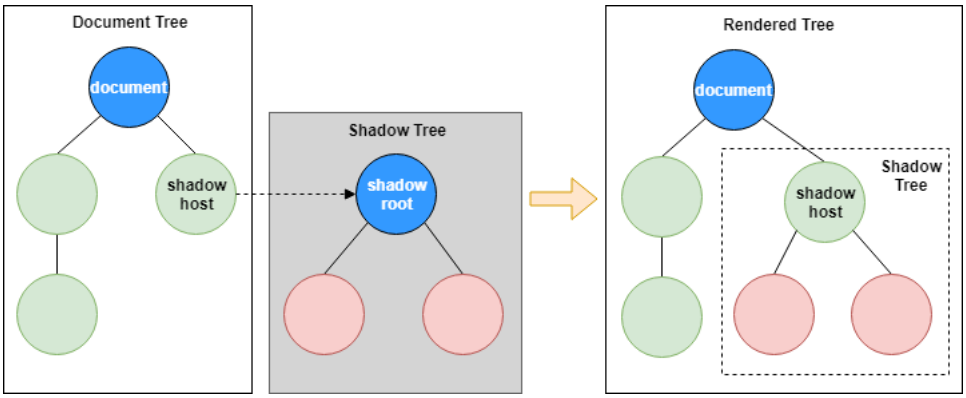
5.3 Shadow DOM

Problémem při vývoji webových aplikací je globálnost CSS a JS v rámci celého DOM (Document Object Model) aplikace. Pokud jsou vytvořeny CSS styly pro danou třídu, všechny prvky, které jsou třídou označeny budou aplikovat tyto styly. Shadow DOM je postup, jenž tento problém dokáže vyřešit.

Shadow DOM je jedním ze standardů pro tvorbu webových komponent (Web Component). Jeho hlavními myšlenkami a přínosy pro tvorbu webových komponent jsou:

- **Izolovaný DOM**: Komponenta má samostatně stojící DOM, to znamená, že například funkce `document.querySelector()` vrátí pouze vyhovující elementy nacházející se uvnitř Shadow DOM.
- **Uzavřené CSS**: Styly definované uvnitř Shadow DOM jsou uzavřené v rámci něj. Neovlivní tedy styl vnějších elementů a není ovlivnitelný styly z vnějšku.
- **Zjednodušené stylování**: Díky předchozí vlastnosti je možné používat jednoduché CSS selektory a obecnější názvy tříd a identifikátorů bez nebezpečí konfliktů.
- **Produktivita**: Napomůže pro vytvoření webové aplikace ze samostatně stojících nezávislých částí. Ty jsou také znovupoužitelné, čímž zamezíme redundantnosti kódu.

DOM obecně poskytuje strukturovanou reprezentaci vycházející z HTML, která se nazývá Document Tree. Shadow DOM je pak uzavřená část této struktury a je reprezentován pomocí Shadow Tree. Struktura obou zmíněných stromů je vidět na obrázku 5.3. Shadow DOM vzniká na jednom z elementů, jenž je nazýván Shadow Host. Z pohledu Shadow DOM je tento element jeho kořenem a nazývá se Shadow Root.



Obrázek 5.3: Struktura DOM při použití Shadow DOM

Shadow DOM se vytváří pomocí JS metody `element.attachShadow()`. Její parametr `mode` udává, zda jsou prvky Shadow Root přístupné zvenčí. Pro hodnotu `open` volání `element.shadowRoot` vrátí kořenový element. Pokud nastavíme hodnotu na `closed`, tak volání vrátí `null`, v tu chvíli se jedná o plně zapouzdřený Shadow DOM. Na příkladu kódu 5.1 je vidět vytvoření Shadow DOM na existujícím elementu DOM.

Kód 5.1: Vytvoření Shadom DOM

```
const header = document.createElement('header');
const shadowRoot = header.attachShadow({mode: 'open'});
const heading = document.createElement('H1');

document.body.appendChild(header);
heading.innerHTML = 'Nadpis 1';
shadowRoot.appendChild(heading);
```

Ze zmíněných přínosů Shadow DOM plynou vcelku jasné výhody pro implementaci přístupných webových aplikací. Vzhledem k jeho zapouzdření je možno definovat ARIA atributy a další chování jako například klávesové zkratky v rámci nějakého menšího celku bez rizika ovlivnění zbytku aplikace. Zapouzdření Shadow DOM může přinést z pohledu přístupnosti také určité nevýhody. Ty jsou zmíněny po objasnění druhého standardu pro tvorbu

webových komponent v následující kapitole.

5.4 Custom Elements

Pomocí Custom Elements mohou weboví vývojáři vytvářet vlastní HTML elementy, upravovat existující elementy nebo rozšiřovat komponenty, které vytvořili jiní vývojáři. Společně s Shadow DOM přináší standard pro vytváření znovupoužitelných webových komponent (Web Components) pomocí HTML, CSS a JS. Výhodou webových aplikací využívajících tyto komponenty je menší množství kódu, modulární architektura a větší přehlednost. Z toho také plyne rychlejší vývoj a přínos pro implementaci přístupnosti.

Usnadnění přístupu pro uživatele s různými postiženími se implementuje snáze pro jednotlivé komponenty, které představují daný ovládací nebo vizuální prvek. Zároveň se pro to dá využít dokument Authoring Practises popsaný v kapitole 3.5. Všechny samostatně stojící komponenty jako například tlačítko, tabulka nebo stromová struktura jsou v něm detailně popsány včetně rolí, atributů a klávesové ovladatelnosti včetně zkratek, kterými by měla výsledná komponenta disponovat.

Implementace vlastního HTML elementu pro reprezentaci webové komponenty je vidět na výpisu kódu 5.2.

Kód 5.2: Vytvoření vlastního HTML elementu

```
class OwnInput extends HTMLElement {...}
window.customElements.define('own-input', OwnInput);

// Priklad pouziti
<own-input></own-input>
```

Třída definující nově vzniklý HTML element má konstruktor, ve kterém se inicializuje její výchozí stav. Je to také místo, kde se vytváří Shadow DOM. Konstruktor může mít například podobu podle ukázky v kódu 5.3.

Kód 5.3: Konstruktor třídy definující HTML element

```
constructor() {
  super(); // volani konstruktoru tridy HTMLElement
  this.attachShadow({ mode: 'open' });
  this.input = document.createElement('input');
  this.input.id = 'internal-input';
  this.input.type = 'text';
```

```
}
```

Z kódu lze vidět, že kořenový element nově vznikající komponenty je její Shadow Root. Tím se implementuje zapouzdření komponenty a vytvoří se tedy vlastní element s textovým polem. Z pohledu přístupnosti však textovému poli chybí popis. Ten se zpravidla pro textové pole definuje pomocí elementu `<label>` (viz kód 5.4).

Kód 5.4: Popis elementu pomocí label

```
<label for="text">Popis</label>
<own-input id="text"></own-input>
```

To ale díky Shadow DOM nemůže fungovat. Element `<own-input>` je Shadow Root a není možno na něj ani na celý jeho Shadow Tree aplikovat standardní atributy. Což se v tomto případě může zdát jako nevýhoda. Z pohledu Shadow DOM a komponentového přístupu vývoje je to ale výhoda, protože nikdo nemůžu ovlivnit komponenty pomocí atributů z venčí. Custom Elements navíc poskytují možnost definovat vlastní atributy.

Jelikož se v kódu výše jedná o komponentu definující nějakým způsobem rozšířené textové pole, mělo by docházet k jejímu popisu uvnitř. Custom Elements poskytují lifecycle metody (metody pro obsluhu životního cyklu), díky kterým je možno zachytit definici atributu komponenty a nějakým způsobem na ni reagovat. Ukázka použití metody je vidět na zkráceném výpisu kódu 5.5.

Kód 5.5: Metoda pro zachycení definice atributu

```
class OwnInput extends HTMLElement  {
  static get observedAttributes() {
    return ['label'];
  }

  constructor() {...}

  attributeChangedCallback(attr, oldVal, newVal) {
    if (attr === 'label' && newVal) {
      this.input.setAttribute('aria-label', newVal);
    }
  }
}
window.customElements.define('own-input', OwnInput);
```

```
// Priklad pouziti
<own-input label="Popis"></own-input>
```

Díky vlastní implementaci popisu textového pole uvnitř komponenty je dodržena správná implementace přístupnosti při jejím znovupoužití. Kromě popisu můžou být uvnitř také definované například klávesové zkratky a další funkcionalita.

Celkovým přínosem Custom Elements a Shadow DOM pro přístupnost je tedy aparát, který jí umožňuje implementovat na jednom uzavřeném místě s možností definování atributů, které výsledný element z pohledu přístupnosti popisují. Je to celkově aktuální a moderní přístup pro tvorbu webových aplikací, hlavně z pohledu modularity vývoje a přehlednosti výsledného kódu. Nevýhodou je absence podpory v některých prohlížečích. Chrome má podporu od verze 53, Firefox od verze 63, ale prohlížeč Microsoft Edge Shadow DOM vůbec nepodporuje. Východiskem je využití některého z JS frameworků, které komponentový přístup implementují vlastním způsobem a udržují tak podporu skrze všechny prohlížeče za nás.

5.5 Inert polyfill

Atribut `inert` je pro zatím abstraktní HTML atribut, který by měl být v budoucnu součástí HTML5 [29]. Komunita WHATWG, jež se zabývá udržováním a rozvíjením HTML standardu definuje, že element označený pomocí `inert` nesmí být aktivní pro interakci s uživatelem, a to včetně všech jeho vnořených elementů. Interakcí s elementem je také myšleno označování textu a standard říká, že se může týkat také vyhledávání textu ve stránce, inertní element v něm nemusí být zohledňován [29].

Z technického hlediska JavaScriptu je dobré zmínit a okomentovat příklad kódu 5.6. Na něm lze snadno pochopit základní myšlenka `inert` atributu.

Kód 5.6: Ukázka použití atributu inert

```
<body>
  <div>
    <button>Dosazitelne tlacitko</button>
  </div>
  <div inert>
    <p>Paragraf textu</p>
    <button>Inert tlacitko</button>
  </div>
```

```
</body>
```

Na ukázce je jednoduchá webová stránka, která obsahuje dva elementy `<div>`. Jeden z nich je označen atributem `inert`. Pokud uživatel bude používat k pohybu myš a bude se pohybovat nebo bude chtít kliknout na libovolný element inertního bloku, Javascript na něm nevyvolá událost `mouseover`, která indikuje posunutí kurzoru myši na element. Při kliknutí na element inertního bloku na něm nebude vyvolána událost `click`, tato událost nastane na elementu `<body>`. Z toho vyplývá, že inertní element a všechny jeho vnořené elementy nejsou zohledňovány ani v událostech DOM, které JavaScript obsluhuje. Při pohybu pomocí klávesnice nebude na tlačítko uvnitř inertního elementu možno přesunout focus.

Reálným příkladem využití může být například modální dialogové okno. Na několika webech lze zjistit, že pokud se dialogové okno otevře, je možnost pomocí klávesy Tab posunout focus mimo okno zpět do stránky. Pokud to učiní uživatel se čtečkou obrazovky, je to pro něj velmi matoucí, myslí si, že dialogové okno je již zavřené a pohybuje se po stránce. Vizuálně je však dialogové okno v popředí a čeká na interakci uživatele. Tento problém lze vyřešit i jinými způsoby než atributem `inert`, jako třeba zachycení a vlastní obsluha klávesy Tab. Pomocí označení elementů za inertní je ale řešení nejsnazší, přináší i další zmiňované vlastnosti a v budoucnu bude standardním. Zjednodušený kód HTML stránky při otevřeném dialogovém okně je uveden v kódu 5.7.

Kód 5.7: Příklad HTML stránky s otevřeným dialogovým oknem

```
<body>
  <header role="banner" inert>...</header>
  <nav role="navigation" inert>...</nav>
  <main role="main" inert>...</main>
  <footer role="contentinfo" inert>...</footer>
  <div role="dialog">...</div>
</body>
```

Dialogové okno a další modální obsah je dobré přidávat na konec těla stránky. Je to obsah, který se umisťuje pomocí absolutní pozice a nesouvisí se strukturou stránky. Navíc je pak možnost ostatní kořenové elementy označit za inertní a obsah k interakci uživatele zůstává pouze na dialogu. Na příkladu se také potvrzuje přínos využívání sekčních elementů. Označením celé sekce za inertní se všechen její obsah stane pro uživatele nepřístupným. Dalším

podobným příkladem může být navigace, která se pomocí tlačítka skrývá mimo obrazovku. Při skrytém stavu ji stačí označit za inertní a uživatel se pomocí klávesnice nedostane na její položky.

Vzhledem k tomu, že atribut `inert` není zatím v HTML standardně podporován a je označován za abstraktní, pro jeho aplikování se musí využít polyfill. V JavaScriptu je polyfill poměrně často používaný mechanismus. Slouží jako automatické alternativní řešení, které dokáže zajistit funkčnost něčeho, co buď není podporováno nebo podpora chybí v některém z prohlížečů. Jednoduše to lze ukázat na příkladu kódu 5.8.

Kód 5.8: Ukázka jednoduchého polyfill

```
// Polyfill pro funkci
if (!window.log) {
  window.log = msg => console.log(msg);
}

// Volani funkce
log('Logovaci zprava');
```

JavaScript standardně neobsahuje funkci s názvem `log()`. Vzhledem k tomu, že je to dynamický jazyk, umožňuje definovat funkce za běhu. Je tedy možné funkci `log()` definovat a v rámci ní zavolat standardní logovací funkci.

Pro atribut `inert` existuje několik polyfill skriptů, které dočasně zaručují jeho funkčnost. Dočasně je myšleno do doby, kdy bude atribut standardní součástí HTML a bude podporován ve všech prohlížečích. Na GitHub lze najít dva nejznámější, jeden je zahrnut přímo v projektu Google Chrome. Druhý vytvořila skupina WICG (Web Incubator Community Group), která se zabývá implementací a testováním nově vznikajících funkcí pro webovou platformu.

Při testování implementace atributu `inert` lze narazit ve spojitosti s přístupností pomocí čtečky obrazovky na jeden problém. Konkrétně u čtečky NVDA se inertní elementy neskrývají v seznamu prvků, na které může uživatel provést skok. Vzhledem k tomuto faktu by se měl inertní element skrývat také pomocí ARIA atributu `aria-hidden`.

6 Technologie pro vývoj

V této kapitole budou popsány technologie, které byly využity pro vývoj přístupné webové aplikace. Nejdříve to bude technologie Node.js, jež byla využita pro implementaci serverové části aplikace, konkrétně pro frameworku Express. Poté bude objasněn termín dynamických webových aplikací a popsán framework Angular, který byl zvolen pro klientskou část aplikace.

6.1 Node.js

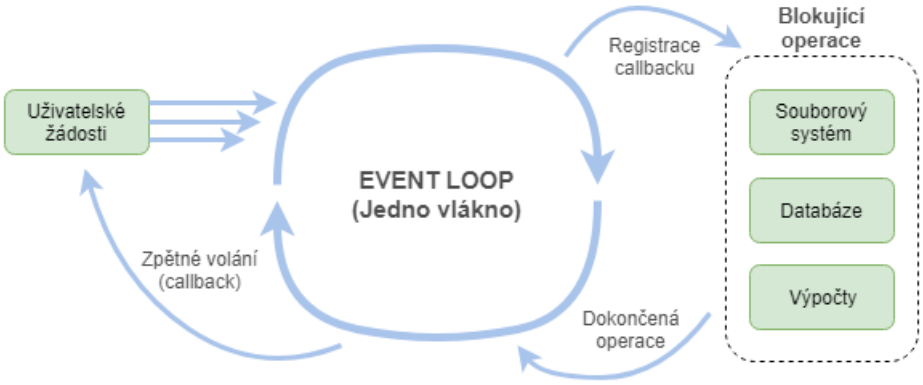
Node.js je platforma umožňující vykonávání JavaScript kódu mimo webový prohlížeč, primárně určena pro tvorbu serverové části webových aplikací (backend) [26]. Je postavena na Google V8 JavaScript enginu, který využívá také prohlížeč Google Chrome. Vzhledem k tomu, že je Google Chrome nejpoužívanějším prohlížečem, V8 engine je vysoce optimalizovaný z pohledu výkonu, z čehož těží také Node.js [26].

Aplikace na Node.js běží v jednom procesu bez tvorby nových vláken pro každý požadavek. Node.js poskytuje sadu neblokujících asynchronních vstupně výstupních (I/O) operací, díky nimž se hodí pro datově náročné realtime aplikace [26]. Typická asynchronní událost může být dotaz do databáze. Node.js nečeká na výsledek, ale operaci odloží do fronty čekajících, ve které požadavek čeká až obdrží data. Poté je naplánován jeho návrat do hlavního vlákna, ve kterém je dále zpracován.

6.1.1 Architektura platformy

Uvedený příklad s asynchronním zpracováním dotazu do databáze je tvořen tzv. smyčkou událostí (Event loop), která je specifická pro architekturu Node.js.

Smyčka událostí běží v jednom vlákně a vstupují do ní všechny uživatelské požadavky. Její běh je synchronní a jakmile narazí na události, jako jsou přístup do databáze nebo souboru na disku, místo blokování vlákna a čekání na jejich vykonání operace odloží do fronty čekajících a zaregistruje na ně callback, díky kterému se k operacím vrátí po jejich dokončení. Na obrázku 6.1 je zmíněné zpracování uživatelských požadavků naznačeno.



Obrázek 6.1: Node.js smyčka událostí (Event loop)

6.1.2 Node Package Manager

Pro vývoj na platformě Node.js se využívá balíčkovací systém NPM (Node Package Manager). Umožňuje instalaci balíčků (packages) a rozhraní pro práci s nimi. Díky NPM je možno balíčky aplikace spravovat a aktualizovat. Node.js také umožňuje automatické stažení závislostí při instalaci určitého balíčku. Veškeré informace o stavu a potřebných balíčcích aplikace se uchovávají v souboru `packages.json`. Po nainstalování je balíček na disku adresář, který obsahuje veškerý potřebný kód a metadata.

NPM byl využit pro správu balíčků během celého vývoje diplomové práce resp. vzniklé webové aplikace. Byl využit jak pro klientskou Angular aplikaci, tak pro serverovou část psanou v Express.

6.2 Dynamické webové aplikace

Jak bylo již zmíněno, implementace přístupnosti pro usnadnění přístupu na web se týká veškerého webového obsahu. Pro statické webové stránky je implementace přístupnosti snazší. Dopředu je známo jak obsah vypadá a je tvořen většinou pouze pomocí HTML. Tato diplomová práce se z pohledu implementace týká pouze webových aplikací, ve kterých se obsah generuje dynamicky. Pro takové aplikace se používají různé JavaScript frameworky, které vývoj usnadňují a hlavně zaručují kompatibilitu napříč webovými prohlížeči.

Dříve byl pro generování dynamického obsahu používán pouze AJAX (Asynchronous JavaScript and XML). Ten poskytuje možnost asynchronní komunikaci se serverem pomocí JavaScriptu resp. jQuery a `XMLHttpRequest`

požadavků [22]. Díky asynchronní komunikaci se může obsah ze serveru zpracovat pomocí JavaScriptu a dynamicky bez nutnosti aktualizace celé webové stránky přidat do HTML.

V dnešní době však existuje standardizovaná možnost komunikace se serverem přímo v JavaScriptu. Od verze ES6 je k dispozici Fetch API poskytující rozhraní pro komunikaci se serverem pomocí HTTP (Hypertext Transfer Protocol) požadavků [24]. Konkrétně k tomu slouží funkce `fetch()`, díky které je možno dotazovat se serveru a po získání odpovědi dynamicky generovat obsah stránky. V tu chvíli je ale kód závislý na podpoře Fetch API v dané verzi prohlížeče, která například v prvních verzích Microsoft Edge chyběla. Existuje ale množství knihoven, které pomocí polyfill umožňují HTTP komunikaci se serverem bez ohledu na používaný prohlížeč. Tyto knihovny jsou také součástí hojně využívaných frameworků pro SPA (Single Page Application), kterým se věnuje následující kapitola.

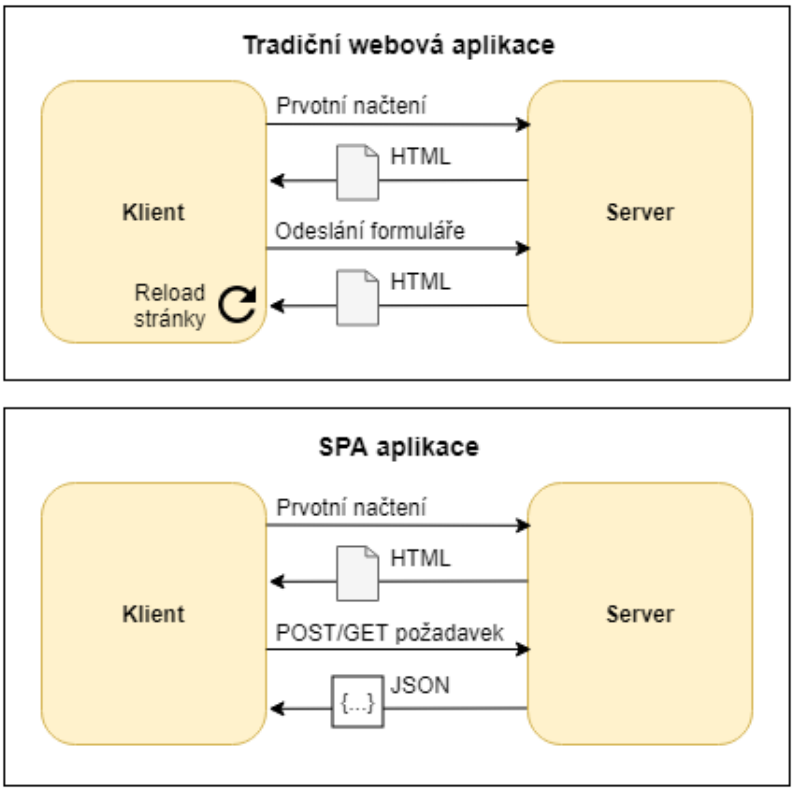
6.3 Single Page Application

SPA jsou webové aplikace, které načtou jednu HTML stránku při prvotním načtení a následně se vlastní obsah generuje dynamicky v prohlížeči pomocí JavaScriptu, server pak slouží pouze jako zdroj dat [28]. Při prvním načtení SPA aplikace se tedy:

1. načte stránka `index.html` a zpracuje se v prohlížeči,
2. podle ní se stáhnou potřebné JS skripty,
3. na základě skriptů se asynchronně stáhnou všechny ostatní potřebné zdroje,
4. načte se aplikace prvotní stav aplikace.

Rozdílem oproti tradičním webovým aplikacím je tedy průběžná komunikace se serverem v průběhu interakce s danou webovou aplikací. Tradiční aplikace každý požadavek uživatele odesílají na server a zpět dostanou vygenerovanou stránku v podobě kompletního HTML kódu. SPA si načte HTML šablonu a do této šablony dynamicky vkládá požadovaná data bez nutnosti opětovného načtení celé HTML stránky. Rozdíl komunikace tradiční webové aplikace a SPA je vidět na obrázku 6.2.

U SPA si prohlížeč všechny skripty a zdroje potřebné pro běh aplikace uloží do cache, a pak už celá aplikace běží z ní. To přesouvá režii běhu



Obrázek 6.2: Komunikace klient-server u tradičních a SPA aplikací

aplikace na klienta a server jen zpracovává požadavky na data. Při dalším navštívení je aplikace načtena z cache, což prvotní načtení urychluje. SPA aplikace komunikují se serverem většinou pomocí REST API (Representational State Transfer API). To zpracovává HTTP požadavky klienta a odesílá požadovanou odpověď.

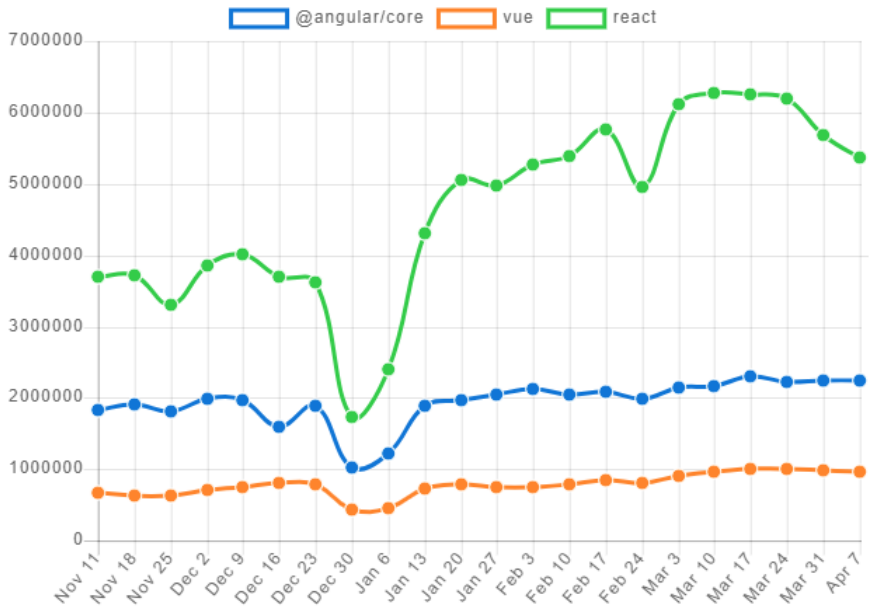
SPA mají oproti tradičním webovým aplikacím několik výhod:

- díky absenci aktualizace stránky po každém požadavku jsou rychlejší, načítají menší objem dat a působí plynuleji,
- jsou jednodušší pro vývoj, protože vyvíjíme oddělenou klientskou část, většinou k již hotové serverové části,
- pokud jsou dobře navrženy, tak se u nich snáze provádí debugování, serverovou část je navíc možné jednoduše nasimulovat,
- je možno implementovat postupy pro vytvoření PWA (Progressive Web Apps) aplikací, které se v dnešní době hodně rozvíjejí.

Mezi rizika patří problémy bezpečnosti webových aplikací např. V podobě častého XSS (Cross Site Scripting) útoku. Jedná se o útok pomocí vložení a vykonání části skriptu ve stránce. Většina frameworků umí takový útok odchytit, ovšem například vložení skriptu přes vývojářskou konzoly prohlížeče lze jen těžko zastavit. Další, velmi důležitou nevýhodou pro tuto práci, je právě přístupnost. Vzhledem k tomu, že se obsah generuje dynamicky bez opětovného načtení celé stránky, asistenční nástroje na takto vzniklý obsah těžko reagují. Zároveň se v SPA aplikacích po zpracování požadavku neobnovuje pozice aktivního prvku, takže se o ni musí starat sám vývojář. Právě z těchto důvodů byla SPA aplikace zvolena pro implementaci klientské části této práce.

6.4 Angular

Framework Angular je JavaScript frontend framework pro tvorbu webových aplikací napříč všemi platformami. Podle počtu stažení z NPM patří do trojice nejvíce využívaných frameworků z této oblasti (viz obrázek 6.3).



Obrázek 6.3: Podíl SPA frameworků podle počtu stažení z NPM

První verze Angularu byla vydána v roce 2010 pod názvem AngularJS, což z něj dělá nejstaršího zástupce těchto frameworků. Je vyvíjen společností Google. V roce 2016 zaznamenal velký posun vydáním Angular 2 (verze Angular 2+ jsou dnes používaný pouze pod názvem Angular). Od verze Angu-

lar 2 se pro vývoj klientských aplikací využívá HTML a TypeScript, což je programovací jazyk vytvořený a spravovaný společností Microsoft. Jedná se o nadstavbu jazyka JavaScript, konkrétně přináší rozšíření o statické typování a další vlastnosti známé z objektově orientovaného programování. To z Angularu dělá efektivní platformu pro velké týmy vývojářů. Aktuální verze je Angular 7, která byla vydána v říjnu 2018 a je použita pro implementaci webové aplikace v rámci této práce.

Samotný Angular je také implementován pomocí TypeScript. Obsahuje jádro frameworku a další sadu volitelných funkcí, které se dají importovat jako knihovny do aplikace. Zároveň umožňuje využití komponentového přístupu včetně zapouzdření komponenty a jejího kaskádového stylu. Architektura se dá rozdělit na následující tři základní stavební bloky.

6.4.1 Modules

Základním stavebním kamenem Angular aplikací jsou moduly (Modules), které slouží jako kompilační kontext pro komponenty (Components). Kompilačním kontextem je myšleno vše potřebné pro správné zkompilování a funkčnost výsledných komponent (soubory, vnořené komponenty atd.) [30]. Moduly tedy sdružují potřebný kód do funkčních celků.

Každá Angular aplikace má tzv. root modul, podle konvencí pojmenovaný `AppModule`, který slouží jako zaváděcí mechanismus všech potřebných částí včetně ostatních modulů [30]. Z toho vyplývá, že jeden modul může importovat funkcionalitu z jiných modulů a zároveň může část funkcionality poskytovat jiným modulům. Příkladem může být implementace modulu pro routování, který typicky využívá funkcí již vytvořené Angular modulu `Router`.

Pro představu jak fungují moduly je na kódu 6.1 zkrácená deklarace `AppModule`. Dekorátor `@NgModule` definuje metadata, ty určují z jakých částí se modul skládá a jakou funkcionalitu z jiných modulů využívá. Jako první se v `declarations` definují části, které modulu patří. Konkrétně u root modulu jsou to všechny komponenty a další části aplikace, jež modul zavádí do prohlížeče. V `imports` jsou pak moduly, které využívá pro svou funkčnost. Poslední část je specifická pro root modul, `bootstrap` definuje hlavní komponentu celé aplikace.

Kód 6.1: Vytvoření root modulu

```
// @NgModule decorator s~metadaty
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule {...}
```

Výhodami modulové architektury je rozdělení funkcionality do menších, znovupoužitelných celků. To pomáhá při řízení vývoje komplexnějších aplikací. Technika také přináší výhodu lazy-loading, moduly jsou načítány až při dosažení části aplikace, která je vyžaduje.

6.4.2 Components

Angular jako framework přináší vlastní syntaxi pro tvorbu komponent, ale vnitřně se jedná o definování komponent, které bylo zmíněno v kapitole 5.4. Samotné definování komponent usnadňuje, zaručuje kompatibilitu skrze všechny prohlížeče a rozšiřuje ji o vlastní rozhraní pro vstupní a výstupní vlastnosti komponenty.

Každá Angular aplikace má minimálně jednu komponentu, konvenčně pojmenovanou `AppComponent` [31]. Ta vytváří základní hierarchii ostatních komponent a jejich propojení s DOM. Každá komponenta je definována pomocí třídy, která obsahuje data a aplikační logiku, a je asociována s HTML šablonou (template), jež definuje výsledný vzhled komponenty v prohlížeči. Třída vytvářející komponentu se označuje dekorátorem `@Component()`, pomocí něj se třída definuje. Mezi základní metadata dekorátoru komponenty patří:

- `selector` - identifikuje název elementu, pomocí kterého se komponenta vytváří v HTML
- `template` - inline šablona komponenty, zadává se jako standardní JS řetězec obsahující HTML
- `templateUrl` - relativní nebo absolutní URL cesta k HTML souboru, který obsahuje šablonu komponenty, nahrazuje tedy iniline šablonu

`template`

- `styles` - jeden nebo více inline CSS stylů komponenty, zadává se jako JS řetězec
- `styleUrls` - jedna nebo více relativních nebo absolutních URL cest k CSS souborům definujícím styl komponenty
- `inputs, outputs` - slouží pro definování sady vstupních proměnných a sady výstupů vázaných na události vyvolané komponentou, jako například `ClickEvent`

Ukázka takto definované Angular komponenty je na výpisu kódu 6.2.

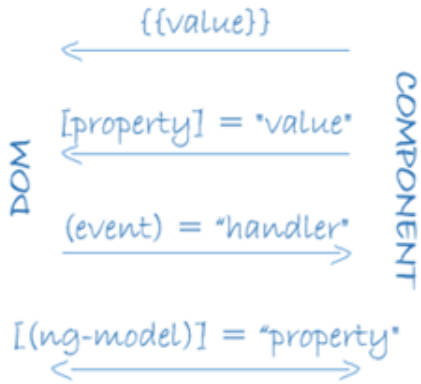
Kód 6.2: Vytvoření komponenty

```
@Component({
  selector: 'menu-item',
  template: `<a (click)="linkClick()">{{linkText}}</a>`,
  styleUrls: ['./menu-item.component.css'],
  inputs: [linkText],
  outputs: ['clickEvent']
})
export class MenuItemComponent {
  linkText: string;
  clickEvent: EventEmitter = new EventEmitter();

  linkClick() {
    this.clickEvent.emit();
  }
}

// deklarace komponenty
<menu-item (click)="ownClickHandler"></menu-item>
```

Šablona (template) kombinuje HTML s Angular značením, které mimo jiné umožňuje definovat rozhraní pro komunikaci DOM šablony s třídou komponenty. Využívá se pro to tzv. data binding, jenž se dělí na dva typy:

- Event binding - umožňuje aplikaci resp. komponentě reagovat na interakci uživatele pomocí událostí.
- Property binding - umožňuje převést data definovaná v komponentě do šablony.

Data binding slouží mimo jiné i pro komunikaci mezi rodičovskou a vnořenou komponentou. Využívá čtyři druhy značení (viz obrázek 6.4), které určují jeho typ. První dva typy umožňují transfer hodnot z komponenty do DOM šablony. Třetí slouží pro navázání metod komponenty na uživatelem vyvolané události. Posledním je tzv. two-way databinding. To slouží jako obousměrná kombinace předchozích. Z komponenty do DOM je možno navázat hodnotu proměnné a z DOM jí ukládat při vzniku uživatelské události. Jednoduché využití si lze představit na formuláři pro úpravu nějakého objektu, již existující hodnota je do něj načtena a při změně je uložena zpět do proměnné v komponentě.



Obrázek 6.4: Značení a typy Angular Data Binding [31]

6.4.3 Services

Pro data a aplikační logiku, která není přímo spojená s některou z komponent a je potřeba ji poskytovat napříč všemi komponentami, se vytváří servisní třída (Service). Definuje se dekorátorem `@Injectable()`. Jak už z názvu vyplývá, servisy mohou být pomocí Dependency injection přidány do libovolné třídy. Dependency injection je návrhový vzor, při kterém třída místo vytvoření vlastní instance nějaké třídy (dependency), získává již vytvořenou instanci z nějakého úložiště. Angular poskytuje jeho vlastní implementaci DI. To udržuje aplikace přehledné, zvyšuje to jejich účinnost a modularitu [32].

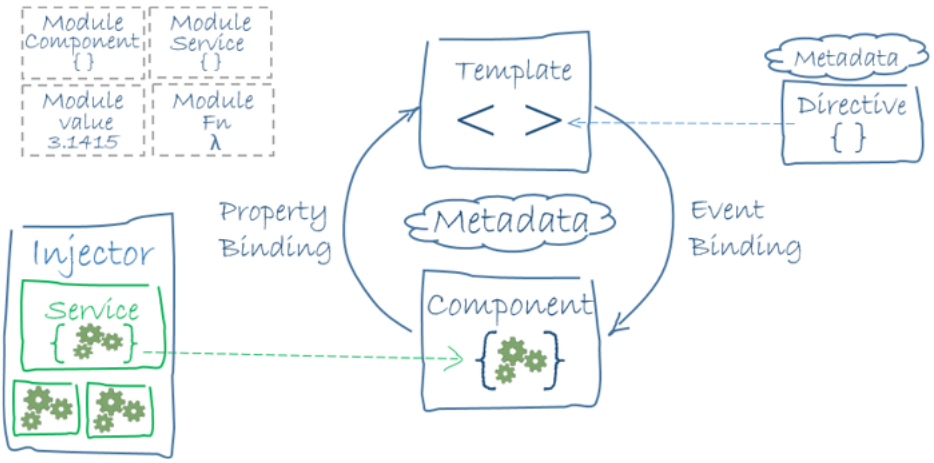
Typické využití servisních tříd je HTTP komunikace se serverem, jednotná validace dat nebo logování. Zjednodušený příklad implementace servisní třídy, včetně DI do komponenty je na výpisu kódu 6.3.

Kód 6.3: Vytvoření servisní třídy

```
@Injectable()
export class ErrorLoggerService {
  log(msg: any) { console.log(msg); }
}

@Component({...})
export class ExampleComponent {
  constructor(private errorLogger: ErrorLoggerService) {}

  exampleMethod(): void {
    this.errorLogger.log('Log chyby do konzole');
  }
}
```

6.4.4 Architektura frameworku

Byly zmíněny základní prvky architektury frameworku Angular, zbývá vysvětlit jejich vzájemnou souvislost v aplikaci jako celku. Obrázek 6.5 znázorňuje, jak spolu jednotlivé prvky souvisejí. Komponenta spolu s šablonou definují vzhled aplikace. Využívají property a event binding. Directive definuje rozšíření chování pro jednotlivé prvky DOM. Příkladem je například directive `*ngIf`, pomocí něhož se definuje podmínka, na základě které je prvek přidán resp. odebrán z DOM. Injector slouží jako mechanismus pro Dependency injection servisních tříd. Moduly definují z jakých částí se celá aplikace může skládat.



Obrázek 6.5: Architektura frameworku Angular [30]

6.5 Express framework

Express je populární Node.js framework pro tvorbu webových aplikací a API. V rámci diplomové práce byl využit pro tvorbu serverové části aplikace.

Je to jednoduchý a minimalistický framework, který obsahuje jádro pro tvorbu serverové části aplikace, které se rozšiřuje přístupnými NPM balíčky. Balíčky se pak využívají jako tzv. middleware. Middleware jsou funkce, které mají přístup k objektu s HTTP požadavkem (request) a jeho odpovědi (response). To se využívá například pro obsluhu session, autentizaci uživatelů, parsování URL parametrů, hlavičky HTTP requestů a další funkcionalitu. Kromě toho middleware může:

- vykonávat libovolný kód,
- provádět změny v objektech `request` a `response`,
- zastavit nebo přerušit request-response cyklus,
- volat následující middleware funkci.

Objekty `request` a `response` procházejí middleware funkcemi v pořadí, ve kterém jsou volány. Přidání takového middleware do aplikace se provádí funkcemi `app.use()` a `app.METHOD()`, kde `METHOD` je jedna z podporovaných HTTP metod. To že vytvoření API je v Express jednoduché dokazuje i kód 6.4.

Kód 6.4: Ukázka implementace jednoduchého API

```
const express = require('express');
const app = express();
const port = 3000;

app.use((req, res, next) => {
  console.log('Time:', Date.now());
  next();
});

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(port, () => console.log(`Ukazkove API na portu
    ${port}!`));
```

Jedná se o kompletní kód, který vytvoří REST API dostupné na portu 3000, při každém požadavku loguje middleware funkce do konzole text a při `GET` požadavku API odešle textovou odpověď. Samozřejmě se jedná o ukázku se zpracováním pouze jednoho požadavku, větší API vyžadují určitou strukturu kódu, která je zmíněna v implementační části práce.

7 Demonstrační webová aplikace

Tato kapitola se věnuje návrhu funkcionality a implementaci aplikace. Aplikace byla vytvářena primárně za účelem prezentace technik pro přístupnost webových aplikací. Podle toho byly také vybrány její ovládací prvky a komponenty. Samotná implementace byla rozdělena na klientskou a serverovou část aplikace.

7.1 Návrh aplikace

Vzhledem k tomu, že primárním účelem webové aplikace, která měla vzniknout v rámci diplomové práce byla demonstrace přístupnosti, speciálně pro uživatele se zrakovým postižením, její návrh se odvíjel od ovládacích prvků a komponent, které jsou pro zpřístupnění webu něčím specifické a dají se na nich prezentovat techniky implementace přístupnosti.

Nejdříve došlo k navržení ovládacích prvků a komponent, poté k navržení uživatelské rozhraní, které je využívá. Nakonec bylo určeno, co bude aplikace spravovat a podle toho proběhl její návrh aplikačního rozhraní pro správu dat.

7.1.1 Ovládací prvky a komponenty

Pomocí ARIA a JavaScript lze vytvořit jakýkoliv ovládací prvek. Není ovšem žádoucí tímto způsobem vytvářet standardní HTML ovládací prvky, které přinášejí veškerou funkcionalitu optimalizovanou skrze všechny prohlížeče. Jsou ale ovládací prvky, které v HTML nativní podporu nemají, případně se skládají z několik standardních prvků komunikujících mezi sebou.

Některé komponenty byly zvoleny na základě dokumentu ARIA Authoring Practises, který obsahuje popis implementace nestandardních ovládacích prvků [17]. Další komponenty byly vybrány na základě problematických oblastí přístupnosti jako je obsah mimo obrazovku nebo skrývatelné části webu. V aplikaci byly využity následující komponenty.

- **Formuláře s notifikacemi**: Je důležité poskytovat zpětnou vazbu uživateli o stavu formuláře po jeho odeslání. V SPA aplikacích je tento stav přidáván pomocí JavaScriptu a je nutné upozornit asistenční nástroje o jeho zobrazení na stránce.

- **Skrývatelná navigace**: Vzhledem k velkému počtu uživatelů přistupujících na web pomocí mobilních zařízení, se v dnešní době hojně využívají skrývatelné navigace. Navigace je schována mimo obrazovku a zobrazuje se většinou pomocí tlačítka. Takové navigace vyžadují několik postupů pro jejich zpřístupnění.

- **Menu vyvolané tlačítkem**: Jedná se tlačítko, které po kliknutí otevře menu. Většinou se pro to využívá standardní HTML tlačítko, na kterém je graficky znázorněno, že slouží k otevření nabídky. Pro nevizuální uživatele webu je třeba tlačítka alternativně popsat.

- **Toolbar s textovým editorem**: Toolbar (panel nástrojů) je kontejner pro seskupení sady ovládacích prvků jako jsou tlačítka, zaškrtávací pole a další. V rámci webové aplikace této práce budou panely nástrojů využity pro textový editor. Ten je pro demonstraci přístupnosti také zajímavý a v kombinaci s panely nástrojů na něm lze implementovat ovládání pomocí vlastních klávesových zkratek. Pro toolbar se také často využívá vlastní obsluha aktivního prvku při pohybu klávesnicí.

- **Modální dialog**: Dialog je okno překrývající primární obsah. Většinou se využívá jako potvrzení určitých operací nebo pro rozšířené zadávání formulářů. Obsah stránky pod dialogem musí být inertní aby s ním uživatel nemohl interagovat. Někdy je to chybně řešené pouze překrytím obsahu HTML prvkem, což uživateli stále umožňuje dosáhnout obsahu pomocí klávesnice. Pro správnou přístupnost je třeba okno také řádně popsat pomocí ARIA.

Na základě zvolených komponent byla k implementaci vybrána webová aplikace, která slouží pro vytváření editovatelných příspěvků. Ty mohou ostatní uživatelé procházet a vytvářet k nim komentáře.

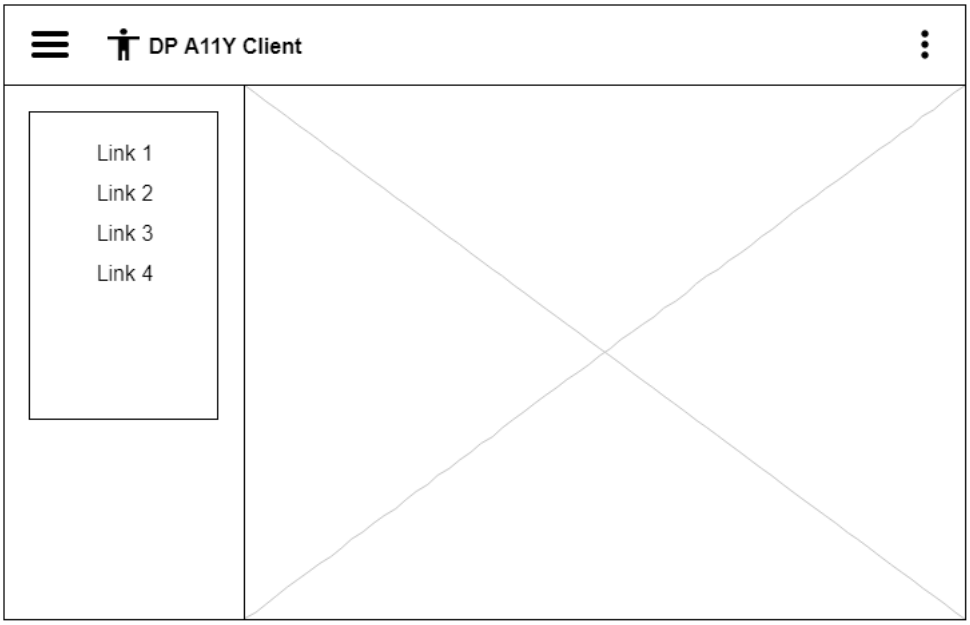
7.1.2 Uživatelské rozhraní

Kromě návrhu komponent je dobré vytvářet návrh rozložení sekcí webové aplikace. Jak bylo již zmíněno v kapitole 5.1, aplikace by měly obsahovat základní oblasti stránky, podle kterých se mohou uživatelé s asistenčními

nástroje pohybovat po webové stránce. Webová aplikace vzniklá v rámci této práce obsahuje následující sekce definované pomocí sekčních HTML elementů.

- `<header>` - hlavička aplikace, které obsahuje uživatelské menu a tlačítko pro zobrazení resp. skrytí navigace
- `<aside>` - boční kontejner, který slouží pro hlavní navigaci aplikace
- `<nav>` - hlavní navigace aplikace, pro udržení responzivního designu aplikace je skrývatelná
- `<main>` - kontejner pro hlavní obsah webu, je to místo ve kterém se mění obsah podle toho, kde se uživatel v aplikaci nachází

Aplikace je uzavřena nepřihlášeným uživatelům, kteří se nejprve musí zaregistrovat. Registrace a přihlášení slouží k demonstraci přístupnosti formulářů včetně uživatelských notifikací. Po přihlášení je uživatel přesměrován na hlavní obrazovku aplikace. Ta obsahuje hlavní oblasti rozvržené podle wireframe návrhu na obrázku 7.1.

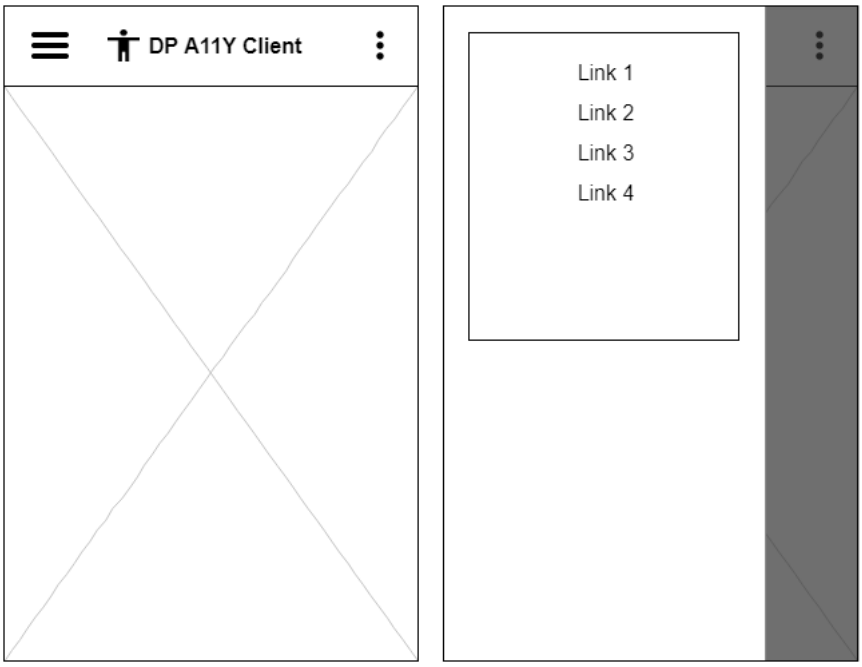


Obrázek 7.1: Základní rozvržení aplikace

Z návrhu je vidět hlavička, která vlevo obsahuje již zmíněné tlačítko pro uživatelské menu a vpravo tlačítko pro navigaci. Navigace je situována uvnitř oblasti pro boční kontejner a slouží pro demonstraci přístupnosti webového

obsahu, který se skrývá mimo obrazovku. V hlavní oblasti, vyobrazené pomocí šedého křížku, se nachází různý obsah podle stavu aplikace.

K podpoře přístupnosti je dobré webové aplikace vyvíjet tak, aby byly přístupné na všech zařízeních. Na obrázku 7.2 je vidět rozmístění oblastí pro mobilní zařízení. Navigace je primárně skryta a po jejím zobrazení překrývá zbylý obsah. Aplikace tedy musí zohledňovat velikost zařízení a při zobrazení navigace na displejích s malým rozlišením označovat všechen zbylý obsah jako inertní.



Obrázek 7.2: Základní rozvržení aplikace na mobilních zařízeních

7.1.3 Aplikační rozhraní

Jak bylo zmíněno v kapitole s technologiemi, které byly využity pro implementaci webové aplikace, aplikační rozhraní je řešeno pomocí REST API implementovaného ve frameworku Express běžícím na technologii Node.js.

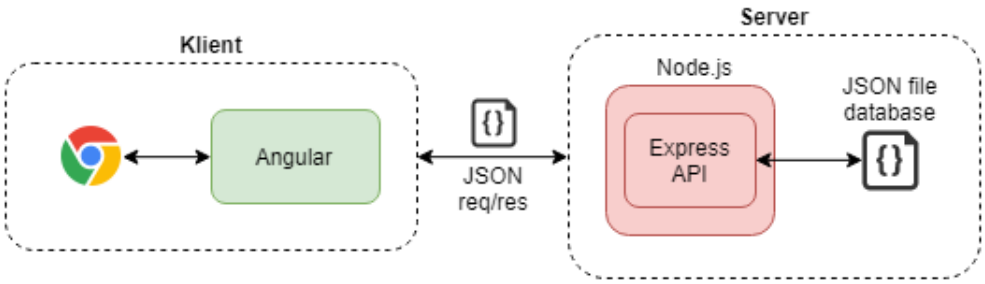
Poskytuje autentizaci a registraci uživatelů a ukládání potřebných dat aplikace. Komunikace probíhá pomocí HTTP požadavků, na které API odesílá odpověď ve formátu JSON.

Vzhledem k primárnímu účelu aplikace, kterým je demonstrace přístupnosti, bylo rozhodnuto data uchovávat pomocí NoSQL dokumentové databáze v JSON formátu. NoSQL dokumentová databáze je mechanismus uklá-

dání strukturovaných dat, většinou ve formátu JSON [33]. Takto vzniklá data v rámci aplikace jsou uloženy do souborů na lokálním disku. Pro potřeby této diplomové práce je to dostačující řešení a pro budoucí zprovoznění aplikace na jakémkoliv počítači je to nejjednodušší přístup, neboť není třeba instalovat žádnou lokální databázi.

7.1.4 Struktura aplikace

Na základě navržených technologií a rozvržení jednotlivých částí aplikace vznikla klientská a serverová část komunikující mezi sebou pomocí HTTP požadavků využívajících JSON formát dat. Zjednodušený návrh celkové architektury je zobrazen na obrázku 7.3. Na klientu v prohlížeči běží webová aplikace vytvořená pomocí frameworku Angular, která využívá aplikačního rozhraní implementovaného ve frameworku Express. Data ve formátu JSON jsou na straně serveru ukládány do souborů na lokálním disku.



Obrázek 7.3: Struktura klientské a serverové části aplikace

Na tyto dvě části je také rozdělen popis implementace v následujících dvch kapitolách.

7.2 Implementace serverové části

Implementační část diplomové práce začala vytvořením aplikačního rozhraní. Většinou je to standardní postup při vývoji webových aplikací, které využívají jako zdroj dat API. Je samozřejmě možné si serverovou část nasimulovat na straně klienta, ale i v tomto případě je nutná znalost budoucích URI (Uniform Resource Identifier) pro jednotlivé požadavky.

Pro správnou implementaci byla nejdříve navržena architektura rozhraní a vybrány knihovny potřebné pro běh API. Poté navrženy URI pro jednotlivé

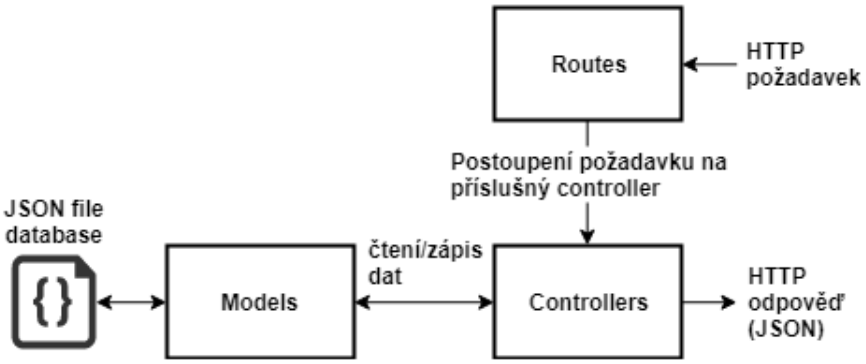
routy (routes), k nim implementované kontrolery (controllers) a nakonec modely pro správu potřebných dat vznikající webové aplikace.

7.2.1 Architektura aplikačního rozhraní

Pro srozumitelnost a modularitu vytvořeného API, bylo rozděleno zpracování HTTP požadavků do tří částí.

1. **Routes (routování)** slouží pro předání podporovaných HTTP požadavku podle URI na konkrétní funkci kontroleru, která požadavek zpracuje.
2. **Controllers (kontrolery)** jsou třídy obsahující funkce pro obsluhu čtení, zápisu a mazání dat z určitého modelu. Po zpracování požadavku odesílají HTTP odpověď na klienta.
3. **Models (modely)** udržují strukturu dat a obsahují logiku pro jejich ukládání, úpravu a mazání.

Při pohledu na architekturu z příchozího HTTP požadavku, jeho zpracování probíhá podle modelu na obrázku 7.4. Požadavek dorazí na server a pomocí routování je postoupen na příslušný kontroler resp. jednu z jeho funkcí. Podle ní se provede potřebná operace modelu, který vrátí požadovaná data. Ty se poté odesílají klientovi v HTTP odpovědi na původní požadavek ve formátu JSON.



Obrázek 7.4: Architektura aplikačního rozhraní

Tyto tři části jsou detailněji rozebrány dále. Podle těchto částí byla také rozdělena struktura projektu.

7.2.2 Struktura a využité knihovny

Server je spuštěn pomocí hlavního souboru, jenž byl podle konvencí pojmenován `index.js`. V něm se importují potřebné knihovny, je zaregistrováno routování a spustí se server na zadaném portu.

Kromě knihovny `express` obsahující samotný framework bylo použito ještě několik dalších NPM knihoven a middleware funkcí. Pro potřeby konfigurace byla zvolena knihovna `config`. Umožňuje pomocí konfiguračního souboru `default.json` definovat libovolnou konfiguraci využitelnou skrze celý projekt. Dále to byla knihovna `express-bearer-token` poskytující middleware pro parsování autentizačního tokenu. Detailněji je popsána v kapitole 7.2.3. Pro vývoj bylo také využito knihovny `morgan`, která trasuje do konzole veškeré příchozí požadavky a stav jejich zpracování.

Mimo využití existujících knihoven byly vytvořeny dvě vlastní middleware funkce. První byla využita pro definování CORS (Cross-origin resource sharing) hlaviček HTTP odpovědí. Funkce na každé odpovědi nastaví hlavičky (headers), díky kterým pak může dojít ke komunikaci mezi serverem a klientem. Druhý middleware byl vytvořen pro zavedení latence při komunikaci. Při vyvíjení serveru i klienta na jednom stroji nedochází k latenci sítě. Takové pozdržení může způsobit špatné vykreslení obsahu, a tím i problémy s přístupností ve webové aplikaci. Hlavně z tohoto důvodu byla latence do HTTP odpovědí zavedena.

Kód hlavního souboru `index.js` pro spuštění serveru je uveden v kódu 7.1.

Kód 7.1: Hlavní soubor API

```
const app = express();

app.use(bearerToken());
app.use(cors);
app.use(latency);

require('./startup/routes')(app);

const port = process.env.PORT || config.get('port');
app.listen(port, () => {
  console.log(`Listening on port ${port}...`);
});
```

V kódu jsou vynechány importy knihoven. Ve volání `require()` je vidět

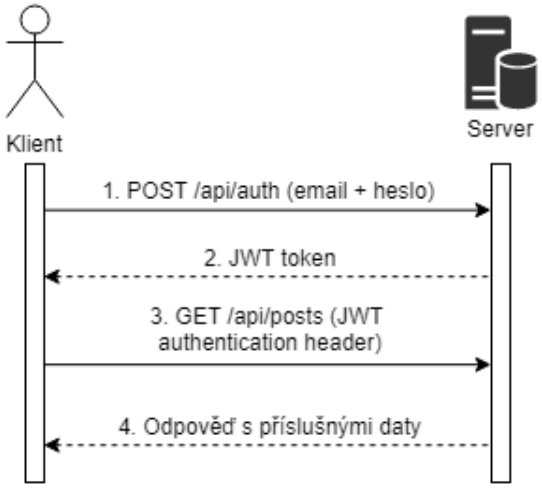
registrace routování, podle kterého se obsluhují všechny příchozí požadavky. Jeho detailnější popis je zmíněn v kapitole 7.2.4.

7.2.3 Autentizace

Aplikace je uzavřena nepřihlášeným uživatelům, bylo tedy třeba vyřešit autentizaci. Bylo zvoleno řešení pomocí JSON Web Token (JWT). Je to otevřený standard RFC 7519, který definuje kompaktní a samostatně stojící způsob zabezpečeného přenosu informací mezi dvěma stranami pomocí JSON token objektu [34]. Přenos informací může být označen za bezpečný, protože objekt s tokenem je digitálně podepsaný pomocí privátního klíče.

JWT má více využití, tím hlavním je však autentizace uživatelů pro webové aplikace. Jakmile se uživatel prokáže platnými přihlašovacími údaji je mu odeslán vytvořený JWT token, který je před tím zakryptovaný pomocí privátního klíče. Tento token je pak pro autentizaci přidáván do hlavičky každého požadavku, který klient na server odesílá.

V API diplomové práce autentizace probíhá odesláním `POST` požadavku na URI `/api/auth`. V těle požadavku jsou očekávány přihlašovací údaje. Pokud jsou údaje úspěšně ověřeny, vytvoří se JWT token, který je odeslán jako odpověď na klienta. Diagram autentizace je vidět na obrázku 7.5.



Obrázek 7.5: Autentizace klienta a následná komunikace

Jak je vidět v komunikaci na obrázku 7.5, po úspěšné autentizaci musí klient odesílat JWT token v hlavičce každého požadavku. Důvodem je bezstavovost API, nedrží si konekci s klientem a ten se tak musí serveru autentizovat při každém požadavku. Token se přikládá do hlavičky `Authorization`

podle ukázky kódu 7.2.

Kód 7.2: Podoba autentikačního tokenu

```
Authorization: Bearer <token>
```

Autentizace na úrovni API je implementována pomocí middleware `auth`. Ten je při vytváření routování přidán do každého požadavku, který má být přístupný pouze přihlášeným uživatelům. Kód tohoto middleware provede verifikaci tokenu. Pokud verifikace proběhne v pořádku požadavek postoupí k vykonání. V opačném případě je požadavek odeslán s chybovým kódem 400 (Bad request).

Parsování hlavičky požadavku obsahující token probíhá pomocí middleware, který je součástí frameworku Express `express-bearer-token`. Token je přiřazen do proměnné `token` objektu požadavku.

7.2.4 Routes

Jak bylo zmíněno u architektury aplikačního rozhraní, routy (routes) slouží pro nasměrování požadavku k příslušné funkci kontroleru. Zároveň se routováním definují dostupné URI rozhraní.

Nejprve byla navržena struktura všech routů, která vycházela z potřeb navržené webové aplikace. Bylo potřeba spravovat autentizaci, správu uživatelů, jejich příspěvky a komentáře. To jsou hlavní zdroje, pro které se muselo v API vytvořit URI, pomocí kterých se na ně přistupuje.

Pro přehlednost výsledného kódu aplikačního rozhraní bylo zvoleno definování dostupných routů v odděleném skriptu, jehož ukázka je vidět na výpisu kódu 7.3.

Kód 7.3: Skript pro definování routů

```
const express = require('express');
const authRoute = require('../routes/auth');
const usersRoute = require('../routes/users');
const postsRoute = require('../routes/posts');

module.exports = (app) => {
  app.use(express.json());
  app.use('/api/auth', authRoute);
  app.use('/api/users', usersRoute);
  app.use('/api/posts', postsRoute);
```

```
}
```

Skript slouží jako rozcestník požadavků, jenž podle typu zdroje v URI určuje, který route ho zpracuje. Příkladem může být zdroj `users`. Pokud přijde požadavek s URI `/api/users` projde do skriptu `usersRoute`. Ten má podobu znázorněnou v kódu 7.4.

Kód 7.4: Skriptu definující Express route

```
const express = require('express');
const router = express.Router();

const auth = require('../middleware/authorization');
const UsersController = require('../controllers/
   usersController');

router.get('/:userId', auth, (req, res) => {
  UsersController.getUser(req, res);
});
```

Jedná se o zkrácený výpis kódu, který pro ukázku obsahuje obsluhu pouze jednoho routu. Konkrétně se jedná o route obsluhující požadavek `GET` s URI `/api/users/:userId`, který slouží pro získání uživatele podle jeho identifikátoru. Stejným způsobem se definují i zbylé obsluhy požadavků. Na ukázce je také vidět použití již zmíněného `auth` middlewaru. Ten určuje, že na tento zdroj může přistupovat pouze autentizovaný uživatel. Pro zpracování se využívá funkcí kontroleru viz následující kapitola.

7.2.5 Controllers

Z daného routu je požadavek postoupen na příslušný kontroler resp. jednu z jeho funkcí. Kontroler byl vytvořen pro každý zdroj aplikačního rozhraní. Řeší validaci požadavku z pohledu parametrů a dat, která jsou zaslána v jeho těle. Využívají funkce modelu pro správu dat a odesílají příslušnou odpověď zpět klientovi.

Po přijetí požadavku kontrolerem dochází k validaci jeho požadovaných parametrů. Parametrem je myšlen například identifikátor zdroje, na který se požadavek dotazuje. Validace je specifická podle typu kontroleru a požadavku.

Pokud je validace parametrů úspěšná dochází k validaci datového ob-

jektu, který zasílá klient v těle požadavku. Pro to byla zvolena knihovna `joi`. Datový objekt je zasílán klientem u HTTP požadavků s metodami `POST` a `PUT`. Pomocí `joi` lze definovat vzorové schéma objektu, který je očekáván v těle požadavku. Toto validační schéma se pak funkcí `validate()` porovná s přijatým objektem. Příklad validace je vidět na výpisu kódu 7.5.

Kód 7.5: Validace objektu pomocí knihovny joi

```
function validate(object) {
  const schema = {
    email: Joi.string().email(),
    name: Joi.string().min(5).max(255)
  };

  return Joi.validate(object, schema);
}
```

Při chybné validaci dat funkce vrací pro uživatele čitelné chybové hlášky. Ty spolu s chybovým kódem mohou být zaslány jako odpověď na požadavek. V případě chybných požadavků jsou v API použity chybové kódy podle tabulky 7.1.

Status kód	**Význam**
400	Bad request (špatně definovaný požadavek)
401	Unauthorized (uživatel nebyl autentizován)
403	Forbidden (klient nemá práva k požadovanému obsahu)
404	Not found (server nenalezl požadovaný zdroj)

Tabulka 7.1: Chybové HTTP status kódy [35]

V případě úspěšné validace požadavku jsou volány funkce modelů pro vytvoření, úpravu a získání uložených dat.

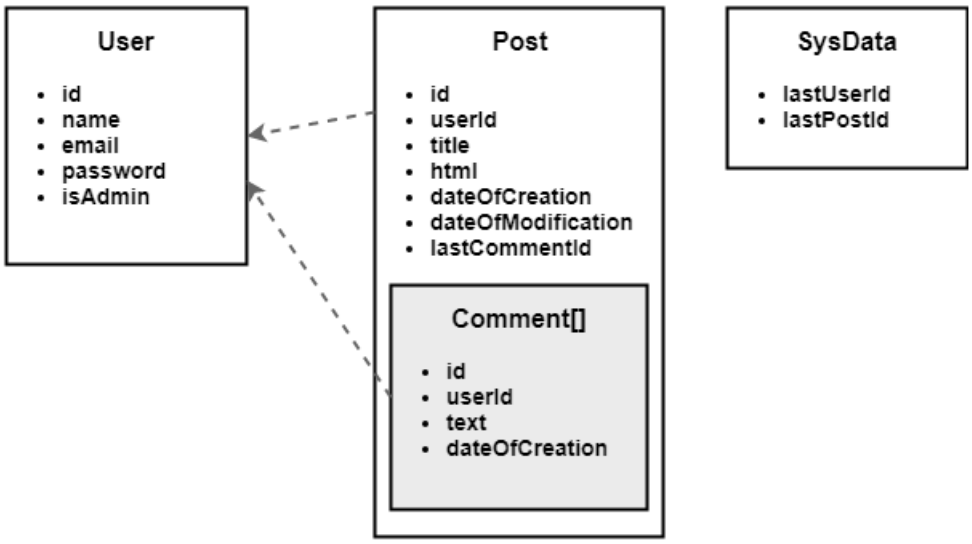
7.2.6 Models

Modely udržují strukturu ukládaných dat a poskytují funkce pro jejich manipulaci. API bylo navrženo tak, aby umožňovalo využívat libovolné modely. V rámci práce byly zvoleny vlastní modely, které data ukládají pomocí NoSQL přístupu do souborů, ale mohly by být vyměněny např. za modely ukládající data do libovolné databáze.

Uložení dat do databáze je při implementaci nad Node.js bráno jako

asynchronní operace. Aby byly modely samostatně stojícím celkem a byla dodržena jejich modularita každá funkce modelu vrací jako návratovou hodnotu objekt `Promise`. Je to mechanismus umožňující vykonávání asynchronních operací. Tento objekt používají jako svou návratovou hodnotu i modely frameworků spravujících různé databáze. Pokud by tedy byla potřeba změnit způsob pro uchování dat, vyměnil by se pouze kód modelů, ale volání z kontrolerů by zůstalo stejné.

V rámci práce vznikly dva modely obsluhující dokumenty s daty. Jeden pro správu uživatelů a jeden pro správu příspěvků včetně jejich komentářů. Schéma dat, které modely spravují je znázorněno na obrázku 7.6.



Obrázek 7.6: Schéma dat spravovaných pomocí modelů

Filtrování a další operace nad JSON objekty byly řešeny pomocí JavaScript funkcí `filter()`, `sort()` a dalších. Při vytvoření nebo změně dat jsou objekty uloženy do souborů. Pro synchronizaci dat do souborů byla využita Node.js knihovna `fs`, konkrétně její funkce `writeFileSync()`.

7.2.7 Popis výsledného API

Konečná podoba API pro přístupnou webovou aplikaci obsahuje celkem dvanáct koncových bodů pro manipulaci s daty pomocí HTTP požadavků. V tabulce 7.2 je vidět jejich podoba. Pro každý route je v ní uvedena HTTP metoda a krátký popis.

Metoda	Route	Popis
POST	/api/auth	autentizace uživatele
POST	/api/users	vytvoření uživatele
PUT	/api/users/:id	úprava uživatele
GET	/api/users/:id	získání uživatele podle id
GET	/api/users/:id/posts	příspěvky uživatele
POST	/api/posts	vytvoření příspěvku
PUT	/api/posts/:id	úprava příspěvku
GET	/api/posts?limit&offset	filtrování příspěvků
GET	/api/posts/:id	získání příspěvku podle id
DELETE	/api/posts/:id	smazání příspěvku
POST	/api/posts/:id/comments	vytvoření komentáře
DELETE	/api/posts/:id/comments/:id	smazání komentáře

Tabulka 7.2: Výsledná podoba routování API

7.3 Implementace klientské části

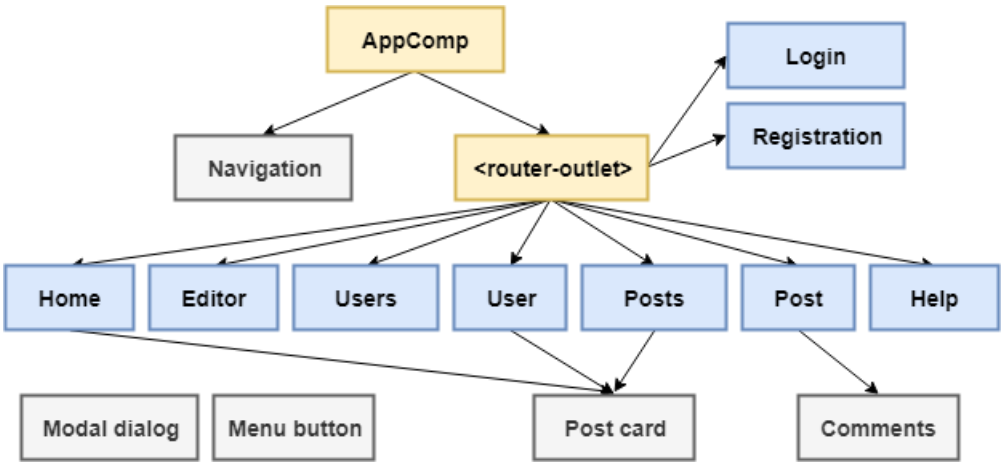
Po dokončení implementace aplikačního rozhraní následovala webová aplikace. Tato část vycházela z technik zmíněných v kapitole 5. Byla vytvořena pomocí frameworku Angular s dodržením doporučené architektury zmíněné v kapitole 6.4.4. Nejdříve byla navržena architektura komponent, implementovány servisní třídy (services) pro získání dat z REST API, vytvořeno routování a poté postupně implementovány jednotlivé komponenty. Všechny komponenty navržené a implementované v rámci této práce neobsahují žádný kód ani CSS styly z externích knihoven. Pro stylování bylo využito vizuálního jazyka Google Material Design. Ten definuje vizuální podobu komponent včetně rozměrů a udává správné způsoby jejich využití.

7.3.1 Návrh komponentové architektury

Prvním krokem implementace webové aplikace bylo navržení komponent včetně určení závislostí mezi nimi. Tento krok vycházel z navržené funkcionality a komponent, které byly zvoleny pro demonstraci implementace přístupnosti.

Diagram konečného návrhu komponent včetně jejich závislostí je na obrázku 7.7. Žlutě jsou vyznačeny komponenty, které slouží pouze k definování struktury ostatních komponent. Je to hlavní komponenta `AppComponent`

a `<router-outlet>`, která slouží pro vkládání komponent do DOM podle aktivního routu (viz kapitola 7.3.3). Modře jsou vyznačeny komponenty dosažitelné pomocí routování. Tedy kořenové komponenty, na které se může uživatel dostat pomocí změny URL prohlížeče. Šedou barvu mají znovupoužitelné komponenty, které mohou být vkládány resp. obsluhovány libovolnou komponentou za účelem zobrazení nějaké informace, nebo vytvoření ovládacího prvku.



Obrázek 7.7: Diagram komponent webové aplikace

Podle diagramu komponent byl vytvořen Angular projekt a inicializovány všechny komponenty. Pro obsluhu projektu byl používán NPM balíček s nástrojem `angular-cli`. Je to rozhraní pro příkazovou řádku, které přináší vše potřebné pro obsluhu Angular projektů. Umožňuje generovat projekty, komponenty a další části Angular aplikace, testovat, buildovat a spouštět aplikace na lokálním webovém serveru.

7.3.2 Implementace servisních tříd

Účelem servisních tříd je sdílení funkcionality, která není přímo spojená s komponentou (viz kapitola 6.4.3). Pro účely aplikace vyvíjené v rámci této práce byly využity dva typy servisních tříd.

1. Servisní třídy pro HTTP komunikace s API.
2. Servisní třídy pro účely obsluhy uživatelského rozhraní.

První typ servisních tříd slouží primárně pro komunikaci s aplikačním rozhraním. Třídy mohou obsahovat také metody podporující komunikaci.

Příklad může být metoda autentizační servisní třídy `isUserLoggedIn()`, která vrací `boolean` hodnotu pro kontrolu, zda je uživatel přihlášen. Pro odesílaní HTTP požadavků byla využita Angular třída `HttpClient`. Komunikaci s API řeší následující čtyři servisní třídy.

- `AuthService` - poskytuje metody týkající se přihlášení uživatelů včetně správy autentizačního tokenu.
- `UsersService` - obsahuje metody pro registraci uživatele a obsluhu požadavků na uživatele.
- `PostsService` - slouží pro vytváření a úpravu příspěvků a poskytuje metody pro filtrování příspěvků ostatních uživatelů.
- `CommentsService` - poskytuje metody pro vytvoření a smazání komentáře.

Servisní třídy pro obsluhu uživatelského rozhraní slouží pro definování vlastností, které nějakým způsobem ovlivňují společné komponenty. Příkladem může být `LoaderService` s vlastností `loaderState`, která zobrazuje animaci načítání při každém HTTP požadavku. Dalšími zástupci těchto tříd jsou `NavigationUiService` a `ConfirmModalService`, které jsou detailněji popsány u implementace komponent, kterých se týkají.

7.3.3 Routování

Routování umožňuje navigaci z jednoho pohledu (view) komponenty na jiný [36]. Je vyvoláno nějakou interakcí uživatele. V Angularu je doporučeno pro routování vytvořit modul, který se zavede při spouštění aplikace do `AppModule`. Podle konvencí by měl být pojmenován `AppRoutingModule` [36].

Tento modul byl také vytvořen v rámci vyvíjené aplikace. Přináší to přehlednost a dodržuje doporučení vývojářů Angular frameworku pro co největší modularitu kódu. V modulu je definováno pole objektů typu `Routes`, které obsahuje mapování URL aplikace na její komponenty. Ukázka jednoho z mapování je na výpisu kódu 7.6.

Kód 7.6: Definice Angular routu

```
const routes: Routes = [{
  path: 'home',
  component: HomeComponent,
```

```
  canActivate: [AuthGuard]
}];
```

Pro každý takto definovaný Angular `Route` bylo využíváno následujících vlastností.

- `path` - řetězec pro porovnání s aktuálním URL
- `component` - komponenta, která má být zobrazena v případě aktivace tohoto routu
- `canActivate` - odkazuje na handler, ve kterém je kontrolováno, zda může být komponenta vzhledem k aktuálnímu stavu aplikace zobrazena

Po tomto kroku byla vytvořena struktura aplikace, implementované servisní třídy a routování. Zbývala implementace jednotlivých komponent.

7.3.4 Komponenta Login

Implementaci byla započata komponentami pro přihlášení a registraci. Obě využívají stejný přístup k validaci a zobrazování přístupných upozornění o chybách, proto je zde zmíněna pouze jedna z nich.

Pro validaci formulářů poskytuje Angular třídu `FormGroup`. Je to objekt obsahující jednotlivé ovládací prvky formuláře definované jako `FormControl` objekty, na kterých lze definovat validátory. Takto vytvořený objekt formuláře pro přihlášení má podobu dle kódu 7.7.

Kód 7.7: Angular objekt mapující formulář

```
this.loginForm = new FormGroup({
  email: new FormControl('', [ Validators.email,
     Validators.required ]),
  password: new FormControl('', [ Validators.minLength(5),
     Validators.required ])
});
```

Při odeslání formuláře je možno provést kontrolu validity všech jeho ovládacích prvků. V případě chyby validace lze zjistit chybný prvek formuláře a zobrazit uživateli příslušnou chybu. Pro zobrazení chyb byl zvolen postup podle kódu 7.8.

Kód 7.8: Zobrazení validačních chyb formuláře

```
<div [ngClass]="{ 'show' : submitted && showEmailError }"
    role="alert" aria-live="assertive">
  <i class="material-icons">error</i> {{emailErrMsg}}
</div>
```

Chyba se zobrazuje automaticky pomocí Angular direktivy `ngClass`. V ní se kontrolují nastavované `boolean` vlastnosti, konkrétně zda byl formulář odeslán a jestli daný ovládací prvek obsahuje validační chybu. Chyby byly zobrazeny pomocí HTML elementu s rolí `alert` a atributem `aria-live`. Tím je řečeno asistenčním nástrojům, že se jedná o část HTML, která se pravděpodobně dynamicky mění [37]. Nástroje pak na tuto změnu jsou schopni reagovat a informovat o tom jejich uživatele. Vzhledem k tomu, že byly chyby generovány dynamicky, tak je to jediná cesta, jak o chybě informovat asistenční nástroje. Hodnota `assertive` říká, že se jedná o důležitou změnu a asistenční nástroj má přerušit aktuálně čtený obsah a reagovat na ni [37].

7.3.5 Komponenta Navigation

Pro komponentu navigace byl zvolen návrh Material Design - Navigation Drawer. Jedná se o navigaci známou z produktů Googlu nebo z většiny nativních Android aplikací. Navigace může být buď trvale zobrazena nebo pomocí tlačítka skryta mimo obrazovku.

Právě skrývání mimo obrazovku je důvod, proč byla komponenta zvolena. Obsah po skrytí by neměl být přístupný. Zároveň při menším rozlišení obrazovky navigace překrývá primární obsah aplikace, který v tu chvíli také nesmí být přístupný resp. musí být inertní.

Pro potřeby této komponenty byla implementována také servisní třída `NavigationUiService`. Ta udržuje informace o stavu a maximální šířce obrazovky, při které navigace překrývá primární obsah. Důvodem volby servisní třídy bylo umožnění distribuce těchto vlastností i pro hlavní komponentu `AppComponent`, která pak označuje obsah jako inertní.

Navigace má tedy dva stavy, při kterých se nastavují následující vlastnosti.

1. **Navigace je skrytá**: v tomto případě je pomocí CSS stylů posunuta mimo obrazovku a na její kořenový HTML element je nastaven atribut `inert`, který zamezuje uživateli s ní jakkoli interagovat. Zobrazí se

pomocí tlačítka v hlavičce stránky.

2. **Navigace je zobrazená**: zde musely být řešeny dvě různé situace podle minimální šířky obrazovky, deklarované v proměnné `widthLimit` v `NavigationUiService`.

   (a) Šířka obrazovky je větší: v tomto případě je navigace zobrazena a není třeba dalších nastavení k její přístupnosti.

   (b) Šířka obrazovky je menší: v tu chvíli navigace překrývá primární obsah aplikace, ten musí být nastaven jako inertní.

Další implementovanou funkcionalitou byl handler, který kontroluje šířku okna při její změně. Pokud je překročena již zmíněná hodnota `widthLimit` nastavuje dle potřeby inertnost navigace.

7.3.6 Komponenta MenuButton

Další komponentou bylo menu vyvolané pomocí tlačítka. Komponenta se skládá ze dvou HTML elementů. Jedním je tlačítko `<button>` a druhým je seznam `<ul>` obsahující nějaké akce nebo odkazy. Komponenta byla zvolena, protože disponuje několika následujícími technikami, které pro ni musí být implementovány:

- propojení standardních HTML elementů do jedné komponenty,
- vlastní obsluha aktivního prvku,
- ovladatelnost pomocí klávesnice,
- dynamicky se měnící ARIA atributy vzhledem ke stavu komponenty.

Z pohledu HTML a příslušných ARIA atributů má komponenta podobu naznačenou v kódu 7.9.

Kód 7.9: HTML struktura komponenty MenuButton

```
<div>
  <button id="menuBtn"
      aria-label="Expandable menu"
      aria-controls="menu"
      aria-haspopup="true"
      aria-expanded="false">
    Menu
```

```
  </button>
  <ul id="menu" role="menu" aria-labelledby="menuBtn">
    ...
  </ul>
</div>
```

Tlačítko disponuje popisem a dalšími třemi atributy. První atribut je `aria-controls`, který určuje prvek, jenž je tlačítkem ovládán. Další dva atributy tvoří mechanismus pro popisování popup obsahu, `aria-haspopup` říká, že tlačítko obsluhuje popup prvek a `aria-expanded` obsahuje `booelan` hodnotu určující, jestli je popup prvek otevřený. Hodnota se dynamicky mění v závislosti na stavu komponenty. Seznam ovládacích prvků obsahuje jediný ARIA atribut, jenž určuje, že alternativní popis prvku je zděděný z popisu tlačítka.

Pomocí atributů byla vyřešena část popisu prvku. Zbývala jeho ovladatelnost pomocí klávesnice. Menu se otevře po standardním stisku tlačítka. Tento stisk ještě ARIA doporučuje rozšířit o klávesy šipka nahoru a šipka dolu [38]. Po otevření se focus přesune na první prvek menu. V něm jsou klávesy obsluhovány pomocí události `keydown`. V rámci menu byla implementována následující obsluha kláves.

- Šipka dolu: posune focus na následující prvek, pokud aktuální aktivní prvek posledním v menu, focus je posunut na první prvek.
- Šipka nahoru: posune focus na předchozí prvek, pokud je aktuální aktivní prvek prvním v menu, focus je posunut na poslední prvek.
- Home: posune focus na první prvek menu.
- End: posune focus na poslední prvek menu.

Prvky menu jsou do komponenty vkládány jako vnořená komponenta. To usnadňuje a zpřehledňuje jeho vytvoření. Výsledné komponenta se pak v šabloně libovolné jiné komponenty používá dle kódu 7.10.

Kód 7.10: Definování komponenty MenuButton

```
<ui-dropdown-menu title="Razeni prvku" icon="sort">
  <ui-menu-item (click)="orderUserPosts(true)" label="A-Z">
     </ui-menu-item>
  <ui-menu-item (click)="orderUserPosts(false)" label="Z-A">
     </ui-menu-item>
</ui-dropdown-menu>
```

Komponenta `menu-item` předává událost `click` jako její výstup. Tím je umožněna obsluha této události v komponentě, která menu vytváří.

7.3.7 Komponenta Editor

Demonstrace klávesových zkratek a panelu nástrojů byla implementována na jednoduchém textovém editoru. Panel nástrojů, který tvoří skupina ovládacích prvků slouží pro úpravu textu v editoru.

Pro vytvoření editovatelného obsahu byla využita vlastnost `designMode` objektu `document`. Vlastnost umožňuje definovat, zda je objekt dokumentu editovatelný [40]. Celý editor byl pak vytvořen jako `<iframe>`, kterému se tato vlastnost nastavila na hodnotu `'on'`. To poté umožňuje vykonání příkazů na úpravu textu v rámci celého editoru pomocí funkce `execCommand()` na objektu `document` (viz ukázkový kód 7.11).

Kód 7.11: Editace obsahu editovatelného dokumentu.

```
var editorDoc = document.querySelector('#editor')
    .contentDocument;
editorDoc.designMode = 'on';
editorDoc.execCommand('bold');

// iframe v~HTML
<iframe id="editor"></iframe>
```

Panel nástrojů pro editaci textu byl vytvořen pomocí několika tlačítek a výběrových polí. Ty volají obslužnou metodu pro vykonání funkce `execCommand()`, během které bylo ještě vyřešeno nastavení focusu zpět do editoru a odchycení případných výjimek samotné editace.

Pro implementaci přístupnosti bylo opět vycházeno z ARIA specifikace. Ta pro panel nástrojů definuje roli `toolbar` a doporučuje pohyb v rámci něj implementovat pomocí kláves šipek (pro horizontální toolbar levá a prává šipka) [41]. Klávesa TAB má pak sloužit pro focus na prvek, který následuje po toolbaru. Tato klávesová obsluha byla implementována pomocí odchycení události `keydown` a vyvolání focusu na další prvek přímo v kódu.

Kromě obsluhy focusu pro panel nástrojů byly pro celou komponentu implementovány následující dvě klávesové zkratky.

- `Alt+Shift+T`: přesune focus na panel nástrojů.

- `Alt+Shift+E`: přesune focus do editoru.

Klávesové zkratky pro úpravu textu jako například tučné písmo či kurzívu, jsou implementovány pro `designMode` nativně. Mezi nimi ale chybí zkratky pro zarovnání odstavců. Ty byly vytvořeny a obslouženy v kódu a mají následující podobu.

- `Ctrl+Shift+L`: zarovnání odstavce vlevo.
- `Ctrl+Shift+R`: zarovnání odstavce vpravo.
- `Ctrl+Shift+E`: zarovnání odstavce na střed.
- `Ctrl+Shift+J`: zarovnání odstavce do bloku.

Na všechny ovládací prvky byl jejich popis a klávesová zkratku uvedena pomocí atributu `aria-label`. To umožňuje nevidomým uživatelům při prvním průchodu toolbarem zjistit účel a klávesovou obsluhu každého prvku a poté jí v editoru využívat.

7.3.8 Komponenta Dialog

Poslední komponentou, která je obtížná z pohledu dodržení přístupnosti aplikace je modální dialog. Je to komponenta, která se přidává do DOM dynamicky a překrývá veškerý obsah stránky.

Modální dialog resp. jeho vytvořená komponenta byla umístěna na konec `AppComponent`, aby její pozicování pomocí CSS stylů nebylo ovlivněno v rámci jiného HTML elementu a také pro nastavení `inert` atributu viz dále.

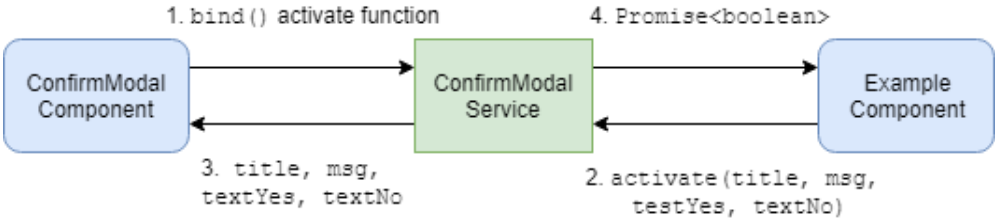
Aby byl dialog ovladatelný skrze celou aplikaci, byla pro něj vytvořena servisní třída `ConfirmModalService`. Ta propojuje komponentu dialogu a poskytuje metodu pro jeho zobrazení všem komponentám aplikace. Propojení a kroky, které vedou k vyvolání dialogového okna (viz také obrázek 7.8) jsou následující.

1. Při inicializaci komponenty `ConfirmModalComponent` dojde k namapování její metody `activate()` pro zpracování odpovědi dialogu na metodu `activate()` servisní třídy.

2. To pak umožňuje libovolné komponentě volat metodu `activate()` z třídy `ConfirmModalService`. Při jejím volání se určují texty, které mají být do dialogu vloženy.

3. Díky namapování je možno texty vypsat do dialogu v komponentě `ConfirmModalComponent`.

4. Komponenta, která dialog vyvolala, pak už jen čeká na odpověď uživatele. Ta je řešena asynchronně pomocí objektu `Promise`.



Obrázek 7.8: Diagram kroků pro potvrzení operace dialogovým oknem

Z pohledu přístupnosti muselo být vyřešeno nastavení inertnosti. Vzhledem k tomu, že dialog je jako element přidán na nejvyšší úroveň root komponenty, stačí nastavit atribut `inert` na všechny ostatní kořenové elementy. Pro indikaci, že je dialog zobrazen byla využita globálně dostupná vlastnost `isConfirmVisible` v `ConfirmModalService`. Pro modální dialog ARIA poskytuje roli `dialog`. Ten má být nastaven na kořenovém elementu dialogu a kromě ní má být přidán atribut `aria-modal` s hodnotou `true` [39]. Pro uživatele využívající klávesnici musí dialog umožnit zavření klávesou ESC [39]. To bylo implementováno pomocí události `keydown` také na kořenovém elementu dialogu.

7.3.9 Výsledná aplikace

Výsledná aplikace se skládá z popsaných komponent a jako celek poskytuje responzivní uživatelské rozhraní demonstrující techniky usnadnění přístupu pro uživatele s různým postižením. Primárně se zaměřuje na techniky pro zrakově postižené uživatele, pro které přináší alternativní popis všech ovládacích a vizuálních prvků, ovládání celé aplikace pomocí klávesnice včetně vlastních klávesových zkratek, HTML strukturu využívající sekční elementy a další v práci zmíněné techniky.

Pro demonstraci usnadnění přístupu zrakově postižených uživatelů je dobré využít některou z dostupných čteček obrazovek, které byly zmíněny v kapitole 4.1.

8 Testování

V této kapitole bude popsáno testování aplikace. Nejprve bude zmíněno testování přístupnosti klientské části. Budou uvedeny nástroje, kterými byla přístupnost webové aplikace ověřována během vývoje. Poté bude popsáno testování z pohledu kontroly implementace jednotkovými testy.

8.1 Testování přístupnosti

Během vývoje klientské části aplikace bylo použito několik nástrojů a způsobů testování přístupnosti. Nejčastěji to byla čtečka obrazovky, kterou byla použita pro manuální testování vznikajících částí. Užitečné nástroje pro přístupnost obsahuje také DevTools v prohlížeči Google Chrome. Dále bylo využito nástroje aXe, jenž kontroluje přístupnost dle analýzy kódu stránky. V neposlední řadě byl také používán plugin do Google Chrome, který byl mnou vytvořen během předmětu Oborový projekt (KIV/OPIS).

8.1.1 Manuální testování

Jak bylo popsáno v kapitole 2.1, web je přístupný, pokud je v celém rozsahu čitelný a pochopitelný pro uživatele asistenčních nástrojů (viz kapitola 2.1). Je tedy dobré během vývoje testovat web také z tohoto pohledu. K tomuto účelu byla využita čtečka obrazovky NVDA. Pomocí ní je možné testovat ARIA atributy, které se dynamicky mění na základě stavu ovládacího prvku. Dynamicky generované atributy se obecně těžko testují pomocí nástrojů DevTools či axe. Lze to ale ověřit pomocí jednotkových testů, které jsou zmíněny v kapitole 8.2.

Příkladem těžko odhalitelného problému v přístupnosti pomocí automatizovaných testů může být tlačítko vyvolávající menu. Většina nástrojů zkontroluje, zda tlačítko a menu obsahují potřebné ARIA atributy a jestli mají popisné informace. Jestli se ale v tomto případě změní hodnota atributu `aria-expanded` po otevření menu už neověří.

8.1.2 Vlastní rozšíření pro Google Chrome

Jak bylo již zmíněno, během předmětu Oborový projekt bylo mnou vytvořeno rozšíření prohlížeče Google Chrome. Rozšíření slouží pro testování hlavních parametrů přístupnosti webové aplikace v reálném čase. Při každém načtení se web otestuje pro následující parametry:

- definice validní HTML5 hlavičky,
- využití HTML5 nativních elementů,
- definice meta tagu pro podporu mobilních zařízení,
- alternativních popisů všech obrázků,
- popisu ovládacích prvků,
- využití ARIA atributů.

Rozšíření bylo využívání během celého vývoje pro kontrolu vznikající webové aplikace v reálném čase. Bez nutnosti nějaké obsluhy bylo možno detekovat některé nedostatky aplikace.

8.1.3 Testování pomocí DevTools

Vývojářské nástroje prohlížeče Google Chrome poskytují možnosti kontroly vývoje přístupného webu. Mezi dvě hlavní funkcionality určené přímo pro ověření přístupnosti patří:

- audit zaměřený na přístupnosti (Accessibility audit),
- panel pro přístupnost vybraného elementu DOM.

Audit přístupnosti analyzuje aktuálně načtený stav HTML stránky z pohledu správného označení elementů, které vede ke správné interpretaci obsahu pomocí čteček obrazovky [42]. Kromě HTML také kontroluje, zda prvky na stránce mají dostatečný kontrast barev. Audit byl při vývoji spouštěn vždy po dokončení větší části aplikace pro vyhodnocení přístupnosti implementované části.

Panel pro kontrolu přístupnosti elementu byl používán během vývoje také poměrně často. Obsahuje tři oblasti pro analýzu vybraného elementu stránky.

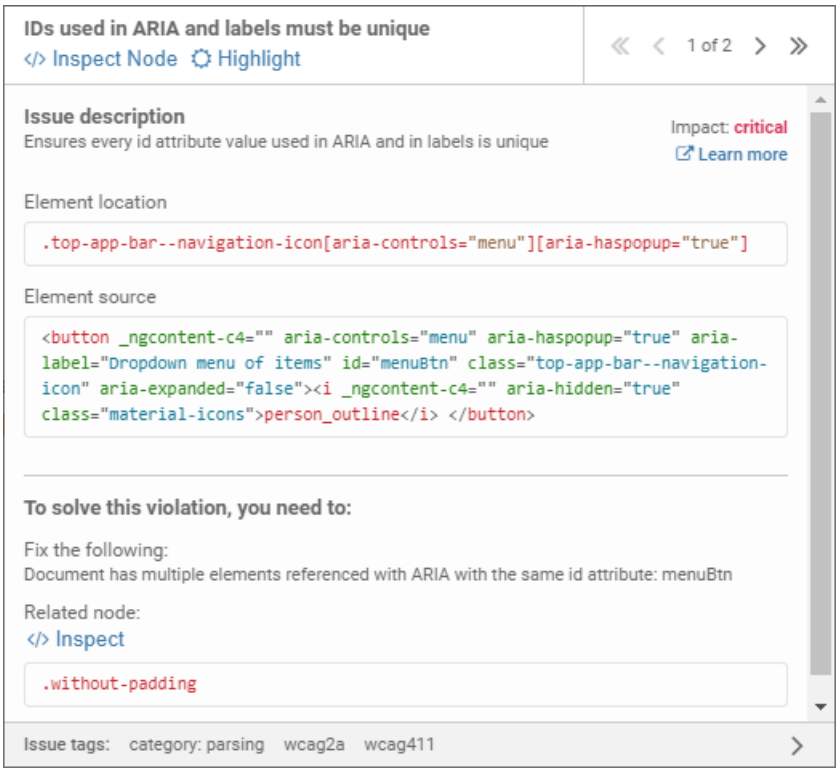
- **Accessibility Tree**: obsahuje strom elementů z DOM relevantních pro čtečky obrazovek. Lze z něj tedy zjistit zanoření elementu vzhledem k finálně prezentovanému obsahu pomocí čtečky. Užitečné to bylo hlavně pro rychlou analýzu toho, z jakých informací, důležitých pro čtečku obrazovky, se daný element skládá.
- **ARIA Attributes**: zobrazuje výčet ARIA atributů elementu. Byl použit při vývoji složitějších komponent, které obsahují několik těchto atributů a interagují s nějakým dalším elementem.
- **Computed Properties**: Prohlížeče dynamicky vypočítávají některé vlastnosti elementů, příkladem může být název prvku [42]. Tyto vlastnosti byly pro kontrolu u vznikajících komponent zobrazovány a dle potřeby byly upravovány ARIA atributy.

8.1.4 Nástroj axe

Axe je testovací nástroj pro usnadnění přístupu na webové stránky a jiná uživatelská rozhraní založená na HTML [43]. Je založen na open source knihovně `axe-core`. Knihovnu lze použít buď přímo v projektu vyvíjené aplikace nebo využít rozšíření pro prohlížeč.

Výhodou zavedení axe přímo do projektu může být jeho spuštění například v rámci integračních testů nebo jednotkových testů pro každou komponentu zvlášť. V této práci bylo během vývoje využito jak rozšíření pro prohlížeč Google Chrome, tak implementace v rámci jednotkových testů (viz kapitola 8.2).

Rozšíření prohlížeče axe je podobné auditu v DevTools, ale obsahuje detailnější analýzu HTML. Pro vzniklou aplikaci bylo využito až na konci vývoje, protože jednotlivé komponenty byly testovány pomocí zavedení axe do jejich jednotkových testů. Bylo tím odhaleno několik nedostatků, které audit v prohlížeči přehlédl. Příkladem chyby, kterou axe odhalil a audit nikoli je využití atributu `aria-controls`, který jako hodnotu nese identifikátor elementu. Na stránce byl využit identifikátor dvakrát a element nesoucí tento atribut tak nevěděl, jaký element ovládá. Ukázka analýzy webu pomocí tohoto nástroje je na obrázku 8.1. Je na něm vidět zmíněná chybu s duplicitními identifikátory.



Obrázek 8.1: Ukázka webového rozšíření axe

8.2 Jednotkové testy

Pro ověření funkcionality kritických částí webové aplikace byly vytvořeny jednotkové testy, pro které Angular využívá framework Jasmine. Testy jsou pak vykonány v testovacím prostředí Karma. V rámci jednotkových testů se práce zaměřovala primárně na ověření přístupnosti komponent. Bylo využito také také knihovny `axe-core`.

V jednotkových testech byla pro komponenty definována a ověřována existence ARIA atributů. Před samotnou implementací komponenty byly na základě specifikace Authoring Practices (viz kapitola 3.5) vytvořeny testy definující povinné ARIA atributy. Na ukázce kódu 8.1 je testování atributů vidět. Konkrétně se jedná o test komponenty MenuButton. Na ukázce je nejdříve uloženo do proměnné tlačítko a po dokončení renderování stránky následně testovány jeho atributy. Stejným způsobem bylo pro komponentu MenuButton testováno menu, které se po kliku na tlačítko zobrazuje.

Kód 8.1: Testování ARIA atributů.

```
it('should have ARIA attributes', () => {
  let btn = fixture.debugElement.nativeElement
```

```
      .querySelector('.dropdown-btn-container button');

  fixture.whenRenderingDone().then(() => {
    expect(btn.getAttribute('aria-label')).toBeTruthy();
    expect(btn.getAttribute('aria-controls')).toBeTruthy();
    expect(btn.getAttribute('aria-haspopup')).toBeTruthy();
    expect(btn.getAttribute('aria-expanded')).toBeTruthy();
  });
});
```

Jednotkové testy u komponent také poskytly mechanismus pro kontrolu dynamicky generovaných atributů. Ukázka takového testu je v kódu 8.2. Konkrétně se jedná o test pro atribut `aria-expand`, který by se měl po otevření menu změnit z hodnoty `false` na `true`.

Kód 8.2: Testování dynamicky generovaného atributu.

```
it('should dynamically change aria-expanded', () => {
  let btn = fixture.debugElement.nativeElement
      .querySelector('.dropdown-btn-container button');

  expect(btn.getAttribute('aria-expanded')).toBe('false');
  component.isMenuOpen = true;
  fixture.detectChanges();
  expect(btn.getAttribute('aria-expanded')).toBe('true');
});
```

V rámci jednotkových testů byla také zařazena kontrola přístupnosti pomocí knihovny `axe-core`. Každá komponenta byla tímto způsobem otestována zvlášť už během vývoje, což usnadnilo finální kontrolu a debugování přístupnosti aplikace jako celku. Spuštění takového testu je vidět na ukázce kódu 8.3. Pomocí `context` se určuje CSS selector elemenetu, jenž je předmětem validace. Validace pak vrací počet nalezených chyb, který očekávám nulový.

Kód 8.3: Testování přístupnosti pomocí axe.

```
it('should have no a11y violations', () => {
  let context = { include: ['.dropdown-btn-container'] };
  axe.run(context, (err, results) => {
    if (err) console.log(err);
    expect(results.violations.length).toBe(0);
  });
});
```

9 Závěr

Cílem této diplomové práce bylo prozkoumat standardy a specifikace pro usnadnění přístupu na web a poté implementovat demonstrační přístupnou webovou aplikaci.

Nejdříve se práce zabývala přístupností webu obecně. Bylo čerpáno z dostupných standardů konsorcia W3C resp. jeho podskupiny WAI. Po analýze standardů se práce zaměřila na specifikaci WAI-ARIA, která definuje způsob, jak implementovat přístupnost a rozšiřovat webové aplikace o nestandardní komponenty. V rámci analýzy byla zmíněna také legislativa ČR a EU týkající se přístupnosti na web. Na základě legislativy byly testovány a zhodnoceny dva weby veřejné správy. U jednoho z nich bylo zjištěno, že požadované standardy nedodržuje.

Před samotnou implementací bylo navrženo několik technik sloužících pro vytvoření přístupné webové aplikace. Jednou z nich byl komponentový přístup vývoje, který přináší jeho zefektivnění a usnadnění. Jedním z frameworků využívajících komponentový přístup je Angular, jenž byl zvolen pro vývoj klientské části aplikace. Pro serverovou část bylo zvoleno REST API využívající framework Express běžící na Node.js.

Realizační část práce začala navržením funkcionality vznikající webové aplikace, uživatelského rozhraní včetně jeho komponent, aplikačního rozhraní a celkové architektury obou částí aplikace. Implementace byla zahájena serverovou částí obsahující REST API. Poté následovala klientská část, jejíž implementace byla rozdělena na část komunikace se serverem a na část uživatelského rozhraní implementovaného pomocí analyzovaných standardů a technik.

Testování obnášelo implementaci jednotkových testů, kterými bylo kromě samotného kódu kontrolováno definování ARIA atributů v jednotlivých komponentách. Z pohledu testování přístupnosti bylo zmíněno několik nástrojů pro její ověření.

Výsledná aplikace demonstruje webové komponenty v přístupné formě pro všechny uživatele. Umožňuje vytvářet formátované příspěvky pomocí jednoduchého wysiwyg editoru a prohlížení příspěvků ostatních uživatelů. Účel webové aplikace vznikl na základě navržených komponent, které byly pro demonstraci přístupnosti zajímavé a něčím specifické. Komponenty je možné použít pro vývoj vlastních webových aplikací v Angularu.

Seznam obrázků

Seznam zkratek

AJAX Asynchronous JavaScript and XML.

API Application Programming Interface.

ATAG Authoring Tool Accessibility Guidelines.

CORS Cross-origin resource sharing.

CSS Cascading Style Sheets.

DI Dependency Injection.

DOM Document Object Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

JS JavaScript.

JSON JavaScript Object Notation.

JWT JSON Web Tokens.

NPM Node Package Manager.

OSN Organizace spojených národů.

PWA Progressive Web Apps.

REST Representational State Transfer.

SEO Search Engine Optimization.

SPA Single Page Application.

UAAG User Agent Accessibility Guidelines.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

W3C World Wide Web Consortium.

WAI Web Accessibility Initiative.

WCAG Web Content Accessibility Guidelines.

XML Extensible Markup Language.

XSS Cross-site scripting.

Literatura

[1] *Introduction to Web Accessibility* [online]. W3C. [cit. 2019/03/09]. What is web accessibility. Dostupné z: https://www.w3.org/WAI/fundamentals/accessibility-intro/.

[2] *Article 9 – Accessibility* [online]. OSN. [cit. 2019/03/10]. Convention on the Rights of Persons with disabilities. Dostupné z: https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities/article-9-accessibility.html.

[3] *Essential Components of Web Accessibility* [online]. W3C. [cit. 2019/03/20]. Describes what is components of the web accessibility. Dostupné z: https://www.w3.org/WAI/fundamentals/components/.

[4] *Tools and Techniques* [online]. W3C. [cit. 2019/03/23]. Tools for people with disabilities. Dostupné z: https://www.w3.org/WAI/people-use-web/tools-techniques/.

[5] *Types of Assistive Technology* [online]. Barkeley University of California. [cit. 2019/03/23]. Technologies which helps people use computers. Dostupné z: https://webaccess.berkeley.edu/resources/assistive-technology.

[6] *Přístupnost webů státní správy* [online]. Přístupnost.cz. [cit. 2019/03/28]. Zákonná úprava přístupnosti webů státní správy. Dostupné z: http://www.pristupnost.cz/pristupnost-webu-statni-spravy/.

[7] *Metodický pokyn* [online]. Ministerstvo vnitra České republiky, 2010. [cit. 2019/03/28]. Metodický pokyn k vyhlášce č. 64/2008 Sb. Dostupné z: https://www.mvcr.cz/clanek/metodicky-pokyn-k-vyhlasce-c-64-2008-sb-o-forme-uverejnovani_informaci-souvisejicich-s-vykonem-verejne-spravy-prostrednictvim_webovych-stranek-pro-osoby-se-zdravotnim-postizenim-vyhlaska-o_pristupnosti.aspx.

[8] *ISO/IEC 40500* [online]. W3C. [cit. 2019/03/29]. Is ISO/IEC 40500 the same as WCAG 2.0? Dostupné z: https://www.w3.org/WAI/standards-guidelines/wcag/faq/#iso.

[9] *Authoring Tool Accessibility Guidelines* [online]. W3C. [cit. 2019/03/29]. Describes what is defined in ATAG. Dostupné z: https://www.w3.org/WAI/standards-guidelines/atag/.

[10] *Web Content Accessibility Guidelines* [online]. W3C. [cit. 2019/03/29]. Describes what is defined in WCAG. Dostupné z: https://www.w3.org/WAI/standards-guidelines/wcag/.

[11] *User Agent Accessibility Guidelines* [online]. W3C. [cit. 2019/03/29]. Describes what is defined in UAAG. Dostupné z: https://www.w3.org/WAI/standards-guidelines/uaag/.

[12] *Technical Specifications* [online]. W3C. [cit. 2019/03/29]. Technical specifications overview. Dostupné z: https://www.w3.org/WAI/standards-guidelines/#technical-specifications.

[13] *HTML5 accessibility* [online]. Steve Faulkner. [cit. 2019/03/31]. Describes support of HTML5 section tags for accessibility. Dostupné z: https://www.html5accessibility.com/.

[14] *WAI-ARIA Overview* [online]. W3C. [cit. 2019/03/31]. Introduction of WAI-ARIA specification. Dostupné z: https://www.w3.org/WAI/standards-guidelines/aria/.

[15] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/03/31]. Definition of aria-describedby attribute. Dostupné z: https://www.w3.org/TR/wai-aria/#aria-describedby.

[16] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/03/31]. How to manage focus in web applications. Dostupné z: https://www.w3.org/TR/wai-aria/#managingfocus.

[17] *WAI-ARIA Authoring Practises* [online]. W3C. [cit. 2019/03/31]. Guidelines for implementing ARIA. Dostupné z: https://www.w3.org/TR/wai-aria-practices-1.1/.

[18] *Using aria-labelledby* [online]. W3C. [cit. 2019/04/02]. Using aria-labelledby for link purpose. Dostupné z: https://www.w3.org/TR/WCAG20-TECHS/ARIA7.html.

[19] *Using the WAI-ARIA aria-expanded* [online]. W3C. [cit. 2019/04/02]. Describes and shows how to use aria-expanded. Dostupné z: https://www.w3.org/WAI/GL/wiki/Using_the_WAI-ARIA_aria-expanded_state_to_mark_expandable_and_collapsible_regions.

[20] *HTML accesskey attribute* [online]. W3schools. [cit. 2019/04/04]. Definition of keyboard shortcuts accesskey attribute. Dostupné z: https://www.w3schools.com/tags/att_global_accesskey.asp.

[21] *Definition of ARIA landmarks* [online]. W3C. [cit. 2019/04/07]. How to use ARIA landmarks for defining page sections. Dostupné z: https://www.w3.org/TR/2019/NOTE-wai-aria-practices-1.1-20190207/examples/landmarks/.

[22] *Introduction to AJAX* [online]. MDN. [cit. 2019/04/11]. Guides through the AJAX basics. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started.

[23] *ARIA banner role* [online]. W3C. [cit. 2019/04/09]. Describes and shows how to use banner role. Dostupné z: https://www.w3.org/TR/wai-aria-1.1/#banner.

[24] *Fetch API* [online]. MDN. [cit. 2019/04/11]. Introduction Fetch API. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

[25] *ARIA contentinfo role* [online]. W3C. [cit. 2019/04/09]. Describes and shows how to use contentinfo role. Dostupné z: https://www.w3.org/TR/wai-aria-1.1/#contentinfo.

[26] *Introduction to Node.js* [online]. Node.js. [cit. 2019/04/12]. JavaScript runtime environment. Dostupné z: https://nodejs.dev/.

[27] *WCAG Guidline 1.4 - Distinguishable* [online]. W3C. [cit. 2019/04/09]. WCAG definition of content distinguishable. Dostupné z: https://www.w3.org/WAI/WCAG21/quickref/?versions=2.0#distinguishable.

[28] *Single Page Application* [online]. Microsoft. [cit. 2019/04/12]. Modern responsive web application. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.

[29] *Inert subtrees* [online]. WHATWG. [cit. 2019/04/11]. Definition of abstract HTML inert attribute. Dostupné z: https://html.spec.whatwg.org/multipage/interaction.html#inert.

[30] *Angular - Architecture* [online]. Angular. [cit. 2019/04/14]. Angular architecture overview. Dostupné z: https://angular.io/guide/architecture.

[31] *Angular - Components* [online]. Angular. [cit. 2019/04/16]. Introduction to Agnular components. Dostupné z: https://angular.io/guide/architecture-components.

[32] *Angular - Dependency Injection* [online]. Angular. [cit. 2019/04/16]. How to use Dependency Injection in Angular. Dostupné z: https://angular.io/guide/dependency-injection.

[33] *NoSQL databáze* [online]. Marek Rychlý, Dušan Kolář, 2013. [cit. 2019/04/20]. Úvod do NoSQL databází. Dostupné z: http://www.fit.vutbr.cz/~rychly/public/docs/slides-nosql-databases/slides-nosql-databases.print.pdf.

[34] *JSON Web Token* [online]. JWT. [cit. 2019/04/20]. Introduction to JWT. Dostupné z: https://jwt.io/introduction/.

[35] *HTTP status codes* [online]. MDN. [cit. 2019/04/21]. List of HTTP response status codes. Dostupné z: https://jwt.io/introduction/.

[36] *Angular - Routing and Navigation* [online]. Angular. [cit. 2019/04/21]. Overview of navigating in Angular apps. Dostupné z: https://angular.io/guide/router.

[37] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/04/21]. Describes and shows how to use aria-live. Dostupné z: https://www.w3.org/TR/wai-aria/#aria-live.

[38] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/04/21]. Describes and shows how to use button component. Dostupné z: https://www.w3.org/TR/wai-aria-practices-1.1/#menubutton.

[39] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/04/21]. Describes and shows how to use dialog component. Dostupné z: https://www.w3.org/TR/wai-aria-practices-1.1/#dialog_modal.

[40] *Document designMode* [online]. MDN. [cit. 2019/04/23]. Editing text of the DOM document. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document/designMode.

[41] *WAI-ARIA Standard* [online]. W3C. [cit. 2019/04/23]. Describes and shows how to use toolbar component. Dostupné z: https://www.w3.org/TR/wai-aria-practices-1.1/#toolbar.

[42] *Accessibility Reference* [online]. Google. [cit. 2019/04/23]. Accessibility features in Chrome DevTools. Dostupné z: https://developers.google.com/web/tools/chrome-devtools/accessibility/reference.

[43] *axe: Accessibility for Development Teams* [online]. Google. [cit. 2019/04/24]. Accessibility Testing App with axe. Dostupné z: https://www.deque.com/axe/.

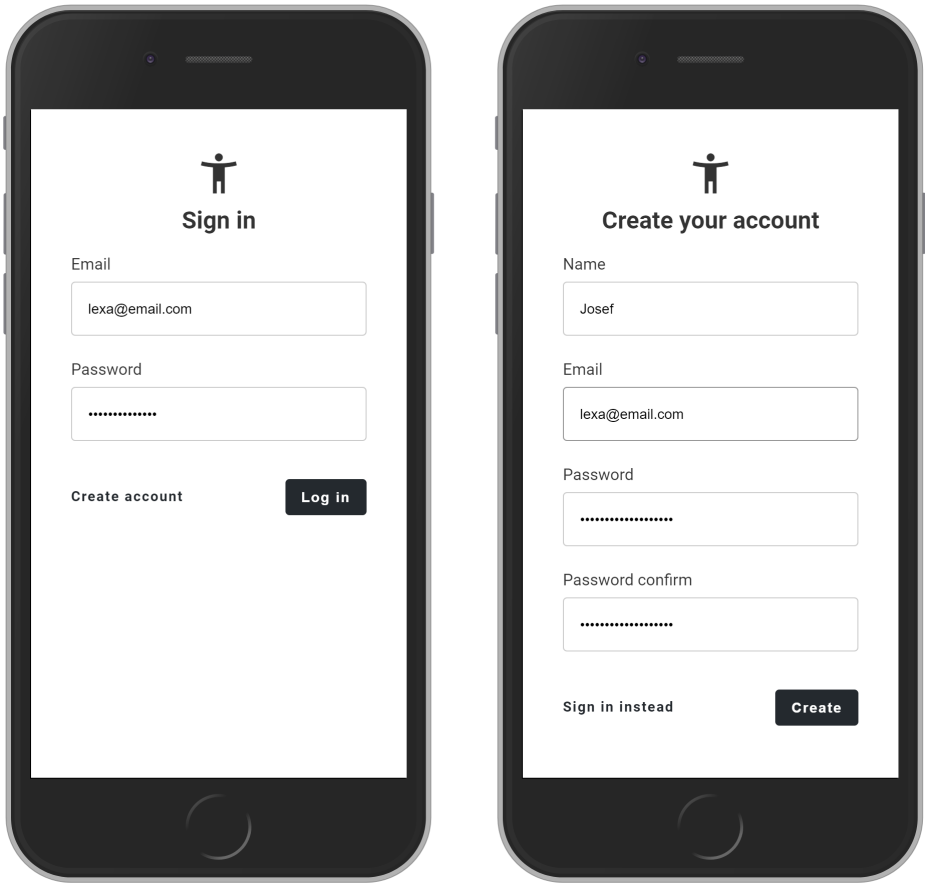
A Přílohy

A.1 Uživatelská příručka

Uživatelská příručka popisuje obsluhu hlavních částí aplikace. Příručka také obsahuje snímky obrazovky při plném a mobilním zobrazení, čímž přináší vhled do vytvořené aplikace.

A.1.1 Přihlášení a registrace

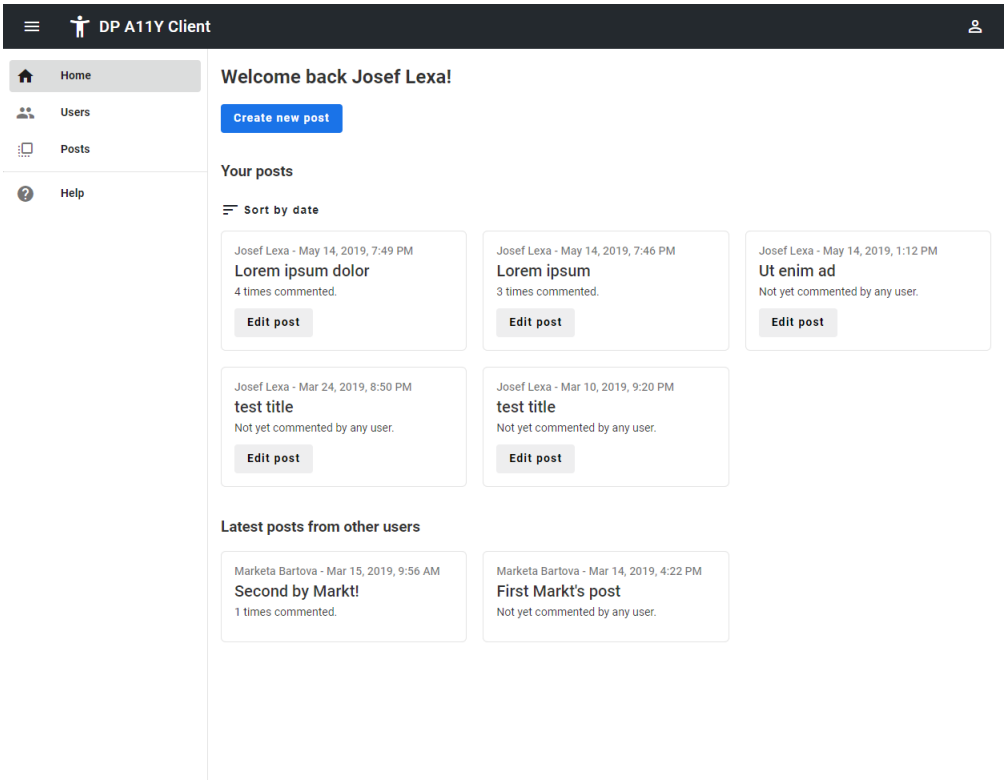
Při prvním spuštění je uživateli zobrazeno přihlášení do aplikace. V případě nově příchozího uživatele je nejprve vyžadována registrace. Oba formuláře jsou pro všechna zařízení totožné. Jejich podoba je vidět na obrázku A.1.



Obrázek A.1: Přihlášení a registrace do aplikace

A.1.2 Hlavní obrazovka

Po přihlášení je uživateli zobrazena hlavní obrazovka. V horní části se nachází lišta s tlačítkem pro skrývání navigace a tlačítkem pro uživatelskou nabídku. Na levé straně se nachází samotná navigace. Na hlavní obrazovce jsou zobrazeny dvě kategorie příspěvků. První kategorií jsou příspěvky uživatele a druhou kategorií nejnovější příspěvky ostatních uživatelů. Kromě výpisu příspěvků je z hlavní obrazovky možno vytvořit nový příspěvek. Tím se aplikace přesměruje na editor viz kapitola A.1.3. Vzhled hlavní obrazovky je na obrázku A.2.

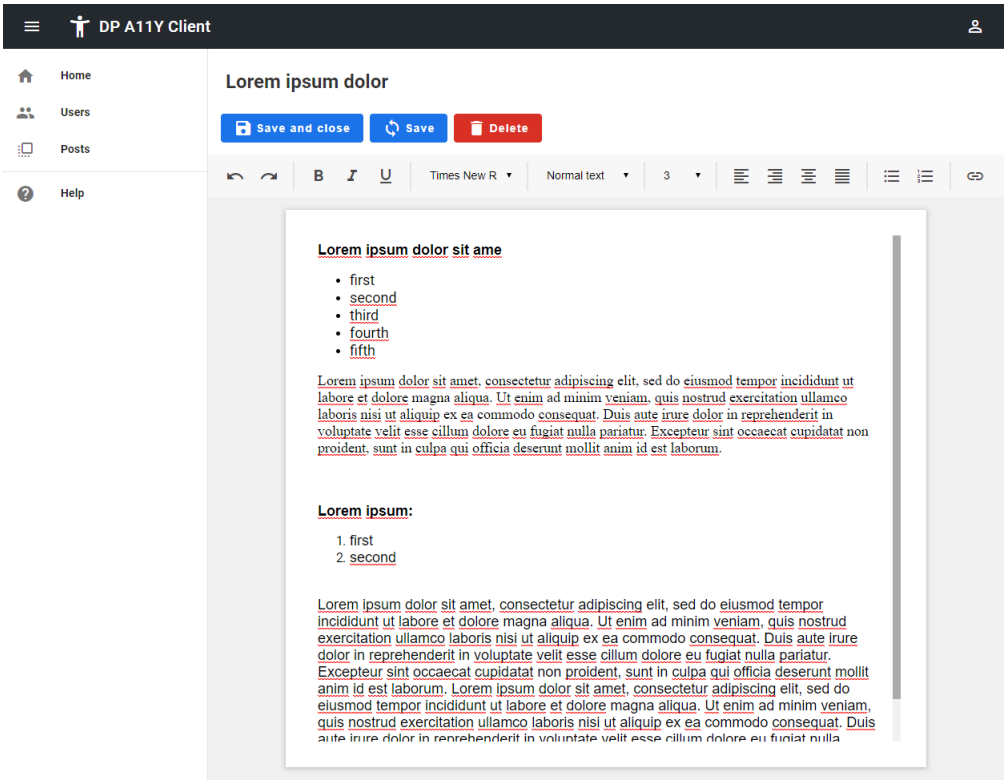


Obrázek A.2: Hlavní obrazovka aplikace

A.1.3 Editor

Editor slouží v aplikaci pro vytvoření a úpravu uživatelských příspěvků. V horní části se nachází editovatelný titulek příspěvku. Pod ním tlačítka pro uložení a smazání příspěvku. Editor umožňuje základní editaci textu včetně vytvoření odrážkového a číslovaného seznamu. Pro úpravu textu je také možné využívat klávesové zkratky, které jsou v aplikaci popsány v sekci

nápovědy. Podoba editoru je vidět na obrázku A.3. Responzivní vzhled editoru pro mobilní zařízení je na obrázku A.4.



Obrázek A.3: Úprava příspěvku

A.1.4 Další části aplikace

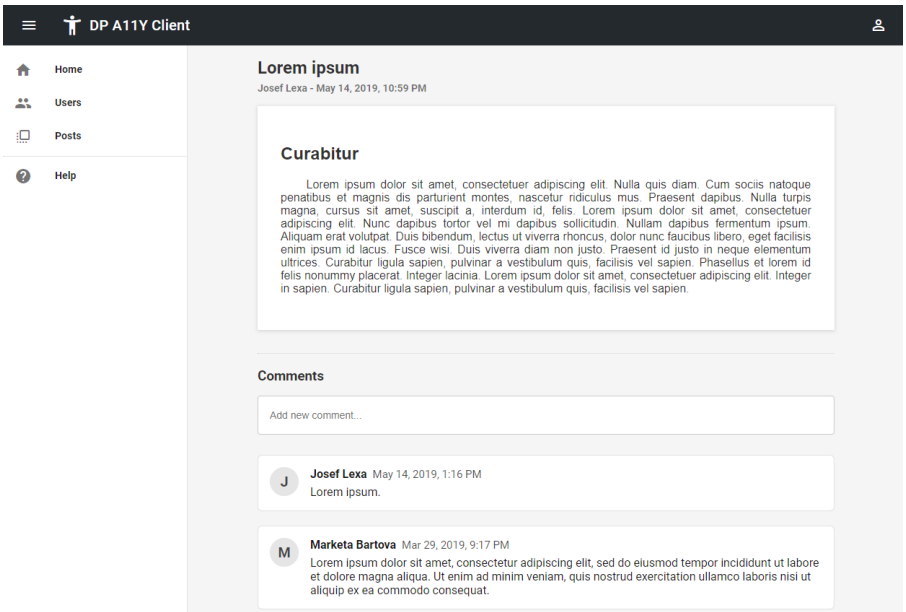
Aplikace dále obsahuje sekci s výpisem všech uživatelů a příspěvků, na které se uživatel dostane pomocí odkazů v navigaci.

V sekci příspěvků se nachází všechny příspěvky ostatních uživatelů. Po vybrání jednoho z nich je uživatel schopen zobrazit jeho obsah a případně k němu vytvořit komentář. Detail příspěvku je vidět na obrázku A.5.

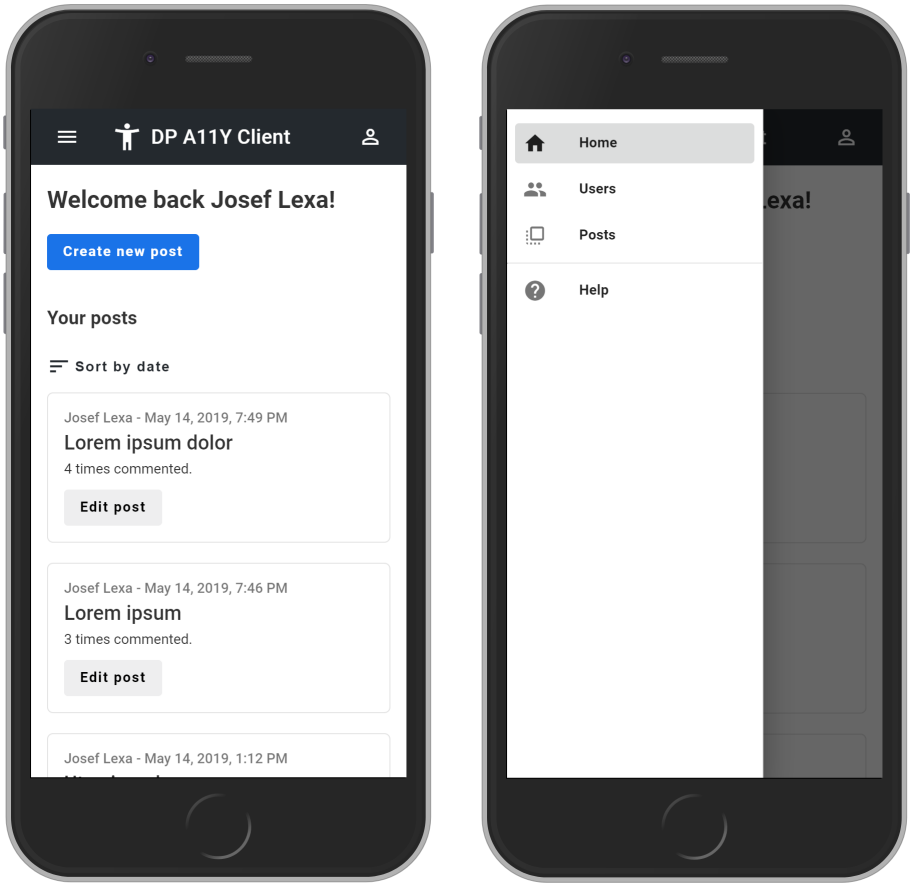
Výpis uživatelů slouží k procházení uživatelů aplikace. Po vybrání jednoho z nich je možné se dostat na jeho detail, na kterém jsou zobrazeny všechny jeho příspěvky, podobně jako je tomu na hlavní obrazovce. Po kliku na některý z příspěvků se opět zobrazí jeho detail.

Veškerá funkcionalita je přístupná také z mobilních zařízení resp. jiných zařízení s nižším rozlišením. Některé náhledy aplikace na těchto zařízeních jsou na obrázku A.6.

Obrázek A.4: Úprava příspěvku na mobilních zařízeních



Obrázek A.5: Detail příspěvku



Obrázek A.6: Další náhledy aplikace na mobilních zařízeních