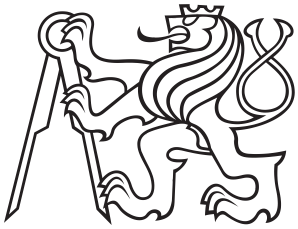
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

UI toolkit přístupných webových komponent

Pavel Sušický

Vedoucí práce: Bc. Petr Hurťák
Studijní program: Softwarové inženýrství a technologie
Zaměření: Programátor / architekt webových aplikací
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sušický** Jméno: **Pavel** Osobní číslo: **499355**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

UI toolkit přístupných webových komponent

Název bakalářské práce anglicky:

UI toolkit of accessible web components

Pokyny pro vypracování:

Webový vývoj dominuje komponentové paradigma. Vývojaři využívají komponent pro tvorbu uživatelských prostředí. Cílem práce je vytvořit sadu přístupných, znovupoužitelných komponent splňující standardy definované iniciativou pro webovou přístupnost v rámci World Wide Web konsorcia (dále jen *„WAI“*) ve vybrané JavaScriptové knihovně Solid.js.
1. Seznamte se s pojmy přístupnost, znovupoužitelnost v kontextu webových aplikací.
2. Seznamte se s technickou specifikací WAI-ARIA a přístupy (WCAG doporučení, ARIA APG návrhové vzory) týkající se přístupnosti na webu.
3. Proveďte analýzu existujících řešení a stav ekosystému ve vybrané knihovně.
4. Pomocí přístupů z bodu 2 naimplementujte vybrané komponenty splňující kritéria přístupnosti.
5. Vytvořte automatizované testy kontrolující kritéria přístupnosti komponent podle přístupů z bodu 2.
6. Vytvořte jednoduché rozhraní pomocí vytvořených komponent a ukažte, že jsou funkční.
7. Vytvořte webovou stránku dokumentující použití daných komponent.

Seznam doporučené literatury:

1. World Wide Web Consortium. “Accessible Rich Internet Applications (WAI-ARIA) 1.2” publikováno 6. června 2023. [https://w3.org/TR/wai-aria-1.2](https://www.w3.org/TR/wai-aria-1.2/).
2. World Wide Web Consortium. “Web Content Accessibility Guidelines (WCAG) 2.2” publikováno 5. října 2023. [https://w3.org/TR/WCAG22](https://www.w3.org/TR/WCAG22/)
3. World Wide Web Consortium. “ARIA Authoring Practices Guide (APG)” [https://w3.org/WAI/ARIA/apg](https://www.w3.org/WAI/ARIA/apg/)

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Bc. Petr Huťťák katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **15.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Bc. Petr Huťťák podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Chtěl bych zde poděkovat svému vedoucímu práce Bc. Petrovi Huťákovi za cenné rady a pomoc při tvorbě této práce.

Dále bych chtěl vyjádřit obrovské poděkování mé rodině a to hlavně mé mamince Pavlíně za veškerou podporu při mém studiu.

Stejně velký vděk patří i mému tatínkovi Jiřímu za jeho cenné rady a zkušenosti, které mi pomáhají v dennodenním životě.

V neposlední řadě i mým bratrům Honzovi a Kubovi, kteří si pro mě vždy našli čas a pomohli mi s čímkoliv, co jsem potřeboval.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2024

Pavel Sušický

Abstrakt

Bakalářská práce se zabývá problematikou znovupoužitelných komponent v kontextu tvorby rozhraní pro webové stránky a aplikace. Cílem práce je implementovat komponenty z kolekce návrhových vzorů publikované informativním zdrojem Aria Authoring Practises ve frameworku Solid.js. Součástí práce je i analýza existujících řešení napříč vývojářským ekosystémem a testování implementovaných komponent. Výsledkem práce jsou komponenty a dokumentace pro jejich použití, tyto komponenty jsou distribuovány v rámci ekosystému pro další vývojáře, kteří je mohou využít pro své projekty.

Klíčová slova: Web komponenty, přístupnost, WAI-ARIA, WCAG, APG, Solid.js

Vedoucí práce: Bc. Petr Huťák

Abstract

The bachelor thesis deals with the issue of reusable components in the context of UI creation for websites and web applications. The thesis aims to implement components from a collection of design patterns published by the informative resource Aria Authoring Practices in the Solid.js framework. The work includes analysis of existing solutions across the JavaScript ecosystem and testing the implemented components. The work yields components and documentation describing their use. These components are distributed within the ecosystem for other developers to use in their projects.

Keywords: Web components, accessibility, WAI-ARIA, WCAG, APG, Solid.js

Title translation: UI toolkit of accessible web components

Obsah

Ukázky kódu

Obrázky

Tabulky

Slovník

APG

ARIA Authoring Practices Guide. 1, 10, 13, 21, 28, 29, 30, 33, 55, 59, 61

API

Application Programming Interface. 19, 41

CDN

Content Delivery Network. 55

CLI

Command Line Interface. 18, 26, 55

CSR

Client-side Rendering. 20

DOM

Document Object Model. 15, 49, 54

E2E

End-to-End. 60, 61

HTA

Hierarchical Task Analysis. 32

HTML

HyperText Markup Language. 6, 8, 15, 17, 27, 30, 43

Hydratace

V kontextu webových aplikací hydratace je proces synchronizace předem vyrenderovaného HTML a klientského JavaScriptu tak, aby byla aplikace interaktivní. 55

IDE

Integrated Development Environment. 32

JSX

JSX je syntaxe pro psaní HTML syntaxe v JavaScriptu. 15, 20

SSG

Static Site Generator. 20, 55

SSR

Server-side Rendering. 20

TodoMVC

TodoMVC je kolekce příkladové aplikace implementované v různých webových technologiích. 15

UI

User Interface. 18, 19

VDOM

Virtual DOM. 15

W3C

World Wide Web Consortium. 6

WAI

Web Accessibility Initiative. 1, 6, 9

WAI-ARIA

Web Accessibility Initiative — Accessible Rich Internet Applications. 1, 6, 7, 8, 10, 13, 30, 59, 60

WCAG

Web Content Accessibility Guidelines. 1, 8, 9, 11

Kapitola 1

Úvod

Přístupnost na webu je v dnešní době důležitá. Tato práce se zabývá implementací JavaScriptové knihovny znovupoužitelných a přístupných komponent, které jsou implementované podle moderních specifikací publikovaných konsorciem W3C.

V teoretické části se rozebírá problematika přístupnosti na webu, kde se popisuje účel existence **Web Accessibility Initiative (WAI)**. Součástí této kapitoly je analýza technických specifikací a doporučení pro tvorbu přístupného webového obsahu pod názvy **Web Accessibility Initiative — Accessible Rich Internet Applications (WAI-ARIA)**, obecných doporučení pro obsah na webu **Web Content Accessibility Guidelines (WCAG)** a doporučení pro tvorbu přístupných komponent **ARIA Authoring Practices Guide (APG)**.

Praktická část se věnuje návrhu, implementaci a testování knihovny komponent, která je přístupná. Komponenty jsou tedy implementované podle standardů a doporučení zmíněných výše.

1.1 Motivace

V současné době jsou znovupoužitelné komponenty nejčastějším způsobem jakým se vytváří webové aplikace, protože nejpoužívanější knihovny využívají komponentové paradigma [1, 2, 3, 4].

Vznik komponent tak vedl k vytvoření knihoven komponent obsahující základní prvky, které denně potkáváme na internetu, aby je vývojář nemusel znovu implementovat pro každou aplikaci zvlášť.

Vývoj znovupoužitelných komponent je však náročný, protože kromě implementačních detailů je nutné se zabývat i přístupové problematice.

Toto vedlo k nekonzistenci ekosystémů, kde hojně využívané knihovny mají

k dispozici knihovny se základními i robustními, přístupnými komponenty na webu. Oproti tomu menší knihovny, které se teprve začínají dostávat do podvědomí vývojářům, mají k dispozici zásadně menší ekosystém komponent, který není zdaleka tak robustní.

1.2 Cíl práce

Hlavním výsledkem této práce by měla být knihovna, která rozšíří ekosystém frameworku Solid.js o znovupoužitelné a přístupné komponenty implementované dle návrhových vzorů APG.

Vývojáři budou schopni využít tyto komponenty ve vlastních projektech a ušetří jim tak čas strávený na vývoji přístupných webových aplikací.

Součástí výstupu je i webová stránka dokumentující použití komponent, vygenerovaná dokumentace pomocí typedocu, automatizované testy a příkladová aplikace využívající dané komponenty.

Část I

Relevantní technologie

Kapitola 2

Přístupnost na webu

Tato kapitola se věnuje představení problematice přístupnosti na webu.

2.1 Přístupný web

Přístupnost na webu je definována jako schopnost webového obsahu být interpretován a používán nejširším možným spektrem uživatelů bez ohledu na jejich schopnosti, nebo fyzický stav [5].

2.1.1 Kategorizace postižení

Moderní internet je plný různorodého multimediálního obsahu a každý má svá specifika, které ho můžou dělat nepřístupným, nebo i přístupným pro určitou skupinu uživatelů. Mezi nejčastější a prozkoumané kategorie patří [6]:

- **Vizuální** — Barvoslepost, zhoršené vidění či úplná slepota. Je důležité udržovat pravidla správného kontrastu barev a správné velikosti písma, nebo minimálně umožnit uživateli tyto hodnoty změnit.
- **Sluchové** — Částečně zhoršené slyšení, nebo úplná hluchota v jednom či obou uších. Média, která využívají zvuk pro přenos informací je potřeba doplnit o možnosti změny hlasitosti, nebo doplnění o textový přepis či titulky.
- **Motorické** — Chybějící končetina, třes, zhoršená koordinace a paralýza. Pro uživatele s motorickými postiženími je důležité mít přístupné ovládání pomocí klávesnice. Uživatelé využívají alternativních zařízení pro manipulaci s obsahem webu jako jsou ergonomické příslušenství, ústní myš, trackbally a další. Přístupnost přes klávesnici velice často pomáhá zmíněným zařízením fungovat dle potřeby uživatele.

- **Kognitivní** — Kognitivní potíže mohou zasáhnout některou část nervové soustavy člověka. Ať už je to vnímaní informací, paměť, učení či porozumění. V kontextu webu je důležité informace prezentovat nejjednodušším, jasným způsobem a nechat uživatelům dostatek času na pochopení a reakci.

Mezi další kategorie postižení patří například poruchy řeči, ale v kontextu této práce se jimi nebudu zabývat, neboť nejsou relevantní pro tvorbu znovupoužitelných komponent.

2.2 W3C a WAI

World Wide Web Consortium (W3C)[1] je mezinárodní konsorcium, které vyvíjí standardy pro web. WAI je jedním z projektů W3C, který se zaměřuje na přístupnost webu.

WAI definuje specifikace a doporučení pro mnohé aspekty přístupnosti na webu, od *user agentů*, evaluačních nástrojů až po nástroje pro tvorbu digitálního obsahu.

2.3 WAI-ARIA

WAI-ARIA[2] je technická specifikace, která rozšiřuje webové technologie **HyperText Markup Language (HTML)** a JavaScript o ontologii atributů představující role, vlastnosti a stavy elementů [7].

Zmíněné atributy umožňují asistivním technologiím (například odečítače obrazovky) předat uživatelům význam a podrobnější informace o elementech, které se nachází na daném webu a tím jim pomoci s pochopením a manipulací obsahu stránky.

2.3.1 ARIA role

Role slouží k přidání sémantického významu HTML elementům tak, aby nevidomí či uživatelé s jinou formou postižení dokázali pochopit účel daného elementu.

Některé HTML elementy mohou mít svůj sémantický význam i bez použití rolí [8]. Například *button* element se používá pro interaktivní tlačítka a je vhodnější jej použít namísto nesémantického divu s rolí button.

[1] `https://w3.org`
[2] `https://w3.org/TR/wai-aria`

```
1  <ul role="menu">
2      <li role="menuitem">Open file</li>
3      <li role="menuitem">Save file</li>
4  </ul>
```

Ukázka kódu 2.1: ARIA role

V ukázce 2.1 na řádku 1 můžeme vidět použití role menu na elementu pro neuspořádaný seznam. Na řádku 2 a 3 máme dva prvky s rolí *menuitem*, které reprezentují položky v menu. Každá role kromě sémantického významu s sebou nese i vlastnosti, které musí být dodrženy podle specifikace. Například výše zmíněné menu musí obsahovat minimálně jeden prvek s rolí menuitem (alternativně *menuitemcheckbox* nebo *menuitemradio*) [9, 10].

Pro tvorbu moderních webových komponent je vhodné preferovat sémantické elementy, které mají svůj význam i bez použití explicitních rolí. Explicitně definované role jsou potřeba pokud informace o elementu nejsou dostatečné pro pochopení jeho účelu.

V kontextu práce budu používat především role z kategorie widget podle WAI-ARIA specifikace, protože jsou vhodné pro komponentový vývoj. Existují však i další kategorie rolí určené pro strukturu webu nebo orientační body [11].

2.3.2 ARIA stavy

Stavy slouží pro upozornění uživatele na změny ve stavu stránky nebo komponenty. Společně s rolemi tvoří základní dvojici pro přístupnost na webu. Je důležité poznamenat, že některé stavy nejdou použít pro všechny role, ale jsou vázané ke specifickým účelům a rolím.

```
1  <button aria-pressed="false"> <!-- true|false|mixed -->
2      Send order
3  </button>
```

Ukázka kódu 2.2: ARIA stavové atributy

V ukázce 2.2 na řádku 1 můžeme vidět použití atributu *aria-pressed*. Tento

atribut slouží jenom pro elementy s rolí *button*. Řádek 1 nám zároveň ukazuje, že hodnota atributu je *false*, ale může nabývat i hodnot *true* nebo *mixed*. Button s takovým atributem představuje tzv. *toggle button* [12]. Tedy tlačítko, které si drží svůj vlastní stav zakliknutí. Můžeme využít JavaScript pro dynamické změny atributu na reakci uživatelova vstupu.

2.3.3 ARIA vlastnosti

Vlastnosti jsou podobné stavům. Jedná se o HTML atributy jejichž primárním účelem je dodání více informací o daném elementu. Rozdíl oproti stavům vychází z četnosti změn, kde u stavů se předpokládá častá změna v životním cyklu stránky, která vede na upozornění uživatele. Oproti tomu vlastnosti se mění jenom zřídka [7, sekce 6.1].

```
1  <label for="username">Username</label>
2  <input id="username" aria-describedby="username-error" />
3  <p id="username-error">Username is required</p>
```

Ukázka kódu 2.3: ARIA vlastnosti

V ukázce 2.3 na řádku 2 můžeme vidět použití vlastnosti *aria-describedby*. Tento atribut slouží k přidání dodatečnému popisu daného *input* elementu. Hodnota atributu je id elementu, který obsahuje textový popis daného *inputu*.

2.4 WCAG

WCAG[3] je mezinárodní, metodický soubor pravidel (úspěchových kritérií) pro tvorbu přístupných webů. Na rozdíl od technické specifikace WAI-ARIA se WCAG zaměřuje na pravidla, jak by se stránky měly chovat, aby byly pro uživatele co nejpřívětivější na používání. Nejedná se tak o technickou specifikaci, která by tvořila nové mechanismy a technologie pro přístupnost. Všechny doporučení jsou dosažitelné pomocí existujících WAI-ARIA atributů a JavaScriptu.

[3] `https://w3.org/TR/WCAG21`

2.4.1 Verze WCAG

Historicky je WCAG zde už dlouho, první verze vyšla již v roce 1999. Postupem času se metodika upravovala, aby vyhověla moderním technologiím a potřebám uživatelů. Nejdůležitější v kontextu této práce bude poslední verze 2.2, která vyšla 5. října 2023. Je důležité poznamenat, že WCAG je zpětně kompatibilní, tedy pokud web splňuje všechna kritéria verze 2.2, tak automaticky splňuje všechna kritéria z předchozích verzí [13].

2.4.2 Úrovně přístupnosti

Každé úspěchové kritérium má přiřazenou úroveň, která rozlišuje jeho důležitost pro uživatele s postižením. Pokud web splňuje všechna kritéria dané úrovně, tak se označuje jako přístupný pro specifikovaný okruh lidí. WCAG definuje tři úrovně přístupnosti [13, Sekce 5.2.1]:

1. **Úroveň A** — Jedná se o minimální úroveň zajišťují základní ovladatelnost a funkcionalitu webu pro uživatele s postižením.

2. **Úroveň AA** — Obsahuje všechna kritéria úrovně A

3. **Úroveň AAA** — Obsahuje všechna kritéria úrovně A a AA

Weby úrovně A obsahují nezbytnou funkcionalitu a přístupnost pro základní používání. Všechny webové stránky i aplikace by měly splňovat alespoň tuto úroveň, ale v praxi WCAG doporučuje alespoň úroveň AA. Rozdíl mezi úrovněmi jsou dohledatelné na referenční stránce provozované WAI [14]. Obecně se dá říci, že každá úroveň přináší více funkcí pro uživatele a jednoduchost používání. Nejzákladnější funkce jako je ovládání klávesnicí, přístupnost formulářových prvků, ukazatele chyb a další jsou v úrovni A, protože to jsou nejčastější problémy se kterými se může uživatel setkat. Každá úroveň přináší kritéria, která jsou těžší na splnění, ale také se méně často vyskytují, protože existují alternativní varianty implementace stejné funkcionality.

Úroveň AAA je nejvyšší možná úroveň přístupnosti, ale v praxi složitá na splnění, proto WCAG doporučuje se k této úrovni alespoň přiblížit.

2.5 APG

APG[4] je soubor nejčastějších návrhových vzorů a *widgetů*[5] vyskytujících se na webu. Každý widget má svoji vlastní stránku s popisem, účelem, klávesovou navigací a používanými atributy, rolemi a stavy z WAI-ARIA. Kromě specifikace widgetů obsahuje i další praktická doporučení mezi která patří [15]:

- Používání orientačních bodů na stránce
- Používání přístupných názvu a popisků pro elementy
- Používání strukturovaných rolí

APG je praktickým zdrojem pro tvorbu přístupných komponent, protože jsou zde popsány důležité informace jako použité role, vlastnosti, stavy z WAI-ARIA. Jsou zde i doporučení pro klávesovou navigaci a obecné informace o tom, jak by komponenta měla být použita.

2.6 Základní principy přístupnosti

Tato sekce popisuje základní principy přístupnosti, které jsou relevantní v kontextu této práce, tedy vytváření znovupoužitelných komponent. Existují avšak i další principy, které se obecně zaměřují na webový obsah, porozumění textu anebo přístupnost audiovizuálního obsahu [16].

2.6.1 Ovládání pomocí klávesnice

Myš je základní výbavou každého uživatele na internetu, ale kromě myši máme i další periférie. Klávesnice je jedna z nejdůležitějších, ale i mnohdy opomíjených periférií v kontextu přístupnosti.

Klíčovým prvkem přístupnosti pomocí klávesnice je *focus*, tedy kde se uživatel zrovna nachází na dané stránce. Společně s odečítači obrazovky tvoří základní výbavu pro nevidomé uživatele, kteří si za jejich pomocí dokáží přečíst a používat obsah na webu.

Přístupný web by měl splňovat základní pravidla pro optimální ovládání klávesnicí [17]:

[4] `https://w3.org/WAI/ARIA/apg`
[5] V kontextu této práce je widget to samé jako komponenta

- Všechny funkcionality a interakce dostupné pomocí počítačové myši jsou dostupné i skrze klávesnici.
- Všechny funkcionality a interakce nevyžadují stisk kláves v definovaném pořadí pokud to nevyžaduje povaha vstupu.
- Nedochází k uváznutí focusu v jakékoliv sekci stránky.

2.6.2 Ovládání pomocí dotyku

Dalším typickým způsobem ovládání je dotykový displej na moderních zařízení typu mobil, tablet, chytré hodinky, kiosky či jiné vestavěné systémy. Ovládání za pomocí dotyku je rozmanité, uživatel může používat jednoduché dotyky, ale i složitější gesta pro interakci.

Dle přednášky Iva Malého z kurzu principy mobilních aplikací rozdělujeme gesta dle počtu použitých prstů, charakteristiky pohybu a použitých technologií [18]. WCAG definuje gesta více abstraktně na tzv. *path-based* (kromě koncové pozice prstu záleží i na přechodných pozicích) a *multipoint* (gesto vyžadující více prstů) [19].

Mezi základní pravidla pro optimalizaci interakce za pomocí dotyku patří:

- Gesta nevyžadují přesnost, nebo je k dispozici alternativní způsob ovládání [19].
- Interaktivní prvky jsou dostatečně velké pro dotyk prstem [20].

2.6.3 Ostatní způsoby interakce

Kromě myši a klávesnice existují další způsoby interakce mezi které patří například hlasové ovládání, ovládání pomocí pohybu, ovládání pomocí očí a další. Výčet důležitých pravidel pro optimalizaci interakce za pomocí těchto způsobů [16]:

- Popisky interaktivních prvků jsou řádně propojeny v kódu pro korektní čtení pomocí odečítačů obrazovky a hlasovému zadávání.
- Uživatelé mají možnost pozastavit, prodloužit, nebo upravit časové limity pro interakci.

2.6.4 Dostatek času na interakci

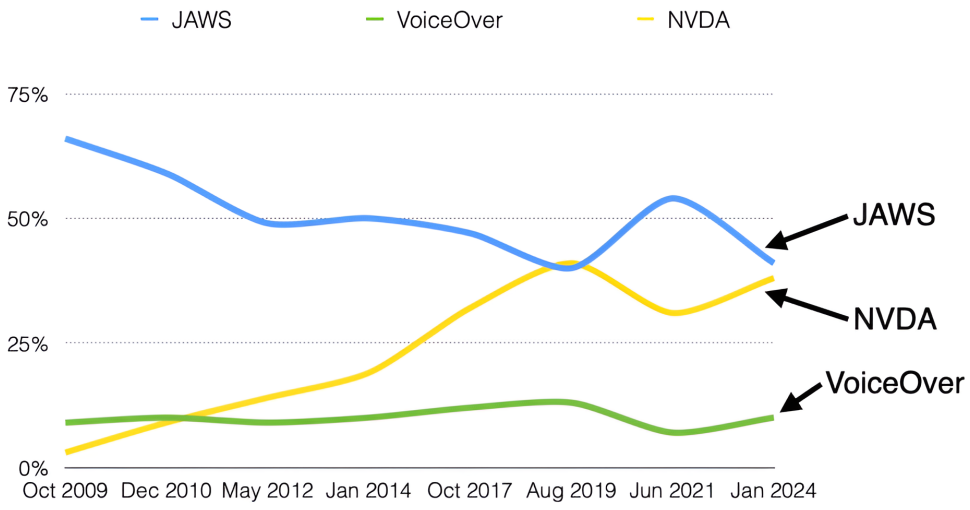
Dalším klíčovým aspektem přístupných komponent je dát uživatelům dostatek času na interakci s danou komponentou. Příliš rychlé uzavírání, mizení, schování interaktivních prvků komponent může vést k frustraci či znemožnění použití důležité funkcionality pro uživatele [16].

2.7 Odečítače obrazovky

Odečítač obrazovky je software, který převádí textový a grafický obsah na zvukový výstup. Nejčastěji je používaný nevidomými uživateli, kteří tak mohou interpretovat obsah na obrazovce zařízení.

Mezi nejpoužívanější odečítače obrazovky patří [21]:

1. JAWS[6] na Windows (40.5%)
2. NVDA[7] na Windows (37.7%)
3. VoiceOver[8] na MacOS (9.7%)
4. Other (12.1%)



Obrázek 2.1: Trend používání populárních odečítačů obrazovky [21]

[6] https://freedomscientific.com/products/software/jaws
[7] https://nvaccess.org
[8] https://en.wikipedia.org/wiki/VoiceOver

V rámci práce budu testovat komponenty s VoiceOver, protože s tímto odečítačem obrazovky mám nejvíce zkušeností a z rozsahu práce a časové dotace je testování všech populárních odečítačů nemožné.

Důležitá poznámka k testování komponent pomocí odečítačů obrazovky je, že tyto testy zatím nejdou plně automatizovat. Je důležitá interpretace výstupu odečítače obrazovky člověkem, protože podobně jako ve vytváření uživatelských rozhraní je důležité jak uživatel chápe výstup aplikace.

2.8 Závěr kapitoly

Tato kapitola popsala principy přístupnosti na webu a její základní mechanismy. Výčet důležitých poznatků pro další části práce je následující:

- Vlastnosti, stavy a role z WAI-ARIA jsou důležitou součástí přístupných rozhraní.
- Návrhové vzory z APG jsou praktickým zdrojem pro tvorbu přístupných komponent.
- Je důležité neopomíjet alternativní způsoby ovládání.
- Testování čteček obrazovky bude provedeno pomocí VoiceOver na MacOS.

Kapitola 3

Technologie

Tato kapitola představuje technologie a přístupy použité v rámci této práce.

3.1 Solid.js

`Solid.js`[1] je JavaScriptová knihovna pro tvorbu uživatelských rozhraní obdobně jako React, Vue, nebo Svelte. Podobně jako React používá JSX pro vytváření komponent a HTML elementů.

3.1.1 Rozdíly

Rozdíl mezi Solid.js a ostatními knihovnami spočívá v technologii synchronizování změn v **Document Object Model (DOM)**.

Populární knihovny jako React, nebo Vue používají tzv. **Virtual DOM (VDOM)**, což je virtuální reprezentace DOM stromu v paměti, která se porovnává s reálným DOM stromem a rozdíly mezi nimi se aplikují pomocí *diffing* algoritmu.

Solid.js a nově i Svelte používají granulární, reaktivní modely pro sledování změn v DOM. Prakticky Solid.js využívá principu *dependency tracking*, kde přístupy k reaktivním proměnným jsou sledovány skrze *subscribers*, kteří jsou upozornění na případné změny respektive zápisy do těchto proměnných [22]. Tento princip připomíná designový vzor *observer*.

3.1.2 Rychlost

Tabulka 3.1 porovnává Solid.js s vybranými *frameworky*. Všechny metriky jsou vypočítané na příkladové aplikaci `TodoMVC`[2] implementované v daném

[1] `https://solidjs.com`
[2] `https://todomvc.com`

frameworku, aby test byl spravedlivý. Každé skóre je číslo normalizované do intervalu $[1, \infty)$, které nám říká velikost zhoršení oproti nejlepší implementaci [23, 24].

První metrika "Operations" je vážený geometrický průměr skóre všech operací nad příkladovou aplikací (např. vytvoření řádku, smazání řádku, vybraní řádku a další...).

Druhá metrika "Transferred size" je vážený geometrický průměr skóre přenesených dat po síti při použití daného frameworku.

Poslední metrika "Memory allocation" je vážený geometrický průměr skóre využití paměti.

Metrika	Solid 1.8.0	Svelte 5	Vue 3.3.6	React 18.2.0
Operations	**1.08**	1.08	1.24	1.53
Transferred size	**2.29**	3.04	7.16	15.32
Memory allocation	**1.45**	1.52	2.13	2.81

Tabulka 3.1: Porovnání rychlosti Solid.js s populárními frameworky.
Zdroj: Data převzata od Stefana Krause [23, 24].

Z výsledku *benchmarku* můžeme vidět, že Solid.js obdobně jako Svelte je velice blízko 1, tedy je jenom o 8% pomalejší než nejrychlejší implementace. Svelte od verze 5 má obdobný styl reaktivního modelu jako Solid.js, což vysvětluje podobné výsledky [25].

3.1.3 Základní primitiva

Solid.js obsahuje několik základních primitiv (funkcí), které umožňují vytvořit reaktivní chování v komponentách.

- **createSignal**[3] — Vrací *tuple* obsahující reaktivní proměnnou a její *setter* funkci.
- **createEffect**[4] — Zavolá callback funkci při každé změně reaktivní proměnné v daném *scope*.
- **createMemo**[5] — Vytvoří memoizovanou odvozenou hodnotu.

Často je potřebné sledovat *lifecycle* události komponenty, pro tyto účely slouží funkce `onMount` a `onCleanup`.

[3] `https://docs.solidjs.com/reference/basic-reactivity/create-signal`
[4] `https://docs.solidjs.com/reference/basic-reactivity/create-effect`
[5] `https://docs.solidjs.com/reference/basic-reactivity/create-memo`

3.2 TypeScript

JavaScript je dynamicky typovaný jazyk, což se v praxi ukázalo jako těžkopádné při škálování velkých projektů. Postupem času vznikly různé nástroje a nadstavby nad samotným jazykem, které přidávají statické typování. Nástroje jako `TypeScript`[6], `Flow`[7], nebo `ReasonML`[8] se ukázaly jako vhodným řešením pro škálovatelnost, znovupoužitelnost a přehlednost kódu. Pomyslným vítězem se v posledních letech ukázal TypeScript, který má vysokou popularitu i podporu v rámci ekosystému, proto není náhodou, že ho ve své práci plně používám.

TypeScript dodává znovupoužitelným komponentám větší přehlednost při zpětném čtení kódu, ale i konzumenti komponent mají k dispozici robustní typovou kontrolu včetně fungujícího automatického doplňování kódu, což je důležitá vlastnost pro splnění nefunkčního požadavku `NFR 1.3`.

3.3 Formát komponent

- **Knihovna komponent** je taková knihovna komponent, která už má předepsané styly. Většinou je možnost přepsat většinu stylů pomocí "themes", ale HTML kód komponent je téměř vždy neměnitelný. Hodně se stává, že vývojáři potřebují odlišnou HTML strukturu než je dána. Častým případem je právě řešení přístupnosti, kdy je potřeba přidat elementy navíc do komponenty pro např. správné fungování čteček obrazovky.

- **Knihovna headless komponent** se liší od klasické knihovny komponent tím, že neobsahuje předepsané styly. Zde můžeme rozlišit několik úrovní *headless* komponent.

  - Exportující jednotlivé prvky komponenty, konzument tak má větší kontrolu nad výsledným kódem komponenty. Ovšem HTML struktura v jednotlivých prvcích komponent není stále plně flexibilní.
  - Exportující pouze logiku v podobě "primitiv". V Reactu se jedná o "custom hooks", ve Vue "composition utilities", ve Svelte to jsou prosté funkce, které pracují s reaktivními proměnnými a obdobně i ve Solid.js.

[6] `https://typescriptlang.org`
[7] `https://flow.org`
[8] `https://reasonml.github.io`

- Hybridní knihovna, která obsahuje i styly, ale je zde mechanismus jak tyto styly odebrat či přepsat. Příkladem může být populární knihovna `Mantine`[9].

- **Copy and paste** není klasická knihovna. Tato varianta vznikla jako reakce na headless komponenty a primitiva. Jedná se o formát komponent, kde konzumenti si zkopírují předpis komponenty. Využívá tak existujících headless komponent, nebo **User Interface (UI)** primitiv. Tato varianta je velmi flexibilní, protože umožňuje distributorům přidávat vzorové styly, které konzument může smazat, nebo vyměnit za vlastní řešení. Zároveň má konzument plnou kontrolu nad kódem komponenty. Nevýhodou je horší znovupoužitelnost a zároveň nutnost manuální aktualizace komponenty pokud se změní její logika. Distribuce takových komponent je většinou v podobě **Command Line Interface (CLI)**, nebo zkopírováním z dokumentace.

```
1  import { Tab } from "headless-component-library"
2
3  export const Toolbar = () => {
4    return (
5      <Tab.Group>
6        <Tab.List className="flex flex-col gap-4">
7          <Tab>
8            {({ selected }) => <button>Tab 1</button>}
9          </Tab>
10       </Tab.List>
11       <Tab.Panels>
12         <Tab.Panel>Content 1</Tab.Panel>
13       </Tab.Panels>
14     </Tab.Group>
15   )
16 }
```

Ukázka kódu 3.1: Ukázka použití headless knihovny

[9] `https://mantine.dev`

V porovnání všech tří přístupů na obrázku 3.1 můžeme vidět, že vyšší flexibilita přináší i vyšší úroveň abstrakce. To znamená, že čím flexibilnější je **Application Programming Interface (API)** komponenty, tím více času je potřeba pro pochopení, použití a údržbu komponent.

Knihovny komponent jsou nejvýhodnější, pokud není potřeba vlastního *brand*[10] designu a zároveň jsme spokojení s úrovní přístupnosti, kterou nám komponenta dodává.

Headless komponenty jsou lepší v situacích, kde potřebujeme zasáhnout významně do designu (kaskádových stylů) komponenty pro vlastní potřeby. Nicméně se tím navyšuje i čas strávený na vývoji.

UI primitiva jsou nejvýhodnější při budování vlastního designového systému, kde potřebujeme plnou kontrolu nad kódem komponenty.

Copy and paste přístup je nejvýhodnější, pokud chceme to nejlepší ze všech přístupů. Flexibilitu headless komponent případně UI primitiv a zároveň rychlost vývoji jako při použití knihovny komponent.

Obrázek 3.1: Diagram abstrakce a flexibility různých přístupu k distribuci komponent (diagram autora)

[10]Brand designem se rozumí vlastní barvy, fonty, styl ikon a celkově unikátní vizuální identita.

3.4 Astro

Astro[11] je **Static Site Generator (SSG)**, který používá vlastní syntaxi pro vytváření komponent. Velkou výhodou je, že Astro umožňuje vnořit komponenty z jiných frameworků (React, Vue, Solid.js, Svelte a další) do Astro komponent bez použití *iframe* [26]. Tento mechanismus lze použít pro vložení příkladové implementace komponenty přímo na stránku s dokumentací a zjednodušit tak uživatelům pochopení a účel komponenty.

3.4.1 Starlight

Vybranou technologií pro vytvoření dokumentace je `Starlight`[12] template postavený nad Astro knihovnou. Starlight funguje nad `MDX`[13], což je formát, který umožňuje vkládat JSX do markdown souborů.

3.5 SolidStart

Důležitým aspektem pro demo aplikaci je co nejbližší simulace reálného prostředí, kde se komponenty budou používat. Pro tento účel jsem zvolil meta-framework `SolidStart`. Meta-framework je své podstatě framework postavený nad vícero frameworky, který zjednodušuje různé aspekty vývoje. V kontextu meta-frameworků na webu se může jednat například o **Server-side Rendering (SSR)**, *routing* a *serverless* funkce [27].

SolidStart je oficiálně podporovaný a vyvíjený autorem Ryanem Carniatem, který je také autorem Solid.js. Výhoda testování komponent v prostředí meta-frameworku spočívá v tom, že komponenta je vystavená různým kontextům jako je SSR, SSG a **Client-side Rendering (CSR)**.

3.6 Závěr kapitoly

Tato kapitola popsala nejdůležitější technologie a přístupy, které jsou využity v rámci této práce. Plná funkčnost celého projektu je závislá na dalších, nezmíněných a důležitých technologiích, které zde nejsou popsány neboť nejsou přímo relevantní k problematice tvorby znovupoužitelných komponent.

[11] `https://astro.build`
[12] `https://starlight.astro.build`
[13] `https://mdxjs.com`

Samotná knihovna bude psána metodou primitiv, tedy funkcí v Solid.js, které vrací APG logiku jako props, state a reference (viz kapitola 7.1). Taková knihovna může poté sloužit jako základ pro tvorbu designových systémů a (headless) knihoven komponent.

Solid.js knihovnu jsem zvolil ze dvou důvodů:

1. Její ekosystém knihoven je stále v raných fázích a podobná knihovna respektive její části zde chybí.

2. Knihovna Solid.js má jisté vlastnosti, které ji odlišují od ostatních knihoven. Její reaktivní model je unikátní a přináší obrovské výhody z hlediska *performance* na webu. Tato knihovna má potenciál optimalizovat a zlepšit uživatelská rozhraní.

TypeScript je velice důležitý pro nové knihovny, neboť usnadňuje budoucí údržbu kódu a vývojářskou přívětivost. Ve světě webového vývoje je TypeScript dnes už téměř standardem a jeho nepoužití by vedlo k výrazně snížené adopci knihovny komunitou vývojářů [28].

Část II

Rešerše a analýza

Kapitola 4

Rešerše existujících řešení

Tato kapitola rozebírá existující řešení v rámci JavaScriptového ekosystému. Vyskytuje se tu bližší pohled na existující řešení pro React a Svelte, protože hodně knihoven ve Solid.js z nich vychází. Následně se rozebírá Solid.js a jeho chybějící části v rámci ekosystému.

4.1 React

React má obrovské zastoupení v rámci JavaScriptového ekosystému, proto kvalita knihoven zde bude vysoká díky velkému množství vývojářů a času věnovanému vývoji v této oblasti.

4.1.1 React Aria

React Aria[1] je open-source projekt od společnosti Adobe, který obsahuje velice robustní sadu primitiv v podobě React hooks. Obsahuje velké množství komponent a interakcí, od nejzákladnějších jako jsou tlačítka, formulářové prvky až po složitější komponenty jako kalendáře. Knihovna je velice dobře dokumentovaná a podporována komunitou. Zároveň je distribuovaná bez stylů, proto je vhodná pro využití v rámci designových systémů.

Výhody	Nevýhody
Flexibilita použití	Vyšší nároky na udržování
Rozmanitost komponent	—

Tabulka 4.1: Shrnutí výhod a nevýhod knihovny React Aria

[1] `https://react-spectrum.adobe.com/react-aria`

4.1.2 Radix UI

Radix UI[2] je knihovna komponent, která je zároveň postavená nad Radix UI primitives. Radix primitives využívá na pozadí headless komponent, ačkoliv by název mohl evokovat použití primitiv.

To je zásadní rozdíl oproti React Aria, kde jsou komponenty vytvořené pomocí primitives.

Výhody	Nevýhody
Jednoduchost použití	Vyšší nároky na udržování
—	Nižší rozmanitost komponent

Tabulka 4.2: Shrnutí výhod a nevýhod knihovny Radix UI

4.1.3 Shadcn UI

Shadcn UI[3] je copy and paste knihovna. Z dokumentace nebo CLI dostupného skrze npm je možné nakopírovat předpis komponenty do uživatelem vybraného projektu. Hodně komponent je založeno na Radix UI primitives.

Výhody	Nevýhody
Jednoduchost použití	Flexibilita je závislá na použitých knihovnách
Možnost upravit zdrojový kód	—

Tabulka 4.3: Shrnutí výhod a nevýhod Shadcn UI

4.2 Svelte

Svelte je novější framework, který se ubírá cestou nejvyššího komfortu pro vývojáře. Rozhodl jsem se provést rešerši v rámci tohoto frameworku, protože Solid.js i Svelte používají reaktivní model změn.

4.2.1 Melt UI

Podobně jako React Aria je Melt UI[4] knihovna primitiv, v kontextu této knihovny se primitiva nazývají “builders”. Uživatelé konzumují tyto primitiva

[2] `https://radix-ui.com`
[3] `https://ui.shadcn.com`
[4] `https://melt-ui.com`

vytváří si tak komponenty s vlastní HTML strukturou. Z toho plyne, že je knihovna velice flexibilní na používání, ale zároveň je tak zvýšená náročnost na udržování komponent. Výhodou je, že je možnost zde vytvořit příkladové použití těchto primitiv včetně základních stylů a usnadnit tak konzumentům knihovny práci.

Výhody	Nevýhody
Vysoká flexibilita použití	Vyšší nároky na udržování

Tabulka 4.4: Shrnutí výhod a nevýhod knihovny Melt UI

4.3 Solid.js

Solid.js je knihovna použitá v rámci této práce, proto je důležité provést rešerši a určit tak chybějící komponenty v rámci ekosystému. Případně zjistit, zda lze navázat na práci již existujících knihoven.

4.3.1 Solid Aria

Solid Aria[5] je port React Aria do Solid.js ekosystému podporovaný komunitou. Bohužel vývoj zde již není příliš aktivní, protože na rozdíl od React Aria tento projekt není sponzorovaný větší společností. Nicméně je zde možnost navázat na práci komunity a pokračovat v rozvoji této knihovny.

Velkou výhodou je, že port React kódu do Solid.js je na hodně místech velice podobný, proto je možné využít pokrok na samotné React Aria knihovně i zde.

Výhody	Nevýhody
Flexibilita použití	Neaktivní vývoj
—	Chybějící základní komponenty

Tabulka 4.5: Shrnutí výhod a nevýhod Solid Aria

4.3.2 Kobalte

Kobalte[6] je UI toolkit pro Solid.js, který obsahuje headless komponenty. Hodně práce na Kobalte vychází ze Solid Aria, protože minimálně jeden z

[5] `https://github.com/solidjs-community/solid-aria`
[6] `https://kobalte.dev`

hlavních vývojářů Kobalte hojně pracoval i na Solid Aria.

Výhody	Nevýhody
Jednoduchost použití	Nižší flexibilita

Tabulka 4.6: Shrnutí výhod a nevýhod knihovny Kobalte

4.4 Závěr rešerše

Z rešerše vyplynulo, že ekosystém okolo Solid.js obsahuje několik knihoven, které řeší problematiku znovupoužitelných, přístupných komponent. Další kapitola se zaměří na analýzu chybějících komponent z APG návrhových vzorů a zda je možné navázat na práci již existujícího řešení.

Kapitola 5

Analýza řešení

V této kapitole se rozebírá vybrané řešení pro tuto práci.

5.1 Aktuální stav

Z rešerše vyplynulo, že Solid.js ekosystém má existující knihovny, které se snaží pokrýt problematiku znovupoužitelných knihoven. V rámci této sekce jsem se rozhodl provést rešerši, které komponenty z APG návrhových vzorů chybí v rámci Solid.js ekosystému. V tabulce C.1 jsou vypsány všechny návrhové vzory z APG a jejich stav existence ve vybraných knihovnách z Solid.js ekosystému. Z tabulky se lze dočíst, že z APG chybí implementace 5 návrhových vzorů.

5.1.1 Navázání na existující knihovny

Z rešerše jsem se rozhodl navázat na existující knihovnu Solid Aria z následujících důvodů:

- Solid Aria je knihovna primitiv pro Solid.js. Primitiva mají výhodu, že jejich flexibilita umožňuje vytvořit headless komponenty, které by se mohly přidat do Kobalte.
- Vývoj Solid Aria je neaktivní na rozdíl od Kobalte. Je zde tak možnost obnovit vývoj Solid Aria knihovny.
- Solid Aria je port React Aria, což znamená, že práce na React Aria může být využita i zde.

5.1.2 Výběr komponent k implementaci

Pro úspěšný výsledek této práce je potřeba si definovat, které komponenty budou implementovány. Z tabulky C.1 vychází, že v ekosystému chybí 5

komponent. Z analýzy byly vynechány některé komponenty, protože jejich implementace by byla triviální, zbytečná, nebo nepraktická:

- **Landmarks** — Landmarks není komponenta, ale doporučení jak používat role pro popisování různých oblastí na webu.
- **Link** — Prohlížeče podporují správnou přístupnost.
- **Table** — Obsahuje hlavně poznatky k správnému použití HTML prvků. Neobsahuje složitou klávesovou či jinou logiku.
- **Window Splitter** — Není zkontrolovaná specifikace.

Z rozhodnutí navázat na Solid Aria knihovnu jsem se rozhodl zaměřit na komponenty, které zde chybí a jsou součástí populárních knihoven [29, 30]. Mezi tyto komponenty patří **Disclosure, SpinButton, Toolbar, Tooltip**.

5.2 Funkční požadavky

V této sekci jsou zadefinované funkční požadavky pro všechny vybrané komponenty k implementaci.

5.2.1 Obecné

V této podsekci jsou popsané obecné funkční požadavky, které by měly být splňovat všechny implementované komponenty.

- **OFR 1.1 — Navigace pomocí klávesových zkratek**

  Komponenta umožní klávesovou navigaci pomocí klávesových zkratek definovaných v APG.

- **OFR 1.2 — Definované aria atributy**

  Komponenta má správně nadefinované role z WAI-ARIA atributů pro všechny její elementy a to včetně pomocných stavových atributů.

- **OFR 1.3 — Pojmenování instance komponenty**

  Komponenta umožní její pojmenování instance dle specifikace WAI-ARIA.

5.2.2 Disclosure

- **DFR 1.1 — Otevírání a zavírání obsahu**

  Komponenta umožní otevírání a zavírání obsahu pomocí klávesových zkratek a tlačítka.

- **DFR 1.2 — Nastavení vlastního stavu**

  Komponenta umožní ovládat její stav z vnějšího zdroje.

5.2.3 SpinButton

- **SFR 1.1 — Diskrétní hodnoty**

  Komponenta umožní nastavit diskrétní hodnoty ze kterých může uživatel vybírat.

- **SFR 1.2 — Zvyšování a snižování hodnoty**

  Komponenta umožní zvyšování a snižování hodnoty pomocí klávesových zkratek a tlačítek.

5.2.4 Toolbar

- **TFR 1.1 — Vnoření libovolného počtu subkomponent**

  Komponenta umožní vkládání libovolného počtu subkomponent, které se skládají z *focusovatelných*[1] elementů a oddělovačů obsahu.

- **TFR 1.2 — Horizontální a vertikální pohled**

  Komponenta umožní nastavit horizontální a vertikální pohled, který mění způsob klávesové navigace v subkomponentách.

5.2.5 Tooltip

- **TLFR 1.1 — Zobrazení a skrytí obsahu Tooltip komponenty**

  Komponenta umožní zobrazit a skrýt obsah Tooltip komponenty pomocí klávesnice a myši.

[1] Elementy, které můžou dostat focus od uživatele (např. tlačítka, zaškrtávací políčka…).

5.3 Nefunkční požadavky

V této sekci jsou zadefinované důležité nefunkční požadavky napříč všemi implementovanými komponenty tak, aby byla zaručena jejich vývojářská přívětivost, použitelnost a kvalita.

- **NFR 1.1 — Podpora prohlížečů**

  Komponenty fungují v dvou posledních verzích prohlížečů Chrome[2], Firefox[3] a Safari[4].

- **NFR 1.2 — Podpora odečítačů obrazovky**

  Komponenty fungují s odečítačem obrazovek VoiceOver pro MacOS.

- **NFR 1.3 — Developer experience**

  Komponenty poskytují doplňování kódu pro vývojáře v Editoru, nebo **Integrated Development Environment (IDE)**.

- **NFR 1.4 — Dokumentace API**

  Dokumentace popisuje API komponenty a její anatomii, účel, použití a příkladový kód.

- **NFR 1.5 — Demo aplikace**

  Demo aplikace demonstruje použití komponenty a její funkčnost.

5.4 HTA diagram vývoje

Podle B. Kirwena a L. K. Ainsworth je **Hierarchical Task Analysis (HTA)** diagram "nejlepší technika pro analýzu úkolů" [31]. Rozhodl jsem se tak použít tento diagram pro vizualizaci úkolů, které je potřeba udělat pro vytvoření knihovny komponent. Z důvodu rozsahu je vizualizace v příloze B.1.

Hlavním úkolem je vytvoření knihovny. Tento úkol se dělí na dvě důležité fáze a to příprava projektu (repozitáře) a následná implementace komponent včetně testování a dokumentace. Součástí přípravy projektu je i nasazení výstupů (demo aplikace, dokumentace a vygenerovaných souborů). Rozhodl jsem se zařadit nasazení už ve fázi přípravy projektu, protože žádný z výstupů neobsahuje komplexní backendovou logiku, která by vyžadovala migraci dat.

[2] `https://google.com/chrome`
[3] `https://mozilla.org/en-US/firefox`
[4] `https://apple.com/safari`

5.5 Závěr analýzy

Z rešerše a analýzy vyplývá, že jsem se rozhodl navázat na existující knihovnu Solid Aria a implementovat 4 chybějící komponenty z APG návrhových vzorů (Disclosure, SpinButton, Toolbar, Tooltip). Navázaní na existující řešení umožní ušetřit čas na vývoji a zároveň vylepšit použitelnost samotné knihovny Solid Aria. V neposlední řadě je možné využít technik z React Aria knihovny, která má větší zdroje na vývoj a testování.

Část III

Návrh a implementace

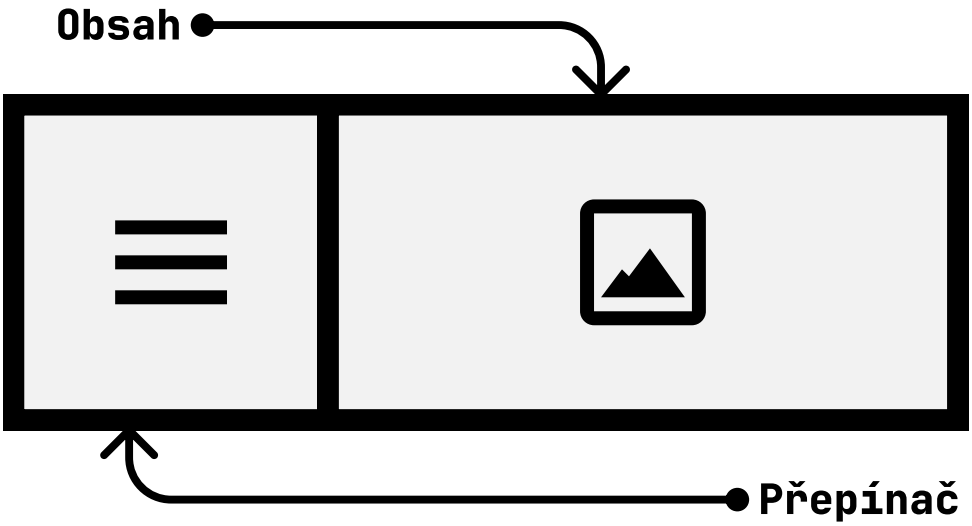
Kapitola 6

Návrh

Tato kapitola popisuje návrh implementace a komplementárních struktur.

6.1 Disclosure

Disclosure je komponenta pro zobrazení a skrývání obsahu. Typicky se skládá z tlačítka, které otevírá nebo zavírá obsah, a samotného obsahu.



Obrázek 6.1: Low-fidelity wireframe komponenty Disclosure (wireframe autora)

Pro splnění požadavku `OFR 1.1` je potřeba zajistit, že klávesy Space, nebo Enter mění stav otevření obsahu.

6.1.1 Controlled vs Uncontrolled

Pro splnění požadavku `DFR 1.2` je potřeba zajistit, že komponenta může být tzv. *controlled* nebo *uncontrolled*. Většina implementací je ve výchozím režimu

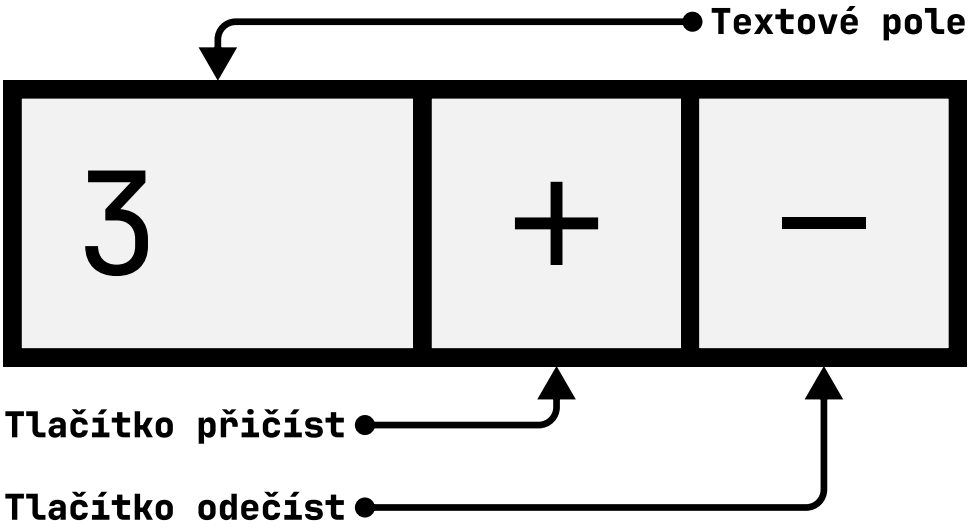
uncontrolled, kdy je stav otevření obsahu řízen interním stavem komponenty. Druhou možností je, aby byla komponenta controlled, kdy je stav otevření obsahu řízen vnějším zdrojem. V Solid.js je toto řešení implementované pomocí props, kdy z vnějšku komponentě předáme náš vlastní stav, který je následně používaný na místo interního stavu.

```
1  const [open, setOpen] = createSignal(false);
2
3  const { togglеProps } = createDisclosure({
4      isVisible: open,
5      onVisibilityChange(state) { setOpen(state) }
6  });
```

Ukázka kódu 6.1: Ukázka primitivní funkce s vnějším stavem (controlled stav)

6.2 SpinButton

SpinButton je formulářová komponenta, která vymezuje rozsah diskrétních hodnot a typicky umožňuje uživateli změnit její hodnotu pomocí tlačítek snížit a navýšit, nebo alternativně skrze textový vstup.



Obrázek 6.2: Low-fidelity wireframe komponenty SpinButton (wireframe autora)

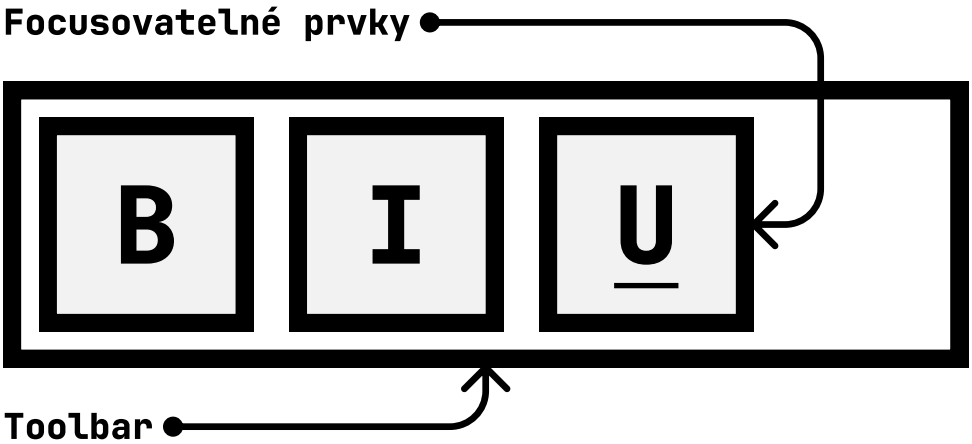
Pro splnění požadavku `OFR 1.1` a `SFR 1.2` je potřeba splnit následující

podmínky:

1. Klávesové šipky ↑ a ↓ zvýší, respektive sníží hodnotu o jedna.
2. Klávesy PageUp a PageDown zvýší, respektive sníží hodnotu o vybraný krok.
3. Klávesy Home a End nastaví hodnotu na minimum, respektive maximum.

6.3 Toolbar

Toolbar je strukturální komponenta, která obsahuje skupinu focusovatelných elementů.



Obrázek 6.3: Low-fidelity wireframe komponenty Toolbar (wireframe autora)

Hlavní přínos této komponenty je jednodušší klávesová navigace pomocí klávesových šipek a dalších zkratek namísto klasického proklikávání tabulátorem.

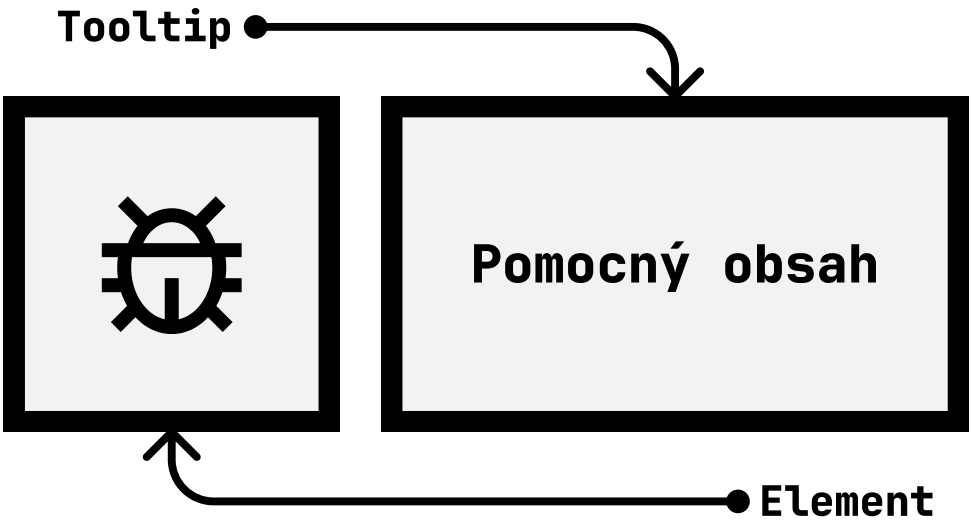
Pro splnění požadavku `OFR 1.1` je potřeba splnit následující podmínky:

1. Klávesa ⇆ (tab), nebo klávesová zkratka ⇆+⇧ skočí na první, nebo naposled vybraný focusovatelný prvek v rámci Toolbaru.
2. Klávesa ⇆, když je focus v rámci Toolbaru, skočí na první focusovatelný prvek mimo Toolbar, který se nachází v hierarchii za Toolbar komponentou.
3. Klávesová zkratka ⇆+⇧, když je focus v rámci Toolbaru, skočí na první focusovatelný prvek mimo Toolbar, který se nachází v hierarchii před Toolbar komponentou.

4. Klávesové šipky ← a → přesunou focus mezi focusovatelnými prvky v rámci Toolbaru pokud je orientace toolbaru horizontální.
5. Klávesové šipky ↑ a ↓ přesunou focus mezi focusovatelnými prvky v rámci Toolbaru pokud je orientace toolbaru vertikální.
6. Klávesy Home a End přesunou focus na první, respektive poslední focusovatelný prvek v rámci Toolbaru.

6.4 Tooltip

Tooltip je komponenta, která zobrazí nápovědu u vybraného elementu (nejčastěji se bude jednat o interaktivní element) po najetí myši, nebo při předání focusu na daný element (viz funkční požadavek `TLFR 1.1`).



Obrázek 6.4: Low-fidelity wireframe komponenty Tooltip (wireframe autora)

Pro splnění požadavku `OFR 1.1` je potřeba zajistit jedinou klávesovou interakci pomocí klávesy Esc, která uzavře obsah pokud je zobrazený.

6.5 Dokumentace

Součástí zadání práce a požadavku `NFR 1.4` je vytvoření dokumentace pro implementované komponenty. Dokumentace by měla obsahovat několik zásadních sekcí:

1. **Anatomie komponenty**

   Anatomie by měla popisovat způsob naimportování knihovny, její typovou signaturu a low-fidelity návrh.

2. **API reference**

   API reference by měla popisovat vstupní parametry, jejich výchozí hodnoty a typ tak, aby ušetřila čas vývojáři pro pochopení implementace.

3. **Příklad**

   Příklad by měl být vnořený na stránce tak, aby komponentu šlo používat jako v normální aplikaci. Součástí příkladu by měl být kopírovatelný zdrojový kód.

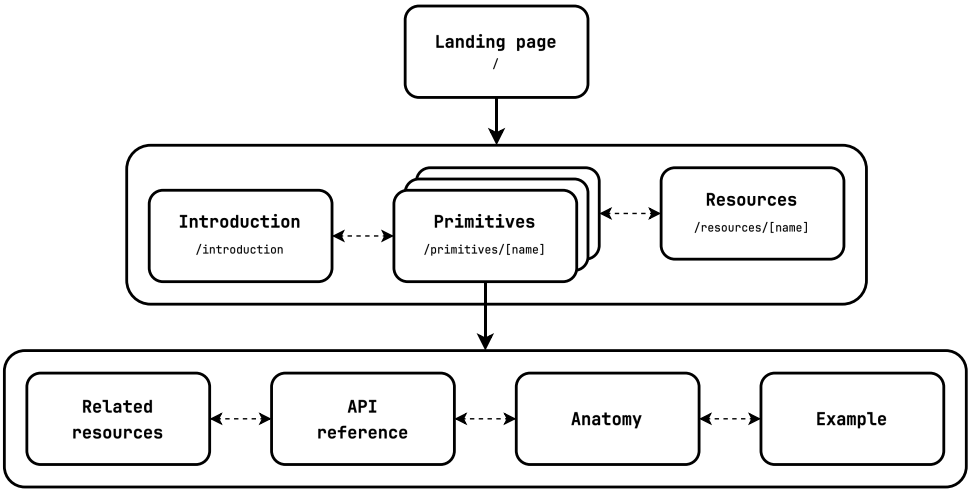
4. **Ukázka s odečítačem obrazovky**

   V dokumentaci by měla být i ukázka toho, jak komponenta reaguje na použití s odečítačem obrazovky.

Kromě výše zmíněných částí u každé stránky s komponentou by měla dokumentace obsahovat informace o tom, jak knihovnu nainstalovat a které technologie je potřeba mít nainstalované. V neposlední řadě by měla dokumentace obsahovat odkazy na informaci o implementovaném návrhovém vzoru, výsledky testů a pokrytí kódu testy.

6.5.1 Diagram obrazovek

Na obrázku 6.5 je diagram obrazovek dokumentace, který i nepřímo popisuje komponentovou strukturu dokumentace. Nejdůležitější částí je sekce `Primitives`, která obsahuje stránky dokumentující každou implementovanou komponentu.



Obrázek 6.5: Diagram obrazovek dokumentace (diagram autora)

Kapitola 7

Implementace

Tato kapitola popisuje implementaci komponent a jejich nasazení.

7.1 Struktura logiky

Všechna logika komponenty je zapouzdřena v primitivní funkci. Tato funkce nese název `create*`, kde `*` je název komponenty. Komponenty mají většinou podobnou signaturu funkce, která vrací objekt s následujícími *properties*:

- **props** — objekt s ARIA a HTML atributy a *event listeners*.
- **state** — stav komponenty, většinou se jedná o vnitřní stav komponenty, který chceme vystavit uživateli komponenty.
- **reference** — některé komponenty potřebují referenci na přímý HTML element, který se v Solid.js předává pomocí proměnných[1] a direktiv[2].

Některé komponenty (například SpinButton) mají více HTML elementů na které je potřeba navázat ARIA atributy a event listeners, proto jejich funkce vrací více **props** properties (například buttonProps, rootProps). Při více props properties je potřeba navázat danou property na odpovídající HTML element, který je zmíněn v dokumentaci.

Koncový uživatel tuto primitivní funkci zavolá v definici komponenty a předá jí všechny potřebné argumenty. Manuálně naváže props na HTML elementy, předá potřebné reference a využije state pro vykreslení stavu komponenty. Výhodou tohoto přístupu z kapitoly 3.3 je, že vývojář má absolutní kontrolu nad HTML strukturou komponenty a použitým CSS řešením.

[1] `https://solidjs.com/tutorial/bindings_refs`
[2] `https://solidjs.com/tutorial/bindings_directives`

```
1  import { createExample } from "lib"
2
3  export const Example = (args) => {
4      const {
5          rootProps,
6          buttonProps,
7          state
8      } = createExample(args)
9
10     return (
11         <div {...rootProps}>
12             <button {...buttonProps}>Toggle</button>
13             {state.isVisible ? <p>Content</p> : null}
14         </div>
15     )
16 }
```

Ukázka kódu 7.1: Příklad implementace komponenty pomocí primitivní funkce

Na ukázce 7.1 je příklad použití primitivní funkce `createExample` pro vytvoření komponenty `Example`. Řádek 8 zavolá primitivní funkci a předá jí argumenty, které jsou definovány v `ExampleArgs`. Elementům na řádku 11 a 12 jsou předány atributy a event listeners pomocí spread operátoru. Na řádku 13 je ukázka využití vnitřního stavu komponenty pro vykreslení obsahu.

7.1.1 Adresářová struktura

Každá komponenta má svůj adresář, který obsahuje všechny soubory týkající se její implementace:

```
example
├── example.spec.ts (jednotkové testy)
├── example.ts (logika komponenty)
├── index.ts (barrel export z types.ts a example.ts)
└── types.ts (definice typů pro komponentu)
```

Soubor `index.ts` slouží pro export všech položek ze souborů `types.ts` a `example.ts` pro usnadnění distribuce. Tato technika se nazývá *barrel* export.

7.2 Disclosure

Požadavek na klávesovou navigaci `OFR 1.1` a otevírání/zavírání obsahu `DFR 1.1` je splněn pomocí primitivní funkce `createDisclosure`.

Příkladová implementace komponenty Disclosure pomocí primitivní funkce `createDisclosure` je možná vidět na ukázce D.1.

7.2.1 Parametry

Funkce má jeden parametr, kterým je objekt s následujícími properties:

Název	Datový typ	Výchozí hodnota
`defaultVisible`	`boolean`	`false`
`id`	`string`	—
`isButtonElement`	`boolean`	`true`
`isVisible`	`boolean`	—
`onVisibilityChange`	`Function`	—

Tabulka 7.1: Parametry funkce createDisclosure

Z ukázky kódu 6.1 je patrné, že `isVisible` a `onVisibilityChange` properties slouží pro řízení stavu komponenty z vnějšku (komponenta typu controlled), jinak je komponenta uncontrolled. Pro případ uncontrolled komponent slouží property `defaultVisible` pro změnu výchozího zobrazení obsahu.

Property `isButtonElement` je určena pro případ, kdy element pro otevírání a zavírání obsahu není `button` a je potřeba přidat správnou ARIA roli a další atributy pro správnou interpretaci odečítači obrazovky.

V neposlední řadě property `id` slouží pro definování vlastního identifikátoru elementu s obsahem. Ve výchozím stavu je identifikátor generován automaticky pomocí `createUniqueId`[3].

[3] `https://docs.solidjs.com/reference/component-apis/create-unique-id`

7.2.2 Návratová hodnota

Tato funkce vrací objekt s properties `toggleProps`, `contentProps` a `state`.

Název	Popis
`contentProps`	Event listeners, atributy pro element držící obsah Disclosure.
`state`	Vnitřní stav komponenty.
`toggleProps`	Event listeners, atributy pro tlačítko pro otevření/zavření obsahu.

Tabulka 7.2: Návratová hodnota funkce createDisclosure

7.2.3 ARIA atributy

Požadavek `OFR 1.2` u Disclosure komponenty vynucuje následující atributy:

Název	Objekt	Typ
`aria-expanded`	`toggleProps`	Stav
`aria-controls`	`toggleProps`	Vlastnost

Tabulka 7.3: Použité ARIA atributy pro Disclosure komponentu

Stavový atribut `aria-expanded`[4] určuje, zda je obsah otevřený nebo zavřený a vlastnost `aria-controls`[5] určuje, který element je ovládán tlačítkem.

[4] `https://w3.org/TR/wai-aria/#aria-expanded`
[5] `https://w3.org/TR/wai-aria/#aria-controls`

7.3 SpinButton

Požadavky **SFR 1.1** a **SFR 1.2** jsou splněny pomocí primitivní funkce `createSpinButton`.

Příkladová implementace komponenty SpinButton pomocí primitivní funkce `createSpinButton` je možná vidět na ukázce D.2.

7.3.1 Parametry

Funkce má jeden parametr, kterým je objekt s následujícími properties:

Název	Datový typ	Výchozí hodnota
`values *`	`T[]`	—
`mapping`	`string[]`	—
`step`	`number`	1

Tabulka 7.4: Parametry funkce createSpinButton

Property `values` je pole hodnot generického typu, který ale musí být typu `number`, nebo `string`. Může se stát situace, kdy hodnoty ve `values` nemají sémantický význam, proto jejich význam je možné definovat v property `mapping`.

Poslední property `step` určuje o kolik se má hodnota zvýšit, nebo snížit při stisknutí kláves PageUp a PageDown.

7.3.2 Návratová hodnota

Tato funkce vrací objekt s properties `inputProps`, `decrementButtonProps`, `incrementButtonProps` a `state`.

Název	Popis
`decrementButtonProps`	Event listeners a atributy pro tlačítko snižující hodnotu
`incrementButtonProps`	Event listeners a atributy pro tlačítko zvyšující hodnotu
`inputProps`	Event listeners a atributy pro vstupní pole.
`state`	Vnitřní stav komponenty.

Tabulka 7.5: Návratová hodnota funkce createSpinButton

7.3.3 ARIA atributy

Požadavek `OFR 1.2` u SpinButton komponenty vynucuje následující atributy:

Název	Objekt	Typ
`role`	`inputProps`	Role
`aria-valuenow`	`inputProps`	Vlastnost
`aria-valuemin`	`inputProps`	Vlastnost
`aria-valuemax`	`inputProps`	Vlastnost
`aria-valuetext`	`inputProps`	Vlastnost
`aria-invalid`	`inputProps`	Stav

Tabulka 7.6: Použité ARIA atributy pro SpinButton komponentu

Role je nastavena na `"spinbutton"`. Vlastnost `aria-valuenow`[6] definuje momentální index hodnoty komponenty a spolupracuje synergicky s atributem `aria-valuetext`[7], který definuje textovou podobu hodnoty.

Dvě důležité vlastnosti `aria-valuemin`[8] a `aria-valuemax`[9] definují minimální, respektive maximální hodnotu, kterou může komponenta nabývat.

Stavový atribut `aria-invalid`[10] definuje zda je hodnota vstupního pole validní, tedy zda se nachází v hodnotách z pole `values`.

[6] `https://w3c.github.io/aria/#aria-valuenow`
[7] `https://w3c.github.io/aria/#aria-valuetext`
[8] `https://w3c.github.io/aria/#aria-valuemin`
[9] `https://w3c.github.io/aria/#aria-valuemax`
[10] `https://w3c.github.io/aria/#aria-invalid`

7.4 Toolbar

Stěžejní logikou komponenty Toolbar je klávesová navigace. Zásadní je správné chování focusu, které jsem přejal z knihovny React Aria, které má robustní řešení s ohledem na různé druhy prohlížečů a kombinace odečítačů obrazovky.

Přejaté soubory jsou popsané níže:

- **dom-helpers.ts** — Pomocné funkce pro získání správného `ownerWindow` a `ownerDocument` objektů z DOM.
- **focus-safely.ts** — Funkce pro přesun focusu na element bez vedlejších efektů pro odečítače obrazovky.
- **focus-without-scrolling.ts** — Funkce pro přesun focusu na element bez posunu stránky.
- **focus.ts** — Funkce pro přesun focusu v rámci elementu.
- **interactions.ts** — Kód, který detekuje jakým způsobem uživatel interaguje se stránkou (klávesnice, dotek, nebo virtualizované klikání)
- **is-element-visible.ts** — Funkce kontrolující zda je daný element viditelný.
- **is-virtual-click.ts** — Funkce detekující virtuální klikání.
- **live-announcer.ts** — Třída `LiveAnnouncer`, která přidává neviditelné elementy do DOM s oznámením změn pro odečítače obrazovky.
- **platform.ts** — Funkce pro detekci platformy na kterém běží kód. Slouží pro případnou korekci chování v různých prohlížečích.
- **run-after-transition.ts** — Funkce, která zavolá callback funkci jakmile skončí všechny přechody v rámci stránky.

Samotná implementace komponenty exportuje funkci `createToolbar` a její použití pro implementaci komponenty Toolbar je možná vidět na ukázce D.3.

7.4.1 Parametry

První parametr je objekt pro konfiguraci Toolbaru s následujícími properties:

Název	Datový typ	Výchozí hodnota
`orientation`	`"horizontal"|` `"vertical"`	`"horizontal"`
`aria-labelledby`	`string`	—
`aria-label`	`string`	—

Tabulka 7.7: Parametry createToolbar funkce

Parametr `orientation` slouží pro splnění požadavku `TFR 1.2` a určuje jaké klávesy slouží pro navigaci v Toolbaru. Pokud je nastaveno `"horizontal"`, tak šipky vlevo a vpravo přesunou focus mezi focusovatelnými prvky v rámci Toolbaru. U hodnoty `"vertical"` se používají šipky nahoru a dolů.

Parametry `aria-labelledby`[11] a `aria-label`[12] slouží pro pojmenování instance komponenty a splnění tak požadavku `OFR 1.3`.

Druhým parametrem funkce je nepovinná Solid.js reference na kořenový element Toolbaru. Pokud není reference předána, tak je možné ji získat pomocí Solid.js direktivy `use:toolbarRef`, která je navrácena z volání funkce `createToolbar`.

7.4.2 Návratová hodnota

Tato funkce vrací objekt s properties `toolbarProps` a nepovinně `toolbarRef`.

Název	Popis
`toolbarProps`	Event listeners, atributy pro element držící obsah Toolbaru.
`toolbarRef`	Solid.js direktiva pro předání reference na HTML element.

Tabulka 7.8: Návratová hodnota funkce createToolbar

[11] `https://w3.org/TR/wai-aria/#aria-labelledby`
[12] `https://w3.org/TR/wai-aria/#aria-label`

7.4.3 ARIA atributy

Požadavek `OFR 1.2` u Toolbar komponenty vynucuje následující atributy:

Název	Objekt	Typ
`role`	`toolbarProps`	Role
`aria-label`	`toolbarProps`	Vlastnost
`aria-labelledby`	`toolbarProps`	Vlastnost
`aria-orientation`	`toolbarProps`	Vlastnost

Tabulka 7.9: Použité ARIA atributy pro Toolbar komponentu

Role je nastavena na `"toolbar"`, aby odečítače obrazovky správně interpretovali element jako Toolbar komponentu. Pokud je v rámci Toolbaru vnořený jiný Toolbar, tak role vnitřního Toolbaru je nastavena na `"group"`. Vlastnost `aria-orientation`[13] je nastavena dle hodnoty argumentu `orientation`, obdobně pro vlastnosti `aria-label` a `aria-labelledby`.

7.5 Tooltip

Pro implementaci Tooltipu bylo využito následujících primitiv ze Solid Aria knihovny:

- **createFocusable** — Vytvoří z elementu focusovatelný element.
- **createFocusVisible** — Sleduje jakým zařízení uživatel interaguje se stránkou a podle toho rozhoduje, zda má být viditelný focus.
- **createHover** — Normalizuje *hover* interakci napříč prohlížeči a dotykovými zařízeními.
- **createInteractionModality** — Sleduje jakým zařízení uživatel interaguje se stránkou.
- **createOverlayTriggerState** — Vytvoří booleovský stav pro zobrazení obsahu.

Logika této komponenty vychází z React Aria implementace a celkově jě rozdělena do tří funkcí. Jejich použití pro implementaci komponenty Tooltip je možné vidět na ukázce D.4.

[13] `https://w3.org/TR/wai-aria/#aria-orientation`

7.5.1 createTooltipTriggerState

První funkce slouží pro správu stavu Tooltipu. Zároveň tato funkce spravuje globální stav všech otevřených Tooltipů tak, aby vždy byl otevřený jenom jeden z nich. Také se stará o rychlost zavírání a otevírání obsahu Tooltipu.

Ze Solid Aria knihovny byla využita funkce `createOverlayTriggerState`. Jedná se o jednoduchou funkci, která vytváří booleovský stav. Na tuto funkci byla navázána vlastní logika pro správu globálního stavu otevření Tooltipů a časování otevírání obsahu.

Parametry

Tato funkce má jeden parametr, kterým je objekt s následujícími properties:

Název	Datový typ	Výchozí hodnota
`closeDelay`	`number`	`500`
`defaultOpen`	`boolean`	—
`delay`	`number`	1500
`isOpen`	`false`	—
`onOpenChange`	`Function`	—

Tabulka 7.10: Parametry createTooltipTriggerState funkce

Properties `closeDelay` a `delay` určují rychlost zavírání, respektive otevírání obsahu Tooltipu. Pro případ, kdy uživatel chce měnit stav Tooltipu z vnějšího prostředí (controlled), tak lze použít properties `isOpen` a `onOpenChange`. Pokud uživatel nechává interní stav komponenty, tak je možné použít property `defaultOpen` pro nastavení výchozího stavu otevření.

Návratová hodnota

Tato funkce vrací jeden objekt s properties `open`, `close` a `isOpen`.

Název	Popis
`close`	Callback funkce pro zavření tooltipu.
`isOpen`	Proměnná indikující stav otevření tooltipu.
`open`	Callback funkce pro otevření tooltipu.

Tabulka 7.11: Návratová hodnota funkce createTooltipTriggerState

7.5.2 createTooltipTrigger

Druhá funkce slouží pro vytvoření elementu s ARIA atributy a event listeners pro který se má zobrazit Tooltip obsah.

Ze Solid Aria knihovny jsou využity funkce `createInteractionModality`, `createFocusable`, `createHover` a `createFocusVisible`.

Parametry

Tato funkce má dva parametry. První parametr je objekt s properties v tabulce 7.12. Druhý parametr je Solid.js reference na element, který má zobrazit Tooltip.

Název	Datový typ	Výchozí hodnota
`isDisabled`	`boolean`	`false`
`trigger`	`"focus"| undefined`	—

Tabulka 7.12: Parametry createTooltipTrigger funkce

Property `isDisabled` slouží pro deaktivaci Tooltipu, když je nastavena na `true`, tak Tooltip se nikdy nezobrazí. Property `trigger` určuje na jaký typ interakce se má Tooltip zobrazit. Ve výchozím režimu se Tooltip otevírá při hover i focus interakci. Při nastavení hodnoty `"focus"` se Tooltip otevírá pouze při focus interakci jinak se otevírá při všech výše zmíněných interakcích.

Návratová hodnota

Tato funkce vrací objekt s properties `tooltipProps` a `triggerProps`.

Název	Popis
`tooltipProps`	Event listeners, atributy pro element držící obsah Tooltipu.
`triggerProps`	Event listeners, atributy pro element, který má na sebe navázaný Tooltip.

Tabulka 7.13: Návratová hodnota funkce createTooltipTrigger

ARIA atributy

Požadavek `OFR 1.2` u `createTooltipTrigger` vyžaduje jedinou vlastnost `aria-describedby`[14], která by měla nést hodnotu `id` atributu elementu s obsahem Tooltipu. To je i důvod proč funkce vrací `tooltipProps` objekt, který s sebou nese atribut `id`, který odpovídá hodnotě předané v `aria-describedby`. Samotný atribut `aria-describedby` slouží pro dodatečný popis elementu, který se nachází na jiném elementu v textové podobě.

7.5.3 createTooltip

Poslední funkce slouží pro vytvoření elementu s ARIA atributy a event listeners pro obsah Tooltipu. Zde je využita funkce `createHover` ze Solid Aria knihovny, protože když uživatel myší přejede na obsah Tooltipu, tak je nežádoucí aby se obsah zavřel.

Parametry

Tato funkce má dva parametry. První parametr je objekt, který přijímá veškeré validní DOM atributy. Druhý parametr je Solid.js reaktivní proměnná, která vrací hodnotu z `createTooltipTriggerState`, protože jsou zde potřeba metody `open` a `close` pro otevření, respektive zavření obsahu při hover interakci.

Návratová hodnota

Tato funkce vrací objekt s jedinou property `tooltipProps`.

Název	Popis
`tooltipProps`	Event listeners, atributy pro element držící obsah Tooltipu.

Tabulka 7.14: Návratová hodnota funkce createTooltip

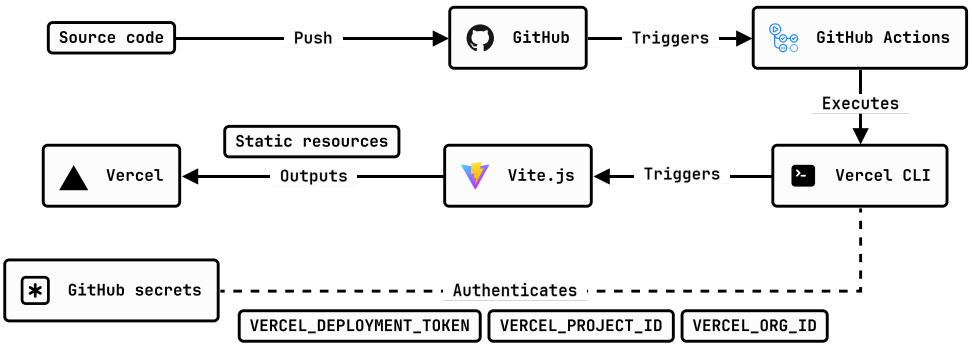
[14] `https://w3.org/TR/wai-aria/#aria-describedby`

7.6 Dokumentace

Na ukázce D.5 řádek 1 až 8 definuje tzv. *frontmatter*, který obsahuje metadata o stránce jako je název a popis. Komponenta **`Links`** (řádek 10) je Astro komponenta, která zobrazuje odkazy na různé užitečné zdroje jako je zdrojový kód, APG dokumentace dané komponenty, pokrytí kódu, nebo vygenerovaná typedoc dokumentace. Další užitečnou Astro komponentou je **`PropTable`** (řádek 20), která zobrazuje tabulku argumentů včetně výchozích hodnot a jejich typu. V neposlední řadě na řádku 24 je ukázka použití implementované komponenty v Solid.js, v tomto případě se jedná o Toolbar. Direktiva `client:idle` je specifická pro Astro, která nám říká jakým způsobem se má komponenta hydratovat.

7.6.1 Nasazení

Pro nasazení dokumentace jsem zvolil platformu Vercel[15], která umožňuje nasazení statických stránek pomocí SSG jako je například Astro.



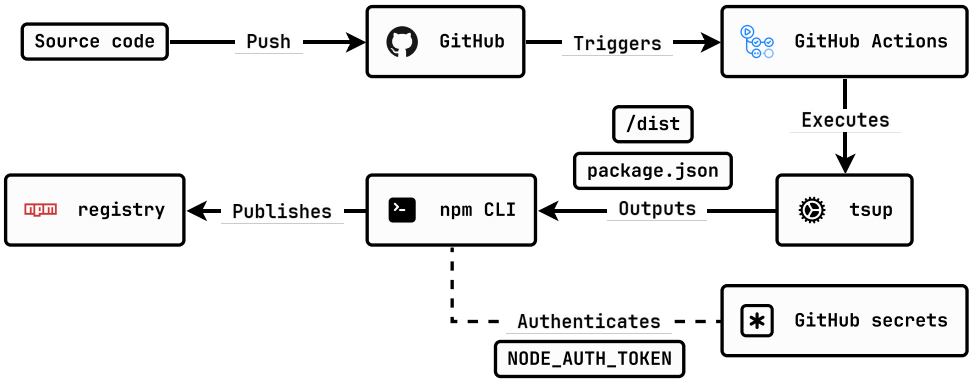
Obrázek 7.1: Diagram nasazení dokumentace (diagram autora)

Průběh nasazení začíná dodáním kódu do GitHub repozitáře projektu. Následně je možné manuálně spustit action[16], která pomocí CLI spustí na platformě Vercel *build* a provede nasazení aplikace. Vercel si stáhne nejnovější verzi kódu z GitHubu a provede build aplikace stejně jako v lokálním prostředí vývojáře. Nasazená aplikace je dostupná skrze **Content Delivery Network (CDN)** a je možné ji zobrazit v prohlížeči. Vercel se stará o cachování a distribuci aplikace po celém světě [32].

[15] `https://vercel.com`
[16] `https://github.com/features/actions`

7.7 Distribuce knihovny

JavaScriptové knihovny jsou distribuovány pomocí npm registry[17]. Pro distribuci je potřeba nadefinovat manifest soubor `package.json`[18], který obsahuje metadata o knihovně.



Obrázek 7.2: Diagram nasazení knihovny (diagram autora)

Důležitým krokem je připravit zdrojový kód k distribuci. Pro tento účel jsem zvolil knihovnu `tsup`[19], která umožňuje jednoduše vytvořit balíček pro distribuci a to včetně TypeScript podpory a různých formátů modulů.

Knihovna je publikovaná pod názvem `solid-apg` a lze ji nainstalovat jedním z příkazů v ukázce 7.2.

```
npm install solid-apg
# or
yarn add solid-apg
# or
pnpm add solid-apg
# or
bun add solid-apg
```

Ukázka kódu 7.2: Instalace knihovny solid-apg

[17] `https://npmjs.com`
[18] `https://docs.npmjs.com/cli/v10/configuring-npm/package-json`
[19] `https://tsup.egoist.dev`

Část IV

Testování a závěr

Kapitola 8

Testování

Tato kapitola popisuje testování implementovaných komponent.

8.1 Jednotkové testy

Jednotkové testy kontrolují logiku komponent, aby byla zaručena jejich správná implementace podle APG návrhového vzoru. Zvolený *runner*[1] pro jednotkové testy je `vitest`[2] a k tomu další pomocné knihovny:

1. `@vitest/coverage-v8`[3] — pro generování pokrytí kódu testy.
2. `@vitest/ui`[4] — pro generování grafického zobrazení výsledků testů.
3. `@testing-library/*`[5] — pro testování uživatelského rozhraní.
4. `happy-dom`[6] — pro testování DOM interakcí bez prohlížeče.

Cílem testů bylo pokrýt většinu požadavků z APG návrhových vzorů, proto by pokrytí testů mělo být vysoké a u všech komponent přesahuje >85% otestovaných řádků implementace[7].

Nejčastěji testovaná logika se týkala správného nastavení WAI-ARIA atributů a klávesové navigace komponenty. Také byly testovány různé varianty vstupních argumentů a návratových hodnot. Celkově bylo napsáno 42 jednotkových testů.

[1] Software, který spouští testy a generuje výsledek.
[2] `https://vitest.dev`
[3] `https://vitest.dev/guide/coverage`
[4] `https://vitest.dev/guide/ui`
[5] `https://testing-library.com`
[6] `https://npmjs.com/package/happy-dom`
[7] `https://susickypavel.github.io/solid-apg/coverage`

Název	Statements	Branches	Functions	Lines
disclosure	100%	100%	100%	100%
spinbutton	88.52%	95%	91.66%	88.52%
state	91.5%	83.33%	100%	91.5%
toolbar	95.51%	88%	100%	95.51%
tooltip	100%	100%	100%	100%

Tabulka 8.1: Pokrytí jednotkových testů

8.2 Testování v prohlížeči

Nefunkční požadavek `NFR 1.1` říká, že komponenty by neměly vyhazovat výjimky ve výše definovaných prohlížečích. Pro tento účel a nefunkční požadavek `NFR 1.5` vznikla demo aplikace, kde je možné implementované komponenty vyzkoušet[8]. V budoucnu bych chtěl zautomatizovat testování v prohlížeči pomocí End-to-End (E2E) testovacího nástroje `playwright`[9]. Z časových důvodů se E2E testování nepodařilo implementovat v rámci této práce.

8.3 Testování s odečítačem obrazovky

Nefunkční požadavek `NFR 1.2` říká, že by komponenty měly fungovat správně s odečítačem obrazovky VoiceOver na MacOS. Pro kontrolu, že implementace funguje správně jsem zvolil manuální testování v prohlížeči na bázi chromia[10]. Videa z testování jsou dostupná v dokumentaci na stránce o vybrané komponentě[11].

Je důležité podotknout, že podobně jako testování přívětivosti uživatelského prostředí je důležité brát v potaz, že uživatelům nemusí vyhovovat jakým způsobem komponenty reagují na jejich změny a to i přestože jsou implementované podle standardu WAI-ARIA. Vývojáři by v praxi měli využít zpětné vazby jejich uživatelů a případně upravit implementaci tak, aby vyhovovala jejich potřebám, avšak standard WAI-ARIA vychází z dlouholetých zkoumání problematiky přístupnosti, proto komponenty splňující tento standard by měly být dostatečně přístupné pro většinu uživatelů.

[8] `https://solid-apg-app.vercel.app`
[9] `https://playwright.dev`
[10] `https://chromium.org/chromium-projects`
[11] `https://solid-apg-docs.vercel.app`

Kapitola 9

Závěr

Výsledkem práce je knihovna 4 implementovaných komponent z APG návrhových vzorů v Solid.js, rozšiřující jeho ekosystém a navazující na práci v rámci Solid Aria.

Na to navazující dokumentace[1], která popisuje použití komponent a demo[2] aplikace pro jejich ukázku. Dále pak vygenerovaný report z testování[3], report o pokrytí testy[4] a typová dokumentace[5].

9.1 Budoucí vývoj

V budoucnu bych chtěl pokračovat v rozšiřování této knihovny o další komponenty. Chtěl bych ze zaměřit na podporu RTL[6] jazyků a internacionalizaci komponent do jiných jazyků než je angličtina.

Rád bych rozšířil oficiální podporu o další populární odečítače obrazovek jako je JAWS nebo NVDA. Je možné, že momentální implementace funguje s výše zmíněnými odečítači obrazovek, ale z důvodu neotestování to nemohu potvrdit.

Jak bylo zmíněno v kapitole 8.2 o testování, rád bych do budoucna zautomatizoval testování knihovny pomocí E2E testů.

V neposlední řadě bych rád vyměnil do budoucna demo aplikaci za Storybook [7], který umožňuje dynamicky měnit vstupní props a lze tak jednodušeji vyzkoušet různé chování komponenty.

[1] `https://solid-apg-docs.vercel.app`
[2] `https://solid-apg-app.vercel.app`
[3] `https://susickypavel.github.io/solid-apg/report`
[4] `https://susickypavel.github.io/solid-apg/coverage`
[5] `https://susickypavel.github.io/solid-apg/typedoc`
[6] Jazyky pro které směr čtení textu je zprava doleva
[7] `https://storybook.js.org`

Přílohy

Příloha A

Seznam literatury

1. META. *React* [online]. [cit. 28. 11. 2023].
   Dostupné z: `https://react.dev`.
2. EVAN YOU. *Vue* [online]. [cit. 28. 11. 2023].
   Dostupné z: `https://vuejs.org`.
3. RYAN CARNIATO. *Solid* [online]. [cit. 28. 11. 2023].
   Dostupné z: `https://solidjs.com`.
4. RICH HARRIS. *Svelte* [online]. [cit. 28. 11. 2023].
   Dostupné z: `https://svelte.dev`.
5. W3C. *Web Accessibility Initiative (WAI)* [online]. [cit. 28. 11. 2023].
   Dostupné z: `https://www.w3.org/mission/accessibility`.
6. W3C. *Diverse Abilities and Barriers* [online]. [cit. 09. 01. 2024].
   Dostupné z:
   `https://www.w3.org/WAI/people-use-web/abilities-barriers`.
7. W3C. *Accessible Rich Internet Applications (WAI-ARIA)* [online].
   1.2. vyd. 06/2023. [cit. 05. 12. 2023].
   Dostupné z: `https://www.w3.org/TR/wai-aria`.
8. W3C. *Implicit WAI-ARIA Semantics* [online]. [cit. 08. 12. 2023].
   Dostupné z:
   `https://www.w3.org/TR/wai-aria/#implicit_semantics`.
9. W3C. *Required Owned Elements* [online]. [cit. 10. 12. 2023]. Dostupné z:
   `https://www.w3.org/TR/wai-aria/#required_owned_elements`.
10. W3C. *Standard Guidelines for Required Owned Elements* [online].
    [cit. 10. 12. 2023]. Dostupné z: `https://www.w3.org/WAI/standards-guidelines/act/rules/bc4a75/proposed`.

11. W3C. *Categorization of Roles* [online]. [cit. 14. 12. 2023]. Dostupné z: https://www.w3.org/TR/wai-aria/#roles_categorization.
12. MOZILLA FOUNDATION. *ARIA attributes - aria-pressed* [online]. [cit. 14. 12. 2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Attributes/aria-pressed.
13. W3C. *Web Content Accessibility Guidelines (WCAG) 2.1* [online]. [cit. 03. 12. 2023]. Dostupné z: https://www.w3.org/TR/WCAG21/#requirements-for-wcag-2-1.
14. W3C. *How to Meet WCAG (Quick Reference)* [online]. [cit. 10. 05. 2024]. Dostupné z: https://www.w3.org/WAI/WCAG22/quickref.
15. W3C. *ARIA Authoring Practices Guide* [online]. [cit. 01. 01. 2024]. Dostupné z: https://www.w3.org/WAI/ARIA/apg.
16. W3C. *Web accessibility principles* [online]. [cit. 01. 12. 2023]. Dostupné z: https://www.w3.org/WAI/fundamentals/accessibility-principles.
17. W3C. *Keyboard accessibility* [online]. [cit. 01. 12. 2023]. Dostupné z: https://www.w3.org/WAI/WCAG22/Understanding/keyboard.html.
18. ING. IVO MALÝ PH.D. *Text entry, Gestures, Voice User Interface, Accessibility* [online]. [cit. 03. 12. 2023]. Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/385769/course/section/66968/pda_lecture_11_gestures_voice_text_entry-v23a.pdf.
19. W3C. *Pointer Gestures* [online]. [cit. 03. 12. 2023]. Dostupné z: https://www.w3.org/WAI/WCAG22/Understanding/pointer-gestures.html.
20. W3C. *Target size* [online]. [cit. 03. 12. 2023]. Dostupné z: https://www.w3.org/WAI/WCAG22/Understanding/target-size-enhanced.html.
21. WEBAIM. *WebAIM Screen Reader User Survey* [online]. 2024. [cit. 07. 04. 2024]. Dostupné z: https://webaim.org/projects/screenreadersurvey10.
22. RYAN CARNIATO. *Reactivity* [online]. [cit. 05. 03. 2024]. Dostupné z: https://www.solidjs.com/guides/reactivity#how-it-works.

23. STEFAN KRAUSE.
Results for js web frameworks benchmark - Chrome 120 [online].
[cit. 10. 01. 2024]. Dostupné z: `https://krausest.github.io/js-framework-benchmark/2023/table_chrome_120.0.6099.62.html`.

24. STEFAN KRAUSE.
Results for js web frameworks benchmark - Chrome 122 [online].
[cit. 10. 01. 2024]. Dostupné z: `https://krausest.github.io/js-framework-benchmark/2024/table_chrome_122.0.6261.69.html`.

25. THE SVELTE TEAM. *Introducing runes* [online]. [cit. 06. 03. 2024].
Dostupné z: `https://svelte.dev/blog/runes`.

26. ASTRO. *Astro Islands* [online]. [cit. 10. 05. 2024].
Dostupné z: `https://docs.astro.build/en/concepts/islands`.

27. BEN HOLMES.
Understanding the JavaScript Meta-framework Ecosystem [online].
2024. [cit. 16. 04. 2024]. Dostupné z: `https://prismic.io/blog/javascript-meta-frameworks-ecosystem`.

28. SACHA GREIF. *State of JS* [online]. 2022. [cit. 13. 03. 2024]. Dostupné z: `https://2022.stateofjs.com/en-US/usage/#js_ts_balance`.

29. ADOBE.
Craft world-class accessible components with custom styles [online].
2024. [cit. 02. 05. 2024].
Dostupné z: `https://react-spectrum.adobe.com/react-aria`.

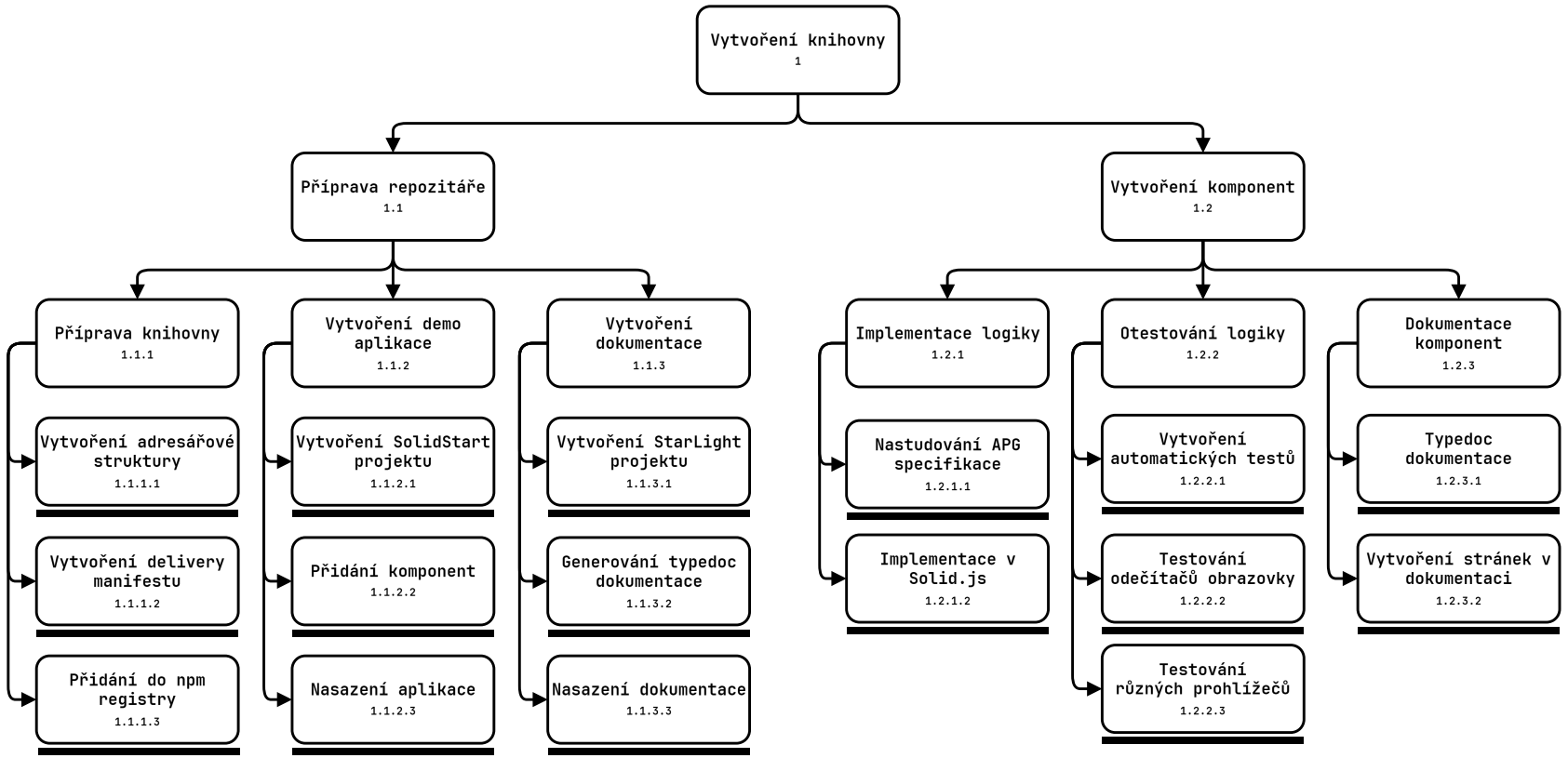
30. SEGUN ADEBAYO. *Chakra UI* [online]. 2024. [cit. 02. 05. 2024].
Dostupné z: `https://v2.chakra-ui.com/docs/components`.

31. BARRY KIRWAN AND L. K. AINSWORTH.
A Guide To Task Analysis. 1st Edition.
London: Taylor & Francis Group, 1992. ISBN 9780429173585.

32. VERCEL. *Edge Network* [online]. [cit. 10. 05. 2024].
Dostupné z: `https://vercel.com/docs/edge-network/overview`.

Příloha B

Obrázky



Obrázek B.1: HTA diagram (diagram autora)

Příloha C

Tabulky

Komponenta	Solid Aria	Kobalte	Existence
Accordion	Ano	Ano	Ano
Alert	—	Ano	Ano
Breadcrumb	Ano	Ano	Ano
Button	Ano	Ano	Ano
Carousel	—	—	**Ne**
Checkbox	Ano	Ano	Ano
Combobox	—	Ano	Ano
Dialog	Ano	Ano	Ano
Disclosure	—	(Collapsible)	Ano
Feed	—	—	**Ne**
Grid	—	—	**Ne**
Listbox	Ano	(Select)	Ano
Menu	Ano	Ano	Ano
Menubar	—	Ano	Ano
Meter	Ano	(Progress)	Ano
Radio Group	Ano	Ano	Ano
Slider	—	Ano	Ano
SpinButton	—	(NumberField)	Ano
Switch	Ano	Ano	Ano
Toolbar	—	—	**Ne**
Tabs	Ano	—	Ano
Tooltip	—	Ano	Ano
Tree View	Ano	—	Ano
Treegrid	—	—	**Ne**

Tabulka C.1: Tabulka implementovaných komponent v Solid.js ekosystému

Příloha D

Ukázky kódu

```
const Disclosure = (props) => {
    const {
        togglеProps,
        contentProps,
        state
    } = createDisclosure(props);

    return (
        <div>
            <button {...togglеProps}>Toggle</button>

            {state.isVisible ? (
                <div {...contentProps}>
                    {props.children}
                </div>
            ) : null}
        </div>
    );
};
```

Ukázka kódu D.1: Ukázka použití createDisclosure funkce

```
const SpinButton = () => {
    const {
        inputProps,
        incrementButtonProps,
        decrementButtonProps,
        state
    } = createSpinButton({
        values: [1, 2, 3, 4, 5, 6, 7],
        mapping: [
            "monday",
            "tuesday",
            "wednesday",
            "thursday",
            "friday",
            "saturday",
            "sunday"
        ],
        step: 2
    });

    return (
        <div>
            <button {...incrementButtonProps}>up</button>
            <input {...inputProps} value={state.value} />
            <button {...decrementButtonProps}>down</button>
        </div>
    );
};
```

Ukázka kódu D.2: Ukázka použití createSpinButton funkce

```
const Toolbar = (props) => {
    let toolbar;

    const {
        toolbarProps
    } = createToolbar(props, () => toolbar);

    return (
        <div ref={toolbar} {...toolbarProps}>
            <button onClick={() => alert("Bold")}>
                Bold
            </button>

            <button onClick={() => alert("Italics")}>
                Italics
            </button>

            <button
                disabled
                onClick={() => alert("Underline")}
            >
                Underline
            </button>
        </div>
    );
};
```

Ukázka kódu D.3: Ukázka použití createToolbar funkce

```
const Tooltip = (props) => {
    const [local, rest] = splitProps(props, ["state"]);
    const {
        tooltipProps
    } = createTooltip(props, () => local.state);

    return (
        <div {...tooltipProps}>{props.children}</div>
    );
};

export const TooltipTrigger = (props) => {
    let trigger;

    const state = createTooltipTriggerState(props);
    const {
        tooltipProps,
        triggerProps
    } = createTooltipTrigger(props, state, () => trigger);

    return (
        <button ref={trigger} {...triggerProps}>
            Hover or Focus me
            {state.isOpen ? (
                <Tooltip {...tooltipProps} state={state}>
                    {props.children}
                </Tooltip>
            ) : null}
        </button>
    );
};
```

Ukázka kódu D.4: Ukázka vytvoření Tooltipu

```
---
title: Toolbar
description: a primitive for creating a Toolbar component
sidebar:
    badge:
        text: v0
        variant: success
---

<Links github="https://example.com" />

## Anatomy

<Anatomy viewBox="0 0 64 64" alt="0" caption="Example">
    <ToolbarAnatomy />
</Anatomy>

## API Reference

<PropTable name="ToolbarArguments" />

## Example

<Toolbar client:idle />
```

Ukázka kódu D.5: Ukázka stránky dokumentace psané v MDX

Příloha E

Použitý software

E.1 Projekt

Veškeré použité závislosti jsou dohledatelné v repozitáři[1] v souborech `package.json`.

E.2 Text

- **diagrams.net**[2] — online nástroj pro tvorbu diagramů.
- **Figma**[3] — online nástroj pro tvorbu wireframů a designů.
- **TinyTex**[4] — zmenšená distribuce LaTeX.

E.3 Umělá inteligence

- **ChatGPT**[5] — využito obecně k dohledávání informací při debugování a testování kódu.
- **GitHub Copilot**[6] — využito pro doplňování kódu.
- **Grammarly**[7] — využito pro korekturu textu.
- **Perplexity**[8] — využito pro rešerši.

E.4 Ostatní

- **Notion**[9] — správa semestrální práce a sledování úkolů.
- **Visual Studio Code**[10] — editor pro psaní kódu i textu.
- **Fork**[11] — Git GUI klient pro správu projektových repozitářů.

[1] `https://github.com/susickypavel/solid-apg`
[2] `https://app.diagrams.net`
[3] `https://figma.com`
[4] `https://yihui.org/tinytex`
[5] `https://chat.openai.com`
[6] `https://github.com/features/copilot`
[7] `https://grammarly.com`
[8] `https://perplexity.ai`
[9] `https://notion.so`
[10] `https://code.visualstudio.com`
[11] `https://git-fork.com`

Příloha F

Odkazy

Dokumentace
https://solid-apg-docs.vercel.app

Demo
https://solid-apg-app.vercel.app

Vitest report
https://susickypavel.github.io/solid-apg/report

Code coverage
https://susickypavel.github.io/solid-apg/coverage

Typedoc dokumentace
https://susickypavel.github.io/solid-apg/typedoc

Příloha G

Zdrojový kód

Struktura každého adresáře v projektu je komplexní a nevešla by se do jednoho diagramu, proto zbytek popisu je v přidruženém souboru `README.md`.

```
/
├── apps
│   ├── docs (zdrojový kód dokumentace)
│   │   ├── ...
│   │   └── README.md
│   └── solid-start-example (zdrojový kód demo aplikace)
│       ├── ...
│       └── README.md
├── packages
│   ├── solid-apg (zdrojový kód komponent)
│   │   ├── ...
│   │   └── README.md
│   └── solid-aria (git submodule)
├── ...
└── README.md
```