Diplomová práce

SBÍRKA ZNOVUPOUŽITELNÝCH NUXT KOMPONENT

Bc. Vojtěch Moravec

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce
Vedoucí: Ing. Oldřich Malec
19. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

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.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 19. května 2024 .....

Abstrakt

Tato práce se zaměřuje na návrh a vývoj sbírky znovupoužitelných komponent pro webové aplikace využívající framework Nuxt. Cílem je vytvořit kolekci komponent, které zjednoduší začátek vývoje a zároveň budou jednoduše rozšiřitelné a upravitelné. Práce se nezaměřuje pouze na samotné komponenty, ale také na jejich dokumentaci a další nástroje podporující celý životní cyklus aplikace od návrhu až po nasazení.

K dispozici jsou i rozsáhlejší sekce, skládající se z těchto komponent. Vývojáři tak mohou využít již předpřipravená řešení, která si mohou dle potřeby upravit, což značně urychluje proces vývoje.

Pro designéry je připravena knihovna ve Figmě, umožňující jim poskytovat vývojářům návrhy, které lze replikovat v přesné shodě s originálem. Vývojáři mohou využít několik způsobů instalace komponent a to sice pomocí kopírování kódu nebo pomocí CLI.

Klíčová slova UI, UX, DX, Nuxt, Vue, komponenty, sbírka komponent, znovupoužitelnost

Abstract

This thesis focuses on the design and development of a collection of reusable components for web applications using the Nuxt framework. The goal is to create a collection of components that simplify the start of development and are easily expandable and modifiable. The work focuses not only on the components themselves but also on their documentation and other tools supporting the entire application lifecycle from design to deployment.

More extensive sections composed of these components are also available. Developers can thus utilize pre-prepared solutions, which they can modify as needed, significantly accelerating the development process.

For designers, a library in Figma has been prepared, allowing them to provide developers with designs that can be replicated in exact accordance with the original. Developers have several ways to install components, either by copying code or using a CLI.

Keywords UI, UX, DX, Nuxt, Vue, components, collection of components, reusable

Seznam zkratek

API	Application Programming Interface
CDN	Content Delivery Network
CLI	Command Line Interface
CSS	Cascading Style Sheets
CI/CD	Continuous Integration/Continuous Deployment
CTA	Call to Action
DNS	Domain Name System
DOM	Document Object Model
FAQ	Frequently Asked Questions
FE	Frontend
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IT	Informační technologie
JS	JavaScript
JSON	JavaScript Object Notation
NPM	Node Package Manager
PC	Personal Computer
SEO	Search Engine Optimization
SPA	Single Page Application
SSL	Secure Sockets Layer
SSR	Server Side Rendering
UI	User interface
UX	User experience
URL	Uniform Resource Locator
WAF	Web Application Firewall
YAML	YAML Ain't Markup Language
XML	Extensible Markup Language

Slovník

Accordion	Rozbalovací menu
Auth	Autentizace
Auto-scroll	Automatické posouvání
Badge	Odznak
Build	Sestavení aplikace do finální podoby
Button	Tlačítko
Card	Karta
Carousel	Posuvný prvek
Components	Komponenty
Composable	Znovupoužitelná funkce zachovávající stav
Contact	Kontakt
Container	Kontejner
Checkbox	Zaškrtávací políčko
Dashboard	Stránka s přehledem
Deploy	Nasazení aplikace do testovacího či prostředí
Dropdown	Rozbalovací menu
Endpoint	Bod API, který obsahuje logiku a odpovídá daty
Error	Chyba
Features	Funkce
Flexbox	CSS technologie pro uspořádání prvků na stránce
Fork	Vytvoření kopie repozitáře
Framework	Softwarové řešení pro podporu programování, vývoje a organizaci jiných softwarových projektů
Frontend	Část aplikace, kterou vidí uživatel a reaguje s ní
Getting started	První kroky
Headless	Aplikace, která nemá vlastní frontend a komunikuje pouze přes API
Header	Hlavička
Hero sekce	Úvodní sekce
Highlighter	Zvýrazňovač
Hook	Volání funkce
Hosting	Služba hostování webové stránky
Hot reload	Automatické načítání změn
Hover	Najetí myši na prvek
Icon	Ikona
Inject	Vložit
Issues	Problémy v repozitáři
Loading	Načítání
Landing page	Úvodní stránka
Layout	Rozložení stránky
Metaframework	Framework postavený nad jiným frameworkem
Notification	Oznámení
Open source	Otevřený zdrojový kód (kdokoliv k němu může přistoupit a zároveň do něj přispět)
Pagination	Stránkování
Patch	Oprava chyby
Pattern matching	Porovnávání vzorů
Pop-up	Vyskakovací okno
Popover	Plovoucí okno
Pull request	Žádost o změnu ve zdrojovém kódu
Push	Nahrání změněného kódu do repozitáře

Prop	Vlastnost komponenty
Props	Vlastnosti komponenty
Provide	Poskytovat
References	Reference
Section	Sekce
Select	Prvek pro výběr
Sidebar	Postranní panel
Slot	Místo pro vložení obsahu
Slots	Místa pro vložení obsahu
Switch	Přepínač
Table	Tabulka
Tabs	Karty
Textarea	Víceřádkové textové pole
Text input	Jednořádkové textové pole
Tooltip	Pomocný text při najetí myši na prvek
Tree shaking	Odstranění nepoužitých částí kódu
Utility-first	Přístup k tvorbě stylů, kde se využívají jednoduché a malé třídy (tedy dává přednost funkčnosti)
Wireframe	Drátěný model
Wrapper	Obal

Kapitola 1
Úvod

UI knihovny jsou nezbytné nástroje ve světě webového vývoje. Poskytují sadu předdefinovaných komponent, které vývojářům umožňují rychle a efektivně vytvářet atraktivní a funkční uživatelská rozhraní. Tyto knihovny jsou důležitou součástí vývoje aplikací, protože usnadňují implementaci složitých UI prvků.

Tato práce se zabývá analýzou různých aspektů UI knihoven, včetně jejich designu, funkcí, a toho, jak tyto knihovny podporují různé části vývoje aplikace. Cílem je poskytnout ucelený pohled na UI knihovny, jejich výhody, výzvy a *best practices* pro jejich využití ve webovém vývoji.

Kapitola 2

Cíl

Cílem této práce je vytvořit sadu znovupoužitelných a snadno modifikovatelných komponent pro webové aplikace postavené na frameworku Nuxt. Tato kolekce komponent má usnadnit počáteční fázi vývoje, nabízet možnosti rozšíření a přizpůsobení podle individuálních potřeb. Práce se neomezuje pouze na vývoj samotných komponent, ale zahrnuje také jejich podrobnou dokumentaci a nástroje, které podporují všechny fáze tvoření aplikace.

Nejdříve je v práci připravena teoretická část, která se zaměřuje na popis zvolených technologií a dalších řešení. Dále se práce zabývá návrhem a vývojem komponent a jejich dokumentace. Nakonec je nastíněn možný budoucí vývoj a rozšíření.

Kapitola 3
Analýza

3.1 Úvod

V této kapitole bude provedena důkladná analýza současných řešení a technologií používaných pro vývoj UI knihoven/kolekcí. Hlavním cílem analýzy je posoudit, které přístupy a technologie nejlépe vyhovují potřebám vývoje kolekce komponent určených pro *framework* Nuxt, usnadňujících proces kopírování a implementace kódových úseků do cílových aplikací.

První část analýzy bude zaměřena na přehled a hodnocení stávajících řešení, která mohou sloužit jako inspirace pro kolekci. Zahrnuty budou knihovny vytvořené pro různé *frameworky*, jako jsou React a Vue. Prozkoumány budou jejich vlastnosti, možnosti integrace a údržby.

Dále budou porovnány dva hlavní přístupy k integraci komponent do koncových projektů: distribuce jako samostatné balíčky (závislosti) a přímé vložení kódu do projektů. Srovnání umožní lepší pochopení dopadů každého přístupu na udržovatelnost, aktualizace a bezpečnost projektu.

Rovněž bude rozhodnuto, zda bude pro vývoj komponent preferován jazyk JavaScript nebo TypeScript, a zhodnoceno, jak volba jazyka ovlivňuje vývojový proces a integraci komponent.

Analýza závislostí jednotlivých komponent na externích knihovnách je dalším bodem, který bude zvážen, aby bylo možné maximalizovat jejich využití a minimalizovat potenciální problémy v budoucích implementacích.

Závěrem kapitoly bude definován dopad zjištěných informací na design a strukturu kolekce komponent, čímž bude kapitola připravena pro hlubší prozkoumání a evaluaci technologií a metod, které budou základem pro další vývoj.

3.2 Existující řešení

Tato část se zaměřuje na analýzu a srovnání existujících řešení ve světě UI knihoven. Tato řešení můžeme kategorizovat do dvou hlavních skupin, které se liší způsobem integrace a možnostmi úprav, což má zásadní dopad na flexibilitu a udržovatelnost výsledných aplikací.

První skupina zahrnuje knihovny poskytující sadu komponent, které jsou distribuovány jako balíčky. Tyto komponenty se do projektů přidávají jako závislosti, čímž se usnadňuje jejich správa a aktualizace. Uživatelé tyto komponenty importují do svých projektů a používají v kódu podle potřeby. Stylování těchto komponent je často omezeno na předpřipravené proměnné nebo API, které nabízí jen omezené možnosti úprav. Pokud je potřeba provést zásadnější změny v chování nebo vzhledu, může to vést k „ohybání“ kódu, což někdy vyústí v těžko udržovatelné řešení.

Druhá skupina se skládá z knihoven, které nejsou primárně určeny k distribuci jako balíček. Místo toho je jejich kód navržen tak, aby byl přímo zkopírován do cílového projektu, nebo jsou k dispozici prostřednictvím CLI nástrojů, které usnadňují přidání komponent jednoduchými

příkazy. Tento přístup je zaměřen na maximální znovupoužitelnost a rozšířitelnost kódu, což umožňuje uživatelům lépe přizpůsobit komponenty svým specifickým potřebám.

V následujících podsekcích bude prozkoumáno, jak tyto dva přístupy ovlivňují vývojový proces, a budou identifikovány klíčové výhody a výzvy spojené s každou skupinou. Tato analýza poskytne důležité informace pro navrhování vlastní kolekce komponent, která by měla kombinovat to nejlepší z obou světů a zároveň adresovat specifické požadavky a omezení spojená s *frameworkem* Nuxt.

3.2.1 Knihovny s principem balíčkování

Knihovny s principem balíčkování představují jeden z nejběžnějších způsobů, jakým vývojáři spravují a integrují komponenty do svých webových aplikací. Tento přístup nabízí několik výhod, které ho činí atraktivním pro mnoho projektů, zejména pro ty, které vyžadují rychlý vývoj a udržování konzistence napříč velkými týmy.

Komponenty distribuované jako balíčky jsou obvykle spravovány prostřednictvím správců balíčků jako npm, yarn nebo pnpm pro JavaScriptové projekty. Tyto nástroje zjednodušují proces instalace, aktualizace a správy závislostí. Když je vydána nová verze komponenty, může být snadno aktualizována ve všech projektech, které ji využívají, což zajišťuje, že všechny aplikace mohou rychle získat přístup k novým funkcím a opravám chyb.

Použití komponent jako závislostí také podporuje konzistenci v designu a funkcionalitě. Knihovny jako Nuxt UI nebo Radix UI nabízí sadu dobře navržených komponent, které dodržují konzistentní designové prvky. To je zvláště užitečné pro týmy, které pracují na rozsáhlých projektech, protože zaručuje, že všechny komponenty v aplikaci budou vizuálně a funkčně sladěné.

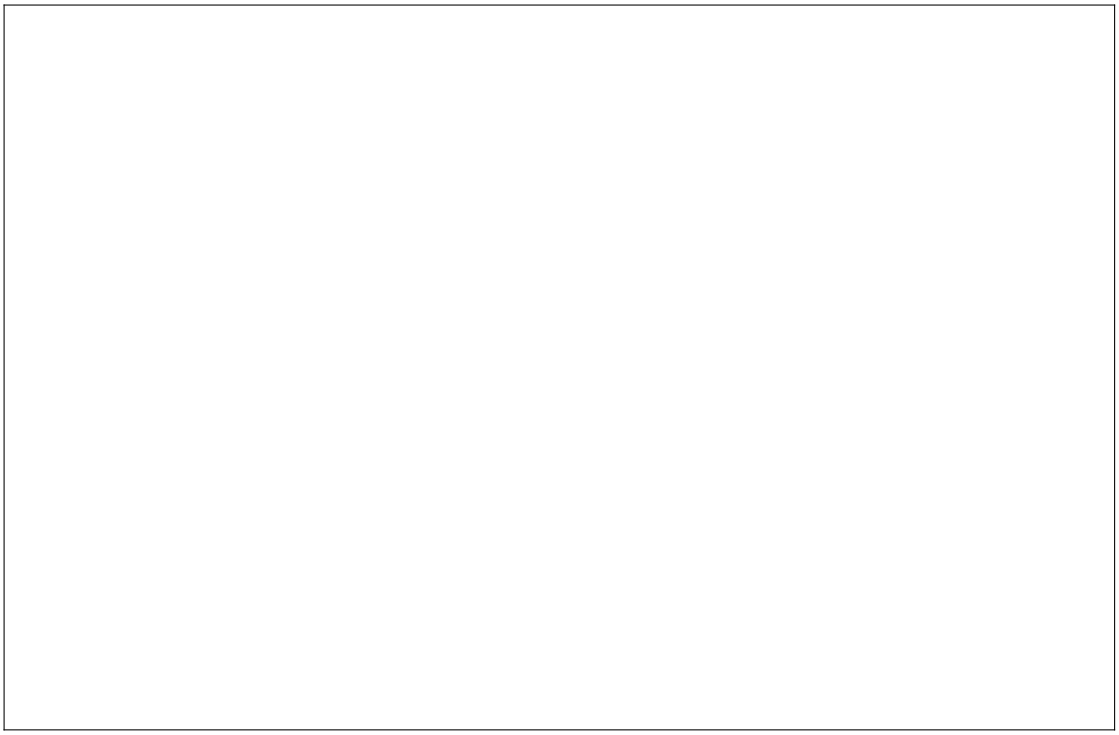
Díky používání balíčkových knihoven mohou týmy výrazně snížit množství duplicitního kódu v projektu. Vývojáři nemusí vytvářet základní komponenty od nuly, ale mohou využít již existující a otestované komponenty, což šetří čas a zdroje. Tato efektivita umožňuje týmům zaměřit se na specifické aspekty svých projektů místo na znovu vynalézání běžných řešení.

Ačkoli přístup balíčkování nabízí mnoho výhod, přináší i některé omezení. Úpravy stylů a funkcionality jsou často omezeny na to, co knihovna explicitně povoluje prostřednictvím API nebo konfiguračních možností. Pokud projekt vyžaduje výrazné úpravy komponent, může být nutné „ohýbat“ kód, což může vést k neudržitelným a obtížně spravovatelným řešením. Tento problém je obzvláště výrazný v projektech, které vyžadují odklon od předpřipravených stylů nebo specifické funkcionality, které standardní komponenty nenabízí.

Nuxt UI

Jeden ze zástupců knihoven s principem balíčkování je Nuxt UI. Nuxt UI je relativně nová knihovna komponent, specificky navržená pro rychlou a pohodlnou tvorbu aplikací. Původně vznikla jako interní knihovna pro NuxtLabs [1], kteří s její pomocí vytvořili např. aplikaci Volta na pomoc vývojářům efektivně spravovat GitHub repozitáře. [2]

Nuxt UI nabízí bohatou sadu komponent, které jsou optimalizovány pro snadné použití s *frameworkem* Nuxt. Tyto komponenty jsou navrženy tak, aby poskytovaly jednotný vzhled a zároveň umožňují jistou míru přizpůsobení. Knihovna je postavena na principu „plug and play“, což znamená, že komponenty lze snadno začlenit do projektů bez nutnosti rozsáhlé konfigurace. To umožňuje vývojářům rychle prototypovat a iterovat designy, aniž by museli obětovat kontrolu nad finálním produktem.



Obrázek 3.1 Komponenty Nuxt UI

Využití v projektech

Nuxt UI je ideální knihovnou pro projekty, kde je primární důraz kladen na funkčnost a efektivitu vývoje, a kde výrazný individuální branding není prioritou. Tato knihovna nabízí řadu dobře promyšlených komponent, které jsou ihned připraveny k použití a pokrývají široké spektrum běžných potřeb webových aplikací.

Typy aplikací

Knihovna je obzvláště vhodná pro vývoj aplikací jako jsou *dashboardy*, administrační panely, nebo interní nástroje, kde je důraz kladen spíše na funkčnost než na unikátní design. V těchto aplikacích je často potřeba rychle zobrazit velké množství dat přehledně a efektivně, a přednastavené komponenty jako tabulky, grafy a formuláře, které Nuxt UI nabízí, mohou výrazně urychlit vývoj a zjednodušit údržbu.

Robustní základ

Pro projekty, které nevyžadují specifický vizuál, poskytuje Nuxt UI robustní základ, který minimalizuje potřebu vlastních úprav. Tím se snižují náklady na vývoj a zvyšuje se konzistence uživatelského rozhraní napříč aplikací. Uživatelé, kteří očekávají konzistentní a funkčně bohaté rozhraní, ocení, že aplikace „funguje jak má“ bez zbytečných komplikací nebo zpoždění způsobených potřebou výrazně přizpůsobovat základní prvky.

Předpřipravené šablony

Nuxt UI nabízí řadu předpřipravených šablon, které jsou k dispozici prostřednictvím jejich „Pro“ plánu. Tyto šablony jsou navrženy tak, aby poskytovaly vývojářům rychlý start a efek-

tivní nástroje pro běžné aplikace a scénáře použití. S šablonami, jako jsou administrační panely, dokumentace nebo *landing page* umožňuje Nuxt UI vývojářům výrazně urychlit vývoj tím, že eliminují potřebu vytvářet často používané stránky od nuly. Každá šablona je plně responzivní, optimalizovaná pro různé typy zařízení a připravená k okamžitému nasazení. Tímto způsobem šablony nabízí nejen zrychlení vývoje, ale i zajištění vysoké úrovně kvality a uživatelské zkušenosti, které jsou důležité pro úspěšnou implementaci moderních webových aplikací. [3]

Omezené úpravy

Přestože knihovna Nuxt UI nabízí široké možnosti pro konfiguraci stylů svých komponent, v praxi se může ukázat, že některé typy úprav nejsou přímočaré a vyžadují větší úsilí než bylo očekáváno. Tento problém může být zvláště patrný, když vývojáři chtějí provést specifické změny, které nejsou přímo podporovány základním API knihovny. V důsledku toho mohou být nuceni používat obcházení standardních metod, což může vést k méně udržitelnému kódu a potenciálně k narušení konzistence a funkčnosti aplikace.

Jedním z příkladů, kde mohou vývojáři narazit na obtíže, je pokus o radikální změnu vzhledu komponent, které Nuxt UI poskytuje jako standardní. Ačkoliv knihovna umožňuje základní úpravy jako změny barev, fontů a odsazení prostřednictvím předdefinovaných proměnných, hlubší zásahy do *layoutu* nebo chování komponent mohou vyžadovat přímé zásahy do CSS nebo struktury komponent. To může být časově náročné a komplikované, zejména pokud se tyto změny snaží zachovat responzivitu a přístupnost komponent.

Figma Kit

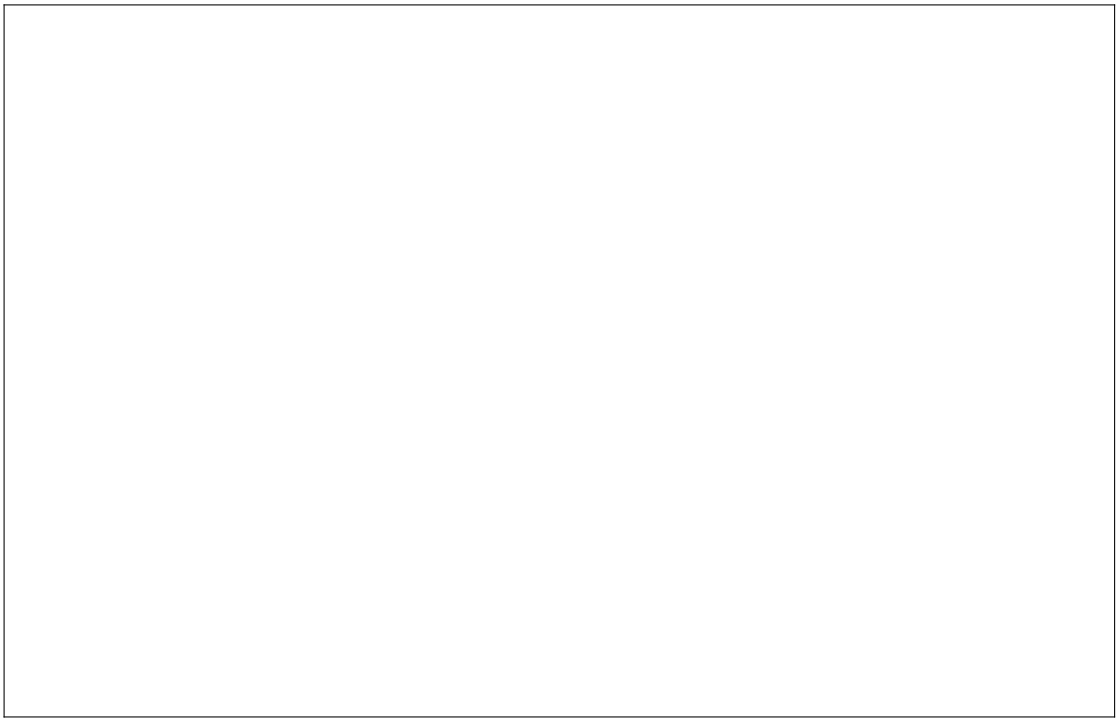
Knihovna nabízí také Figma Kit, který umožní designérům snadno vytvářet prototypy a designové systémy s komponentami Nuxt UI. Tím se zajišťuje, že design a implementace jsou vzájemně sladěny a že výsledná aplikace bude vizuálně konzistentní a uživatelsky přívětivá. [4]

Hodnocení

- **Výhody**
  - Velké množství komponent
  - Propracované styly
  - Připravené pro použití
- **Nevýhody**
  - Omezené možnosti rozšíření
  - Složitější úpravy stylů
  - Složité úpravy funkcionality

Radix UI

Radix UI představuje *open source* knihovnu UI komponent, která je zaměřena na tvorbu kvalitních a přístupných designových systémů a webových aplikací. Jde především o Radix Primitives, což jsou nízkoúrovňové UI komponenty s důrazem na přístupnost a možnost úprav vývojářů za cílem vytvoření vlastní knihovny. Tyto komponenty lze využívat buď jako základní vrstvu designového systému nebo je postupně implementovat do stávajících projektů. [5]



Obrázek 3.2 Primitivní komponenty Radix UI

Jedním z hlavních rysů Radix UI je jeho důraz na přístupnost. Všechny komponenty jsou navrženy tak, aby splňovaly standardy WAI-ARIA a poskytovaly bezproblémovou podporu pro pohyb pomocí klávesnice a asistenční technologie. Tento přístup zajišťuje, že aplikace vyvinuté s Radix UI budou přístupné širšímu spektru uživatelů, což je důležité pro vytváření inkluzivních digitálních produktů.

Radix UI se odlišuje od mnoha jiných UI knihoven tím, že neobsahuje žádné předdefinované styly. Místo toho poskytuje primitivní komponenty, které lze použít jako základní stavební bloky pro vlastní komponenty. Tyto primitivní komponenty jsou navrženy tak, aby byly co nejvíce konfigurovatelné, což umožňuje vývojářům vytvářet unikátní uživatelská rozhraní bez omezení specifickými designovými rozhodnutími.

Radix UI nabízí širokou nabídku komponent, které pokrývají vše od základních prvků, jako jsou tlačítka a přepínače, až po složitější kontrolní prvky, jako jsou dialogová okna a *dropdown* menu. Každá komponenta je navržena tak, aby poskytovala optimální uživatelskou zkušenost a byla snadno integrovatelná do různých projektů. To vývojářům umožňuje rychle sestavit funkcionalitu, kterou potřebují, zatímco si udržují úplnou kontrolu nad chováním a vzhledem aplikace.

Na druhou stranu jedna z hlavních nevýhod Radix UI spočívá v tom, že vývojáři musí vynaložit značné úsilí při vytváření konkrétních komponent, protože Radix UI poskytuje pouze primitivní komponenty, které slouží jako základní stavební bloky. Komponenty neobsahují žádné předdefinované styly ani komplexní funkcionality, což znamená, že na vývojářích leží úkol implementovat vizuální design, interakce a další specifické aspekty, které jsou potřebné pro jejich

projekt. Tento přístup vyžaduje pokročilé znalosti v CSS a správné porozumění nejlepším praktikám přístupnosti a interaktivity, aby bylo možné plně využít potenciál primitivních komponent. Ačkoliv tento model poskytuje maximální flexibilitu, může být také časově náročný a může zpomalit vývoj, zejména ve fázích, kdy je potřeba rychle prototypovat nebo když tým nemá dostatek zdrojů na detailní vývoj každé komponenty od základů.

Hodnocení

- **Výhody**
  - Velké množství primitivních komponent
  - Promyšlený koncept
- **Nevýhody**
  - Velká část implementace spadá na vývojáře

3.2.2 Knihovny s principem vlastnění kódu

Knihovny s principem vlastnění kódu poskytují vývojářům jedinečný přístup k použití a úpravě komponent ve svých projektech. Tento model se liší od běžnějších knihoven tím, že vývojáři mají možnost přímo manipulovat s kódem komponent, což jim umožňuje provádět hluboké a specifické změny, které by jinak byly obtížné nebo nemožné.

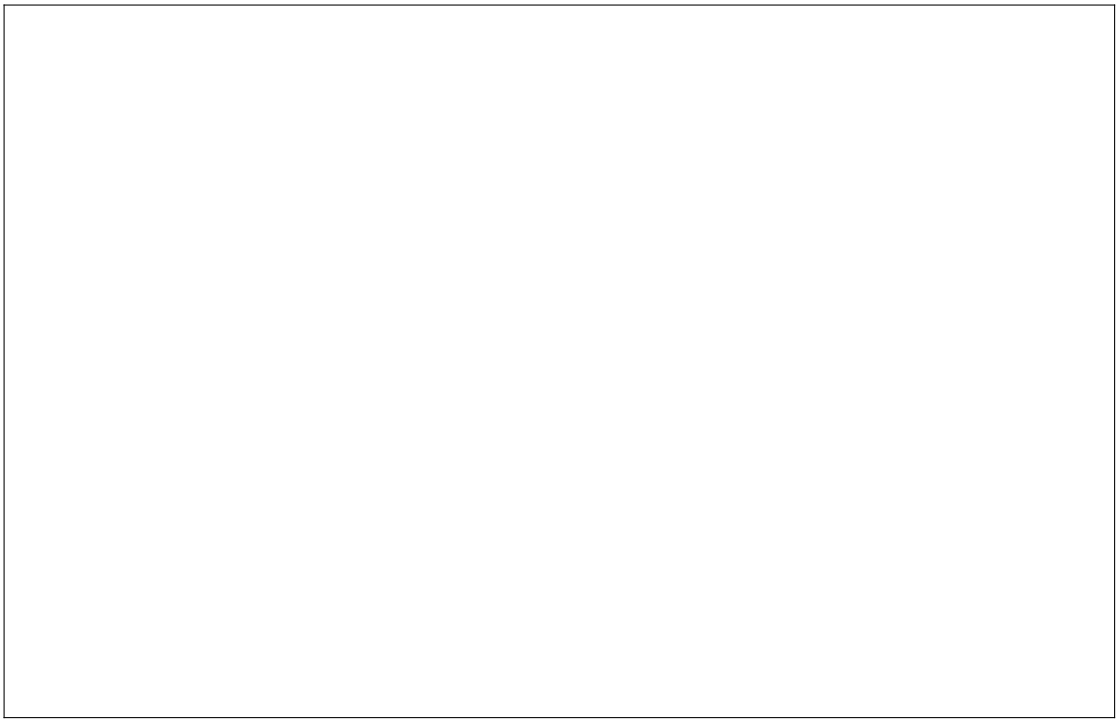
Hlavní výhodou tohoto přístupu je extrémní flexibilita. Vývojáři nejsou omezeni předdefinovaným API nebo funkčností komponenty. Místo toho mohou přímo upravit zdrojový kód komponenty, aby lépe vyhovoval specifickým potřebám jejich aplikace. Tato schopnost je obzvláště cenná v projektech, kde je potřeba pečlivě laděná funkcionalita nebo když standardní komponenty jednoduše neodpovídají požadovaným designovým nebo technickým specifikacím.

Projekty vyžadující vysokou míru přizpůsobení, mohou zvláště těžit z modelu vlastnění kódu. Tento přístup je také vhodný pro vývojové týmy, které mají silné znalosti a zkušenosti s daným programovacím jazykem a *frameworkem*, neboť jim umožňuje plně využít své dovednosti bez omezení.

Přestože poskytuje vysokou míru kontroly, vlastnění kódu také přináší určité výzvy. Správa vlastního kódu může být časově náročnější, protože tým musí zajišťovat údržbu a aktualizace komponent, což by jinak mohlo být delegováno na externí knihovny. Tento model také může vést k větší fragmentaci kódu, pokud každý projekt vyžaduje unikátní úpravy, což komplikuje možnost sdílení kódu mezi projekty nebo při práci ve velkých týmech. Tento problém se dá v rámci *frameworku* Nuxt řešit pomocí vlastních modulů, které umožňují sdílení kódu mezi projekty - např. v rámci monorepozitáře.

shadcn/ui

Shadcn/ui je inovativní *open source* knihovna UI komponent, která je navržena tak, aby vylepšila webový vývoj, zejména pro projekty využívající React. Hlavní předností této knihovny je její lehkost, díky čemuž se snadno integruje do projektů bez nutnosti zatěžujících závislostí. Knihovna staví zejména na komponentách Radix UI, které kladou důrat na přístupnost, což zajišťuje, že komponenty jsou inkluzivní a použitelné pro všechny uživatele. [6]



Obrázek 3.3 Emailový klient vytvořený pomocí shadcn/ui

Několik hlavních myšlenek autora knihovny shadcn/ui, proč je lepší kód vlastnit (kopírovat), než používat balíčky [6]:

- Smyslem je poskytnout vlastnictví, kontrolu nad kódem a možnost rozhodovat o tom, jak budou komponenty použity a stylizovány.
- Začít s nějakými rozumnými výchozími nastaveními a poté přizpůsobit součásti potřebám projektu.
- Jednou z nevýhod balíčkování komponent do npm balíčku je, že styl je spojen s implementací. Styl komponent by měl být oddělen od jejich implementace.

Na rozdíl od Radix UI nemusí vývojáři věnovat tolik času vývoji komponent, protože jsou již předpřipravené (složené z primitivních komponent Radix UI). Také mají možnost upravit kód komponent, což jim umožňuje přizpůsobit je specifickým potřebám jejich projektu. Tento přístup kombinuje výhody distribuovaných knihoven s flexibilitou vlastního kódu, což umožňuje vývojářům rychle upravovat komponenty podle potřeby.

Další významnou součástí knihovny shadcn/ui je integrace s *frameworkem* Tailwind CSS, který poskytuje efektivní a přívětivé prostředí pro vývojáře, preferující tento CSS *framework* orientovaný na utilitu. Výhody Tailwind CSS jsou popsány v kapitole věnované technologiím. Tato integrace umožňuje snadnou úpravu a rozšíření stylů. Shadcn/ui vyniká svou snadnou použitelností a detailní kontrolou nad komponentami. Vývojáři mohou přímo přistupovat ke zdrojovému kódu jednotlivých komponent, což umožňuje jejich efektivní úpravy pro specifické případy užití a požadavky aplikace.

Shadcn/ui také nabízí CLI, který usnadňuje práci s knihovnou komponent. Umožňuje vývojářům inicializovat projekt nebo rychle přidávat komponenty přímo z příkazové řádky. Vývojáři mají také možnost porovnat změny mezi jejich kódem a kódem knihovny, což usnadňuje sledování a správu změn v kódu.

Podobně jako Nuxt UI také shadcn/ui nabízí knihovnu komponent ve Figmě. Na rozdíl od Nuxt UI nebyla vytvořena autory, ale členem open-source komunity. [7]

Hodnocení

- **Výhody**
  - Jednoduché úpravy
  - Kód vlastní uživatel
- **Nevýhody**
  - Omezená funkcionalita
  - Větší objem spravovaného kódu

3.3 Proces aktualizace

Proces aktualizace komponent se výrazně liší mezi knihovnami s principem balíčkování a knihovnami s principem vlastnění kódu. U knihoven s principem balíčkování, je proces aktualizace relativně jednoduchý a efektivní. Vývojáři mohou snadno aktualizovat komponenty na nejnovější verzi pomocí správce balíčků, což zajišťuje, že všechny závislosti jsou správně spravovány a kompatibilní. Tento přístup minimalizuje riziko konfliktů v kódu a umožňuje týmům rychle reagovat na opravy chyb nebo získat nové funkce bez nutnosti zásadně přepisovat existující kód (za předpokladu zachování rozhraní komponent).

Na druhé straně, u knihoven s principem vlastnění kódu, kde vývojáři přebírají základní komponenty a adaptují je podle svých potřeb, není aktualizace tak přímá. Protože tyto komponenty jsou často značně upraveny a integrovány přímo do kódu aplikace, neexistuje jednoduchý mechanismus pro jejich aktualizaci prostřednictvím správce balíčků. Místo toho mohou projekty vyžadovat vlastní řešení, jako je vývoj vlastního CLI nástroje, který by pomáhal spravovat a aktualizovat tyto upravené komponenty. Alternativní možností je, že aktualizace se jednoduše nebudou implementovat, protože jejich komponenty jsou specificky navrženy pro konkrétní použití a nevyžadují pravidelné aktualizace, jaké jsou běžné u standardních knihoven komponent.

3.4 TypeScript vs JavaScript

V současné době, kdy technologie neustále pokračují ve svém vývoji, je volba vhodného programovacího jazyka velmi důležitá. Mezi nejpopulárnější a široce používané programovací jazyky v oblasti webu patří JavaScript a jeho nadstavba TypeScript. Každý z nich přináší specifické výhody i potenciální komplikace. Tato sekce se zaměří na porovnání těchto dvou technologií, aby bylo možné učinit informované rozhodnutí o tom, který jazyk je pro náš projekt nejvhodnější.

3.4.1 JavaScript

JavaScript je dynamický skriptovací jazyk, který byl původně navržen pro manipulaci s webovými stránkami, ale postupem času se jeho využití rozšířilo i do mnoha dalších oblastí včetně serverových aplikací, desktopových aplikací a dokonce i programování hardwaru. Jako jazyk běžící přímo v prohlížeči má JavaScript nezastupitelnou roli ve vývoji front-endu. [**JavaScript**]

Výhody

Univerzálnost a dostupnost JavaScript je podporován všemi hlavními prohlížeči bez potřeby jakéhokoliv doplňku.

Velká komunita a ekosystém Existuje obrovské množství knihoven, *frameworků* a nástrojů, které usnadňují vývoj v JavaScriptu.

Flexibilita JavaScript umožňuje různé styly programování (procedurální, objektově orientované, funkcionální), což dává vývojářům svobodu ve výběru přístupu k řešení problémů.

Nevýhody

Bez typové bezpečnosti Jazyk je dynamicky typovaný, což může vést k chybám za běhu, které jsou obtížně odhalitelné během vývoje.

3.4.2 TypeScript

TypeScript je nadstavba JavaScriptu vyvinutá společností Microsoft, která přidává statickou typovou kontrolu a další funkce zaměřené na zlepšení organizace kódu a jeho udržitelnosti. TypeScript je plně kompatibilní s JavaScriptem, což znamená, že jakýkoli platný JavaScriptový kód je také platným Typescriptovým kódem. [**TypeScript**]

Výhody

Statická typová kontrola Přidává bezpečnost typů na úrovni kompilace, což pomáhá odhalit chyby dříve, usnadňuje *refactoring* a zvyšuje čitelnost kódu.

Podpora pro velké projekty S funkcemi jako jsou rozhraní a generika, TypeScript lépe podporuje vývoj větších a technicky složitějších aplikací.

Vylepšená nástrojová podpora Integrace s vývojovými prostředími (IDE) poskytuje lepší nástroje pro automatické dokončování kódu a navigaci.

Nevýhody

Náročnější na naučení Přechod a následné používání Typescriptu může vyžadovat dodatečnou práci v podobě přidání typových anotací a konfigurace. Také může být pro nové uživatele náročnější se naučit nové koncepty a syntaxi.

3.4.3 Porovnání

V blízké minulosti se některé velké *open source* projekty, jako je například Svelte nebo Drizzle, rozhodly přejít z Typescriptu zpět na JavaScript. Tento trend vyvolal diskuzi o tom, který jazyk je pro vývoj knihoven lepší. Je několik klíčových faktorů, které je třeba zvážit při rozhodování mezi JavaScriptem a Typescriptem.

Projekty, které přešly z Typescriptu na JavaScript stále používají JSDoc definice pro generování typů. Všechny tyto projekty byly velké, složité a potřebovaly vyřešit problémy s definicí typů, které v rámci takto velkých projektů mohou vyžadovat velké množství práce. [8]

Výhody přechodu na JSDoc

- Bez TypeScriptu se zmenší velikost balíčků.
- Možnost přejít na zdrojový kód kliknutím na import funkce usnadňuje úpravy a ladění *frameworku*.
- Pro zobrazení změn není potřeba propojovat repozitář, kompilovat v režimu sledování nebo rozumět mapování mezi zdrojovým a distribučním kódem.

Nevýhody přechodu na JSDoc

- Nutnost používat JSDoc pro psaní typů, což není tak příjemné jako psaní v TypeScriptu.

```
1  /**
2  * Returns the sum of a and b
3  * @param {number} a
4  * @param {number} b
5  * @param {boolean} retArr If set to true, the function will return an array
6  * @returns {(number|Array)} Sum of a and b or an array that contains a, b and
   ↪  the sum of a and b.
7  */
8  function sum(a, b, retArr) {
9      if (retArr) {
10         return [a, b, a + b];
11     }
12     return a + b;
13 }
14
```

Výpis kódu 1 JSDoc komentáře [9]

Na druhou stranu pro koncové aplikace je TypeScript stále lepší volbou, protože poskytuje výhody statické typové kontroly, které mohou výrazně snížit počet chyb a zvýšit kvalitu kódu. [10]

Vzhledem k tomu, že většina kódu se dostane přímo do rukou vývojářů, bude lepší použít TypeScript, se kterým se lépe pracuje v koncových aplikacích. Pro knihovny, které jsou určeny k širokému použití jako balíčky, může být lepší použití JavaScriptu, protože to zjednoduší vývoj knihovny z hlediska typování atd. a zmenší velikost balíčků. Pro použití kolekce tedy bude povinná závislost na Typescriptu.

3.5 Závislosti na knihovnách třetích stran

Využití závislostí třetích stran v procesu vývoje komponent může přinést řadu výhod, zvláště když je důraz kladen na efektivitu a dostupnost. Knihovny poskytující přístupné UI prvky jsou zajímavou možností, která umožňuje urychlit proces implementace bez ztráty kontroly nad funkcionalitou a stylováním. Pro React je zajímavá knihovna Radix UI, avšak její Vue verze je spravována komunitou a nedosahuje takové kvality jako ta původní. Knihovna, která je dostupná pro oba *frameworky* se jmenuje Headless UI.

Přestože knihovny třetích stran mohou nabídnout robustní a funkční komponenty, je důležité, aby vývojáři udrželi plnou kontrolu nad funkcemi a stylováním svých aplikací. To znamená, že i když jsou základní komponenty převzaty, měli by být schopni je dále upravovat a přizpůsobovat tak, aby odpovídaly specifickým požadavkům a estetickým preferencím projektu. Tyto požadavky Headless UI splňuje a dává tedy smysl jej využít. Tím odpadá potřeba implementovat nízkoúrovňové rozhraní, ale vývojáři budou stále mít vysokou kontrolu nad funkcionalitou a vizuálem, díky přímému přístupu k zdrojovému kódu.

Aktualizace využitých knihoven mají pak vývojáři na starosti sami. Na jednu stranu to znamená, že jim přibude určitá zodpovědnost při aktualizaci, na druhou stranu mohou využít nové funkce a opravy chyb bez nutnosti čekat na oficiální aktualizaci knihovny (oproti knihovnám s principem balíčkování, kde by bylo potřeba vytvářet vlastní *patch* nebo *fork*).

3.6 Závěr

Tato kapitola poskytla komplexní přehled stávajících UI knihoven a technologií, které ovlivňují vývoj a implementaci komponent. Důkladné srovnání přístupů založených na balíčkování a vlastnictví kódu odhalilo výhody a nevýhody každého z nich. Zvážení aspektů jako udržovatelnost, aktualizace a flexibilita vedlo k rozhodnutí o preferenci přístupu s vlastnictvím kódu pro maximální přizpůsobení a kontrolu.

Dále bylo provedeno porovnání programovacích jazyků TypeScript a JavaScript, přičemž byl zdůrazněn přínos statické typové kontroly Typescriptu pro větší projekty a jeho dopad na kvalitu kódu. Nakonec, analýza závislostí na knihovnách třetích stran, jako je Headless UI, poukázala na potenciál pro urychlení vývoje a zároveň zdůraznila nutnost zachování kontroly nad funkcemi a stylováním.

Získané poznatky z analýzy poskytují pevný základ pro návrh a implementaci sbírky znovupoužitelných UI komponent, která bude efektivní, flexibilní a přizpůsobitelná specifickým potřebám vývojářů.

Kapitola 4
Technologie

4.1 Úvod

Tato kapitola poskytuje podrobný přehled a analýzu technologií použitých v rámci diplomové práce. Správný výběr technologických nástrojů je zásadní nejen pro funkčnost a efektivitu finálního produktu, ale také ovlivňuje proces vývoje, údržbu a možnosti dalšího rozvoje projektu. Základem projektu jsou moderní technologie a *frameworky*, které jsou velmi populární v oblasti webového vývoje. Mezi klíčové technologie patří frontendový *framework* Vue společně s *metaframeworkem* Nuxt, jazyk Typescript a Tailwind pro stylování. Každá z těchto technologií byla vybrána na základě zkušeností autora, potřeb projektu a jejich schopnosti efektivně adresovat požadavky na vývojářskou a uživatelskou přívětivost. Následující sekce poskytne rozbor důvodů vedoucích k výběru těchto nástrojů a popis, jak jednotlivé technologie spolupracují na dosažení stanovených cílů.

4.2 Dokumentace a komponenty

Tato sekce se věnuje technologiím a nástrojům, které budou použity pro dokumentaci a vývoj komponent.

4.2.1 Vue

Vue.js je *framework* pro Javascript zaměřený na tvorbu uživatelských rozhraní. Vychází z běžných webových technologií, jako jsou HTML, CSS a Javascript, a přináší model programování založený na komponentách a deklarativním přístupu, který usnadňuje vývoj rozhraní různých úrovní složitosti.

Dvě klíčové funkce Vue:

- Deklarativní vykreslování: Vue rozšiřuje standardní HTML pomocí šablony syntaxe, která umožňuje deklarativně specifikovat výstup HTML založený na stavech v Javascriptu.
- Reaktivita: Vue efektivně monitoruje změny ve stavech Javascriptu a při těchto změnách automaticky a efektivně aktualizuje DOM. [11]

Vue je známé svou jednoduchostí a flexibilitou, což z něj činí ideální volbu pro vývojáře různých úrovní zkušeností. V rámci této diplomové práce je Vue použito jako základní technologie pro tvorbu komponent. Avšak komponenty nemusí být 100% kompatibilní s čistým Vue, protože se zde používají některé funkce, které jsou dostupné pouze v rámci *frameworku* Nuxt.

4.2.2 Nuxt

Nuxt je moderní a výkonný open source *framework* založený na Vue, který je navržen pro intuitivní a efektivní vývoj webových aplikací a stránek. Tento *framework* se vyznačuje tím, že automaticky řeší mnoho opakujících se úkolů vývojářů, což jim umožňuje soustředit se na tvorbu samotné aplikace. Nuxt využívá konvence a strukturované adresářové uspořádání pro automatizaci procesu vývoje a nabízí možnost vlastní konfigurace a přizpůsobení výchozích chování.

Nuxt má narozdíl od čistého Vue schopnosti server-side renderingu (SSR). Tato funkce zajišťuje rychlejší načítání stránek a lepší SEO, protože celý obsah stránky je serverem vygenerován dříve, než začne běžet klientský Javascript. Nuxt tento přístup kombinuje v takzvaném hybrid renderingu a má tedy výhody tradičního server-side renderingu s interaktivitou a pokročilými uživatelskými rozhraními single-page aplikací (SPA). [12]

4.2.3 Tailwind

Tailwind je moderní a oblíbený CSS *framework* [13] [14], který se zaměřuje na tzv. *utility-first* přístup. Tento přístup umožňuje vývojářům rychle a efektivně vytvářet vlastní komponenty a designy s použitím nízkoúrovňových CSS tříd, které jsou přímo integrovány do HTML markupu.

Jednou z hlavních výhod Tailwind je možnost psát méně vlastního CSS. Vývojáři mohou využívat předdefinované třídy, jako jsou *flexbox* a padding utility, což znamená, že většina stylů je opakovaně použitelná a zřídka je potřeba psát nové CSS. Tailwind také eliminuje potřebu vymýšlet názvy tříd, protože vývojáři vybírají třídy z předdefinovaného designového systému. To znamená, že nemusí přemýšlet o „dokonalých“ názvech tříd pro určité styly a komponenty nebo si pamatovat složité názvy. [15]

Tailwind je známý svou flexibilitou a kontrolou nad tím, jak aplikace vypadá, což poskytuje větší prostor pro vytváření jedinečných webů. Na rozdíl od jiných CSS *frameworků*, jako je Bootstrap nebo Materialize, Tailwind nenabízí plně stylované komponenty, jako jsou tlačítka nebo *dropdown* menu. Místo toho nabízí utility třídy, které umožňují vytvořit vlastní opakovaně použitelné komponenty.

Přestože Tailwind nabízí mnoho výhod, může být jeho použití obtížné pro ty, kteří nejsou zkušení s CSS, a může způsobovat zmatek kvůli množství informací uložených v HTML souboru. Navíc při instalaci Tailwindu jsou výchozí CSS styly resetovány [16], takže je nutné je pro všechny elementy znovu vytvořit.

Tailwind bude v této diplomové práci použit pro stylování komponent a stránek. Také je využit pro tvorbu motivů a brandingu pomocí jeho konfiguračního souboru a CSS proměnných.

4.2.4 Typescript

Typescript je nadstavba jazyka Javascript, která přidává statické typování a další pokročilé vlastnosti, jež zvyšují kvalitu a udržitelnost kódu ve velkých softwarových projektech. Díky kompatibilitě s Javascriptem umožňuje Typescript vývojářům využívat veškeré existující knihovny a *frameworky*, zároveň ale poskytuje výhody striktnějšího typového systému, jako jsou například lepší nástroje pro refaktoring, snazší navigace v kódu a vyšší bezpečnost během kompilace. V diplomové práci je Typescript využíván pro zlepšení spolehlivosti aplikace a minimalizaci chyb za běhu, což je zvláště důležité při vývoji komplexních systémů s mnoha interakcemi mezi komponenty. Přidání typů do kódu umožňuje vytvářet lepší dokumentaci a zlepšit komunikaci mezi členy vývojového týmu, což zjednodušuje rozvoj a údržbu projektu. V kombinaci s *frameworkem* Vue a Nuxt, Typescript přináší výrazné vylepšení v rychlosti vývoje a kvalitě výsledného produktu. [17]

4.2.5 Nuxt Content

Nuxt Content je *headless* CMS založený na verzovacím systému Git pro Vue vývojáře. Umožňuje vytvářet obsah pomocí Markdown a JSON a dotazovat se na něj pomocí API podobného MongoDB. [18]

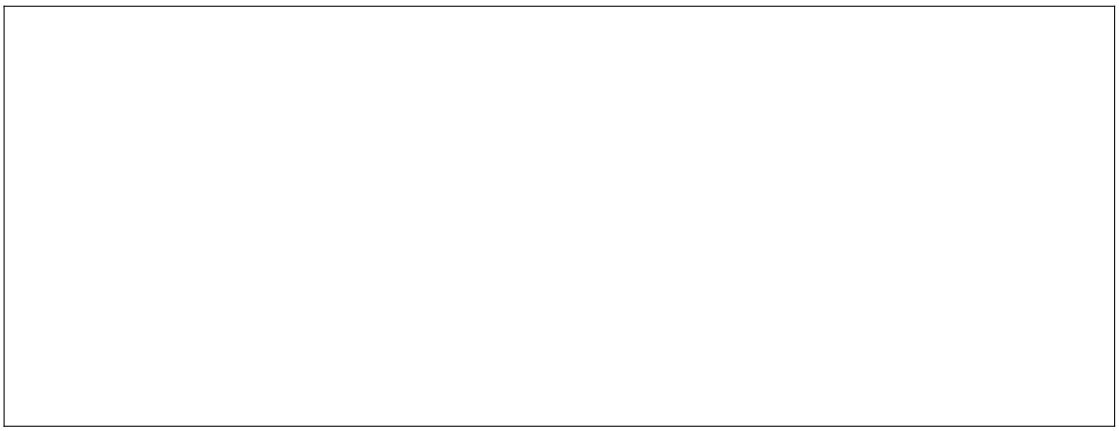
Nuxt Content je modul pro *framework* Nuxt, který usnadňuje práci s obsahem. Tento modul umožňuje vývojářům psát obsah přímo v Markdown, YAML, JSON, a XML souborech a efektivně je integrovat do Nuxt aplikací. Obsah je možné snadno načítat a zobrazovat díky jednoduchému API, které Nuxt Content nabízí. Kromě snadného načítání dat, modul poskytuje nástroje pro full-textové vyhledávání, což je velmi užitečné dokumentace, blogy a další aplikace, které vyžadují efektivní správu obsahu. Integrace s Vue komponentami je přímá a bezproblémová, což výrazně zlepšuje workflow vývojářů a umožňuje jim věnovat více času na kreativní aspekty projektu, místo zbytečného řešení technických detailů správy obsahu.

V této diplomové práci bude Nuxt Content využit pro dokumentaci komponent a strukturovaný obsah, který je snadno editovatelný a spravovatelný v rámci Git repozitáře. Nuxt Content je ideální volbou, protože nevyžaduje žádnou další infrastrukturu.

4.2.6 Shiki

Shiki je knihovna pro syntax highlighting, která je založena na stejném mechanismu, jaký používá Visual Studio Code. Tato technologie poskytuje vysoce přesné a vizuálně atraktivní zvýrazňování syntaxe pro širokou škálu programovacích jazyků. Shiki umožňuje snadnou integraci do různých webových projektů díky svému API a podporuje načítání různých barevných témat, což umožňuje vývojářům přizpůsobit vzhled zvýraznění kódu svým specifickým potřebám. Efektivita a flexibilita Shiki v integraci a konfiguraci z něj činí ideální nástroj pro projekty, které vyžadují zobrazení zdrojového kódu v dokumentaci nebo výukových materiálech, přičemž zachovává konzistenci a čitelnost prezentovaného kódu. [19]

V této diplomové práci bude Shiki použit pro zvýrazňování syntaxe v dokumentaci a ukázkových kódech, což zvyšuje čitelnost obsahu a usnadňuje vývojářům rychleji porozumět kódu.



Obrázek 4.1 Zvýraznění kódu pomocí Shiki

4.2.7 Headless UI

Nestylované, přístupné UI komponenty, navržené tak, aby se jednoduše používaly s Tailwind CSS. [20]

Headless UI je sada přístupných UI komponent, které jsou navrženy tak, aby byly snadno integrovatelné do jakéhokoliv designu nebo frontendového *frameworku*. Tato knihovna poskytuje

základní funkční strukturu pro běžné UI prvky, jako jsou rozbalovací menu, přepínače a dialogová okna, ale neobsahuje žádné vlastní styly, což umožňuje vývojářům plně ovládat vzhled těchto komponent. Headless UI je ideální pro projekty, které vyžadují vysokou míru přizpůsobení a chtějí používat specifický designový jazyk bez nutnosti bojovat s předdefinovanými styly. Komponenty jsou navrženy s ohledem na přístupnost, což zajišťuje, že aplikace jsou použitelné pro širokou škálu uživatelů, včetně těch s omezenými schopnostmi. Headless UI tak usnadňuje vytváření robustních a esteticky příjemných webových aplikací, které jsou zároveň přístupné a uživatelsky přívětivé.

V této diplomové práci bude Headless UI použito pro tvorbu základních UI komponent, jako jsou modální okna, navigační lišty či další prvky. Tato knihovna poskytuje flexibilitu a kontrolu nad vzhledem komponent, což je důležité pro jednoduché úpravy a rozšíření.

4.2.8 Floating UI

Floating UI je Javascriptová knihovna navržená pro pozicování plovoucích prvků, jako jsou *tooltipy*, *popovery* a *dropdown* menu, a umožňuje také vytváření interakcí pro tyto prvky. Knihovna se zaměřuje na robustní ukotvení absolutně pozicovaných plovoucích prvků vedle referenčních prvků s funkcemi, které zabraňují kolizím s okrajem zobrazení, čímž zajišťuje, že plovoucí prvky zůstanou vždy viditelné a optimálně umístěné.

Floating UI podporuje různé platformy včetně webu, React, React Native a Vue, a je vysoce modulární, což umožňuje vývojářům použít jen tolik funkcionality, kolik potřebují, což může vést k optimalizaci velikosti balíčku. Dále je knihovna navržena tak, aby byla flexibilní a umožňovala snadnou integraci bez omezení na specifické technologie nebo *frameworky*. [21]

V této diplomové práce se Floating UI používá pro tvorbu interaktivních prvků, jako jsou *tooltipy* a *popovery*, které zvyšují uživatelskou interaktivitu a zlepšují uživatelskou zkušenost. Důležité je především zabránění přetékání prvků mimo obrazovku a zajištění, že plovoucí prvky jsou vždy viditelné a snadno přístupné.

4.2.9 Embla Carousel

Embla Carousel je malá, open source knihovna pro Javascript, která se zaměřuje na tvorbu pohyblivých prvků, jako jsou *carousely*, s vysokou přesností pohybu a skvělou reakcí na gesta. Tato knihovna je navržena tak, aby byla nezávislá na jiných knihovnách a nevyžaduje žádné další závislosti. [22]

Embla Carousel nabízí možnost snadné integrace do různých *frameworků* jako React, Vue, Svelte a Solid, díky čemuž je flexibilní pro různé typy projektů. Využívá moderní CSS vlastnosti, jako je *flexbox*, pro definování velikosti a mezer mezi prvky a podporuje i pokročilé funkce, jako je automatické přehrávání a responzivní nastavení.

K dispozici jsou různé pluginy, které rozšiřují základní funkčnost, včetně možností pro automatické přehrávání, *auto-scroll*, a adaptaci výšky. Embla také poskytuje rozsáhlé API pro pokročilé manipulace a přizpůsobení *carouselů*, což umožňuje vývojářům plně ovládat chování a vzhled *carouselů* podle potřeb projektu.

V této práci bude Embla Carousel použit pro tvorbu *carousel* komponenty. Díky vysoké flexibilitě poskytuje nepřeberné možnosti konfigurace a přizpůsobení.

4.3 CLI

Tato sekce se věnuje technologiím a nástrojům, které byly použity pro vývoj CLI pro instalaci komponent.

4.3.1 Commander.js

Commander.js je oblíbená knihovna pro Node.js, která usnadňuje vytváření příkazových řádkových rozhraní (CLI). Umožňuje vývojářům definovat příkazy, volby a parametry pro jejich aplikace v jednoduchém a strukturovaném formátu. Commander.js poskytuje funkce, jako je automatické generování nápovědy, parsování příkazů a argumentů, a podporu pro subpříkazy, což umožňuje vývojářům snadno rozšiřovat funkčnost svých aplikací bez nutnosti psát rozsáhlý kód pro zpracování vstupů. Díky své flexibilitě a jednoduchosti použití se Commander.js stal standardním nástrojem pro mnoho projektů založených na Node.js. [23]

V této diplomové práci bude Commander.js použit pro vytvoření CLI pro instalaci komponent.

Jako alternativa se nabízí Citty z ekosystému UnJS. Má podobné funkce jako Commander, avšak v době tvorby této práce byla tato knihovna v rané fázi vývoje a nebyla dostatečně stabilní pro použití v produkčním prostředí. [24]

4.3.2 Ora

Ora je minimalistická a efektivní knihovna pro Node.js, která umožňuje vývojářům přidat elegantní indikátory načítání (tzv. spinners) do jejich CLI. Tyto indikátory jsou užitečné pro zobrazování stavu dlouhotrvajících operací, aniž by uživatele zatěžovaly nadměrnými informacemi. Ora nabízí širokou škálu přednastavených stylů a umožňuje také snadnou personalizaci, včetně změny textu, barvy a symbolů spinneru. Tato knihovna je oblíbená pro svou snadnou integraci a schopnost výrazně zlepšit uživatelskou zkušenost CLI aplikací tím, že poskytuje okamžitou vizuální zpětnou vazbu během procesů, které mohou trvat několik sekund nebo i déle. [25]

V této diplomové práci bude Ora použit pro zobrazení indikátorů načítání během procesů instalace komponent.

4.3.3 Prompts

Prompts je knihovna pro Node.js, která umožňuje vývojářům snadno vytvářet interaktivní CLI. S pomocí této knihovny je možné efektivně spravovat uživatelské vstupy pomocí různých typů dotazů, jako jsou text, heslo, potvrzení a výběr z možností. Prompts podporuje asynchronní logiku, umožňuje validaci vstupů a podmíněné zobrazení dotazů, což vývojářům poskytuje flexibilitu pro vytváření složitějších scénářů interakcí. Tato knihovna je oblíbená pro svou jednoduchost, moderní API a minimální závislosti, což ji činí ideální volbou pro projekty, kde je potřeba rychle a efektivně zpracovávat uživatelské vstupy. [26]

V této diplomové práci bude knihovna Prompts použita pro interaktivní získávání informací od uživatele během procesu instalace komponent.

4.3.4 Execa

Execa je moderní knihovna pro Node.js, která zjednodušuje práci s externími procesy a příkazy shellu. Tato knihovna je navržena tak, aby vylepšila standardní Node.js metodu child_process a poskytla lepší zkušenost s pokročilými funkcemi, jako je podpora asynchronních operací, vylepšená oblast viditelnosti výstupů procesů a lepší manipulace s chybami. Execa umožňuje snadné spouštění shell příkazů a jejich sledování v reálném čase, což je zvláště užitečné v automatizovaných skriptech a nástrojích pro vývoj. Díky execa lze také snadno získat textový výstup příkazů, což umožňuje vývojářům efektivně zpracovávat data bez nutnosti manuálního parsingu. [27]

V této diplomové práci bude knihovna Execa použita pro instalaci závislostí během instalace komponent.

4.3.5 Zod

Zod je knihovna pro Typescript, která slouží k vytváření schémat a validaci dat s cílem zajistit správnost typů během kompilace i za běhu. Zod umožňuje vývojářům definovat strukturu dat pomocí silného typového systému Typescriptu, což vede ke snížení chyb způsobených neplatnými daty. Jednou z klíčových výhod Zodu je, že integrované validace se automaticky odrážejí ve vygenerovaných typech, což znamená, že jakékoli data splňující validaci jsou automaticky správně typována. Tato knihovna také podporuje složitější validační struktury, včetně vnořených objektů, pole, volitelných polí a unie typů, což ji činí ideálním řešením pro aplikace, které vyžadují přísnou kontrolu vstupních dat. [28]

V této diplomové práci je knihovna Zod použita pro definici schémat a validaci dat během procesu instalace komponent.

4.3.6 Tsup

Tsup je minimalistický a rychlý bundler pro Typescript a Javascript, postavený na technologii esbuild. Umožňuje vývojářům efektivně sestavovat své projekty s výrazným zlepšením v rychlosti sestavení oproti tradičním nástrojům jako je Webpack nebo Rollup. Tsup nabízí jednoduché CLI s několika konfiguračními možnostmi, což usnadňuje jeho používání a integraci do různých projektů. Mezi klíčové funkce patří podpora pro tree-shaking, generování deklarací Typescript, a podpora pro ESM a CommonJS moduly. Díky své závislosti na esbuild, tsup poskytuje nejen rychlost, ale také efektivitu při minimalizaci a optimalizaci výsledného kódu. [29]

V této diplomové práci bude tsup použit pro sestavení výsledného balíčku CLI pro instalaci komponent.

4.4 Infrastruktura

Tato sekce se věnuje technologiím a nástrojům, které byly použity pro správu kódu a jeho následné nasazení.

4.4.1 pnpm workspaces

pnpm workspaces jsou součástí správce balíčků pnpm, které umožňují efektivní správu více balíčků v rámci jednoho repozitáře (monorepo). Tato funkcionalita usnadňuje sdílení závislostí mezi projekty, což minimalizuje redundanci a zvyšuje efektivitu správy kódu. Workspaces podporují spojení více projektů pod jednou střechou s jednotnou správou závislostí, což je ideální pro velké projekty s mnoha podprojekty nebo knihovnami. Díky pnpm workspaces lze jednoduše instalovat závislosti pro všechny workspaces najednou, zatímco se zachovává úspora místa na disku a rychlost díky strategii pnpm pro sdílení závislostí. [30]

V této diplomové práci jsou pnpm workspaces použity pro efektivní správu kódu v rámci monorepozitáře. Díky tomu, že jsou oba projekty (CLI a dokumentace) součástí jednoho repozitáře, je možné jednoduše testovat jejich propojení bez nutnosti stahování více repozitářů a jejich propojování, což zjednodušuje vývoj a údržbu.

4.4.2 Changesets

Changesets je nástroj, který pomáhá spravovat, verzovat a zveřejňovat změny v projektech s mnoha balíčky, typicky v monorepozitářích. Tento systém umožňuje vývojářům přidávat „changeset“ soubory do git repozitáře, které popisují změny v balíčcích a jejich dopad na semantické verzování. Když přichází čas na vydání nové verze, Changesets automaticky generuje changelogy a aktualizuje verze balíčků podle pravidel semantického verzování. Tato metoda zajišťuje, že

uživatelé knihovny mají jasné informace o tom, co každá verze zahrnuje. Changesets tak přináší transparentnost a předvídatelnost do procesu vývoje softwaru, což je zvláště cenné v prostředích, kde spolupracuje mnoho vývojářů. [31]

V této diplomové práci bude Changesets použit pro správu změn a verzování balíčků v rámci monorepozitáře. Tento nástroj zajišťuje, že všechny změny jsou správně dokumentovány a verzovány, což usnadňuje vydávání nových verzí.

4.4.3 npm

npm (Node Package Manager) je systém pro správu balíčků, který je standardně používán pro Javascriptový runtime Node.js. npm umožňuje vývojářům snadno sdílet a spravovat závislosti kódu v jejich Javascriptových projektech. Na oficiálních stránkách npm je dostupný obrovský repozitář obsahující přes 2,5 milionu balíčků, které může komunita využívat a přispívat do nich (v rámci open source repozitářů). Balíčky mohou obsahovat vše od malých knihoven až po komplexní *frameworky*. [32]

Přidání balíčku do npm je relativně jednoduchý proces, který začíná vytvořením účtu. Po přihlášení může vývojář vytvořit nový balíček pomocí příkazu npm init, který vede k nastavení balíčku včetně jeho názvu, verze, a závislostí. Po konfiguraci lze balíček publikovat na npm pomocí příkazu npm publish. Je důležité správně nastavit soubor package.json, který obsahuje metadata a závislosti balíčku. Vývojáři musí také zvážit licencování a další právní aspekty, aby byl jejich kód správně chráněn a licencován pro užití ostatními.

npx je nástroj přibalený s npm, který zjednodušuje spouštění balíčků nainstalovaných ve vašem Node.js prostředí nebo přímo z npm repozitáře bez nutnosti je explicitně instalovat. npx je užitečný zejména pro jednorázové spouštění balíčků nebo pro vyzkoušení nových nástrojů bez zatěžování vašeho lokálního vývojového prostředí. Když spustíte příkaz pomocí npx, například npx nuxi init my-app, npx automaticky stáhne potřebný balíček a jeho závislosti, spustí jej a po dokončení operace tyto dočasné instalace odstraní, čímž udržuje vaše prostředí čisté.

V této diplomové práci bude npm použito pro publikaci balíčku CLI pro instalaci komponent. Poté se pro jeho spuštění využívá npx.

4.4.4 Github

GitHub je jedna z největších a nejznámějších platforem pro správu softwarových projektů a verzování kódu pomocí systému Git. Umožňuje vývojářům hostovat a verzovat kód, spravovat projekty a sestavovat software společně s miliony dalších uživatelů po celém světě. GitHub poskytuje nástroje pro kolaborativní vývoj, jako jsou pull requesty (připojení nového kódu do dalších větví), správa větví a *issues* pro každý projekt. Platforma se stala populárním nástrojem pro open source projekty, ale je rovněž populární mezi společnostmi pro soukromý vývoj softwaru díky svým robustním funkcím pro správu přístupu a integraci s různými externími službami. Dále, GitHub rozšiřuje své funkce o GitHub Actions, což jsou automatizační nástroje pro CI/CD procesy, které umožňují automatizovat testování a nasazování aplikací přímo z repozitářů na GitHubu. [33]

V této diplomové práci bude GitHub použit pro správu kódu a verzování projektu. Díky GitHub Actions jsou automatizovány procesy testování a nasazování aplikace.

4.4.5 Cloudflare

Cloudflare je společnost poskytující širokou škálu služeb zaměřených na zlepšení bezpečnosti a výkonu internetových aplikací a webů. Cloudflare funguje primárně jako CDN (Content Delivery Network), které pomáhá zrychlit načítání webových stránek tím, že ukládá obsah na geograficky rozptýlených serverech a nabízí jej uživatelům z nejbližší možné lokality. Kromě toho Cloudflare

poskytuje řešení pro ochranu před DDoS útoky, zabezpečení webových aplikací (WAF), bezpečné DNS služby a optimalizaci provozu. [34]

Cloudflare Developer Platform nabízí vývojářům nástroje a API pro vytváření, testování a nasazování aplikací přímo na jejich edge network. Tato platforma zahrnuje řadu produktů jako Workers (serverless computing), který umožňuje vývojářům spouštět Javascript a WebAssembly kód na Cloudflare servery blízko uživatelů, což výrazně snižuje latenci a zvyšuje výkon aplikací. Avšak cenou za tuto rychlost je nutnost dodržovat určitá pravidla a omezit se na určité technologie, které jsou podporovány edge runtimem. [35]

Cloudflare Pages je platforma pro hosting statických webů a web aplikací, která je integrovaná přímo do Cloudflare infrastruktury. Je ideální pro projekty, které vyžadují rychlé načítání a globální dostupnost bez nutnosti spravovat serverovou infrastrukturu. Pages podporuje moderní *frameworky* a build nástroje jako React, Vue, Angular, včetně Nuxtu, a mnoho dalších, a automaticky konfiguruje HTTPS a SSL pro všechny nasazené stránky. [36]

Hlavní výhody Cloudflare Pages zahrnují:

- Snadné nasazení: Cloudflare Pages jsou navrženy pro snadnou integraci s GitHubem, což umožňuje vývojářům nasadit své aplikace přímo z repozitářů.
- Automatizované buildy: Každý push do hlavní větve repozitáře automaticky spustí proces sestavení a nasazení nové verze aplikace.
- Bezpečnost a rychlost: Využívání Cloudflare CDN a bezpečnostních funkcí zajišťuje, že aplikace jsou nejen rychle dostupné odkudkoli, ale také chráněné před běžnými útoky.

V této diplomové práci budou Cloudflare Pages použity pro nasazení dokumentace a ukázek. Díky integraci s GitHubem je možné jednoduše nasadit novou verzi aplikace po každé změně v repozitáři.

Kapitola 5
Návrh

5.1 Úvod

Návrh je důležitým krokem v procesu vývoje, který předchází samotné implementaci a testování. V této kapitole bude podrobně rozebrán postup a myšlenkové procesy, které vedou k vytvoření sbírky komponent přívětivé jak pro uživatele, tak pro vývojáře. Nejprve bude popsán vývoj a struktura dokumentace, která je nezbytná pro správné použití a rozšíření UI kolekce, dále návrh samotných komponent a příprava Figma kitu, který slouží jako most mezi designéry a vývojáři. Také bude vysvětlen význam a využití bloků, které umožňují rychlejší a efektivnější vývoj webových aplikací. Nakonec bude popsán návrh CLI, které umožňuje snadné přidávání komponent do existujících projektů.

5.2 Dokumentace

Součástí každého softwarového projektu by měla být kvalitní dokumentace (ačkoliv se tak často neděje). Pro účely této sbírky znovupoužitelných UI komponent je potřeba připravit dokumentaci, která nejen vysvětluje jak používat jednotlivé komponenty, ale také poskytuje ucelený pohled na architekturu a filozofii za celým projektem. Tato sekce se zabývá obsahem dokumentace a její strukturou.

5.2.1 Obsah dokumentace

Hlavním cílem dokumentace je poskytnout vývojářům všechny potřebné informace.

Úvod do principů a filozofie knihovny Vysvětlení v čem se knihovna liší od typických UI knihoven.

Návod k použití Průvodce krok za krokem jak začít využívat komponenty knihovny.

Podpůrné materiály Doprovodné zdroje, například Figma kit a tipy pro řešení problémů.

Přehled komponent Detailní popis každé komponenty, včetně jejího účelu, možností konfigurace a příkladů použití.

Dokumentace bude rozdělena do dvou hlavních sekcí: *Getting Started* a *Components*. Sekce *Getting Started* bude poskytovat uživatelům všechny základní informace potřebné pro rychlý start s knihovnou, včetně instalace, konfigurace a prvních kroků s komponentami. To pomůže novým uživatelům snadno se orientovat v možnostech knihovny a začít s jejím používáním. Sekce *Components* se bude zabývat detailním popisem jednotlivých komponent, včetně jejich vlastností, možností přizpůsobení a příkladů použití.

5.2.2 Dokumentace konkrétních komponent

Stránka dokumentace pro každou konkrétní komponentu bude jasně strukturovaná a bude poskytovat ucelené informace, které usnadní její použití. Na začátku stránky bude uveden název komponenty a pod ním krátký popis, který objasňuje, k čemu komponenta slouží. Bude následovat sekce s ukázkou komponenty, doplněná o příslušný zdrojový kód, který bude ilustrovat, jak lze komponentu použít. Návod k instalaci bude popisovat, jak komponentu přidat do projektu, ať už pomocí CLI nástroje nebo prostřednictvím manuálního kopírování kódu. Na stránce vývojáři naleznou také seznam *props* a *slotů*, které umožňují přizpůsobit komponentu specifickým potřebám. Stránka může dále obsahovat další poznámky a varování ohledně specifik komponenty, jako jsou potenciální chyby, omezení, nebo doporučené postupy.

5.3 Figma Kit

Figma Kit je vyvinut s cílem poskytnout designérům a vývojářům snadno použitelnou sadu komponent, které jim umožní efektivně navrhovat a prototypovat aplikace. Kit obsahuje komponenty UI kolekce, přesně tak, jak jsou definovány ve finální implementaci, což zajišťuje, že designy jsou plně kompatibilní s konečnou implementací. Připravené jsou komponenty jako tlačítka, inputy, plovoucí prvky a mnoho dalšího, což designérům umožňuje sestavit uživatelské rozhraní rychle a s vysokou mírou přesnosti.

Pro vývojáře je pak snadné převést designy vytvořené v Figmě do kódu, protože mohou snadno identifikovat, které komponenty mají být použity. Díky definovaným proměnným lze snadno přizpůsobit barvy, velikosti a další vlastnosti komponent, které vývojáři mohou specifikovat v rámci projektu.

U každé komponenty se bude nacházet odkaz do dokumentace, kde mohou jak designéři, tak vývojáři najít podrobné informace o tom, k čemu komponenta je a jaké jsou její možnosti.

Obrázek 5.1 Komponenty ve Figmě

5.4 Komponenty

Výběr těchto komponent byl založen na jejich základní funkčnosti a častém využití ve webových aplikacích. Každá z těchto komponent přináší konkrétní užitečné vlastnosti, které zlepšují uživatelský zážitek a zjednodušují vývojový proces. Tento soubor základních komponent vytváří pevný základ pro další rozšiřování sbírky. V budoucnu bude sbírka doplněna o další komponenty, které budou reagovat na specifické potřeby a požadavky vývojářů.

5.4.1 Accordion

Accordion je komponenta určená k zobrazení a skrytí obsahu v rozbalovacích sekcích. Umožňuje uživatelům zobrazit pouze relevantní část informací a zbytek skrýt, čímž šetří místo na obrazovce a zvyšuje přehlednost. *Accordion* se často používá pro často kladené otázky (FAQ) nebo detaily produktů.

Obrázek 5.2 Accordion

5.4.2 Avatar

Avatar je vizuální reprezentace uživatele, obvykle v podobě kruhového nebo čtvercového obrázku. Používá se k identifikaci uživatelů v aplikacích, například v uživatelských profilech, seznamu kontaktů nebo diskuzních fórech. Avatar může obsahovat iniciály uživatele, ikonu nebo fotografii.

Obrázek 5.3 Avatar

5.4.3 Badge

Badge je komponenta používaná k zobrazení malého označení nebo indikátoru, který poskytuje dodatečnou informaci. Typickým použitím je například zobrazení štítků, počtu nepřečtených zpráv, oznámení nebo označení nových funkcí v aplikaci.

Obrázek 5.4 Badge

5.4.4 Breadcrumbs

Breadcrumbs slouží k navigaci a orientaci uživatelů na webových stránkách nebo v aplikacích. Zobrazuje hierarchii stránek, což uživatelům umožňuje rychle se vrátit na předchozí úrovně nebo domovskou stránku. Breadcrumbs zlepšují uživatelskou zkušenost tím, že usnadňují navigaci v komplexních strukturách.

Obrázek 5.5 Breadcrumbs

5.4.5 Button

Button je základní interaktivní komponenta, která umožňuje uživatelům provádět akce, jako je odeslání formuláře, zahájení procesu nebo navigace na jinou stránku. *Buttony* mohou mít různé styly a velikosti, aby odpovídaly specifickým požadavkům aplikace nebo designu.

Obrázek 5.6 Button

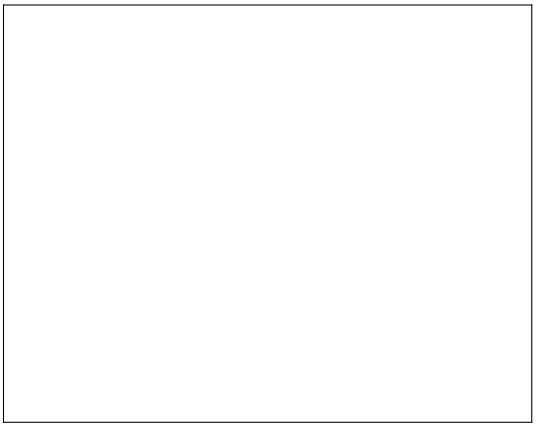
5.4.6 Card

Card je vizuální komponenta, která obklopuje obsah a poskytuje kontext. Karty jsou často používány k organizaci informací do samostatných bloků, které mohou obsahovat obrázky, texty a akční tlačítka. *Cards* se využívají v *dashboardech*, produktech nebo profilech uživatelů.

Obrázek 5.7 Card

5.4.7 Carousel

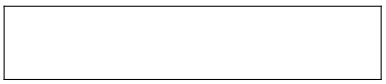
Carousel je komponenta umožňující zobrazení více položek (obrázků, karet) v rotující nebo posuvné formě. Umožňuje efektivně využít prostor a zobrazit více obsahu na omezené ploše, což je ideální pro prezentaci galerií, doporučených produktů nebo novinek.



Obrázek 5.8 carousel

5.4.8 Checkbox

Checkbox je vstupní komponenta, která umožňuje uživatelům vybrat jednu nebo více možností z nabídky. Používá se v formulářích, nastaveních a dalších interaktivních prvcích, kde je potřeba více výběrů.



Obrázek 5.9 checkbox

5.4.9 Dialog

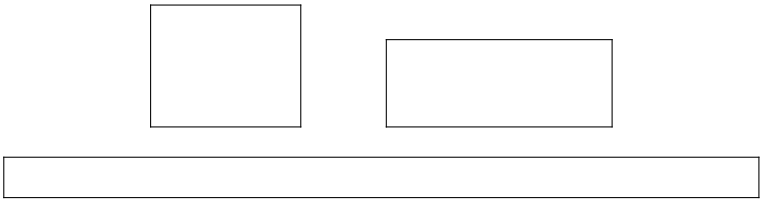
Dialog je modální okno, které se objevuje nad hlavním obsahem stránky, aby poskytlo důležité informace nebo vyžádalo potvrzení akce od uživatele. Dialogy jsou používány pro potvrzovací zprávy, formuláře nebo upozornění, která vyžadují okamžitou pozornost uživatele.

5.4.10 Icon

Icon je grafický symbol reprezentující akci, objekt nebo koncept. Ikony se používají ke zlepšení vizuální komunikace v uživatelských rozhraních, často jako doplněk textových popisků. Mohou být použity v tlačítkách, navigačních lištách nebo jako vizuální indikátory.

5.4.11 Loading

Loading je komponenta, která indikuje probíhající proces nebo načítání obsahu. Pomáhá uživatelům pochopit, že akce je v procesu a vyžaduje čas. *Loading* indikátory mohou mít různé formy, například točící se kolečko, prodlužující se pruh nebo blikající tečky.



Obrázek 5.10 Loading

5.4.12 Notification

Notification je komponenta pro zobrazení krátkých zpráv nebo upozornění uživatelům. Může informovat o úspěšných akcích, varováních, chybách nebo nových událostech. Notifikace mohou být dočasné nebo trvalé, a obvykle se zobrazují v rohu obrazovky.

Obrázek 5.11 notification

5.4.13 Pagination

Pagination je komponenta, která rozděluje obsah do více stránek a poskytuje ovládací prvky pro navigaci mezi nimi. Používá se v aplikacích a na webových stránkách s velkým množstvím dat, aby se usnadnila jejich přehlednost a navigace.

Obrázek 5.12 pagination

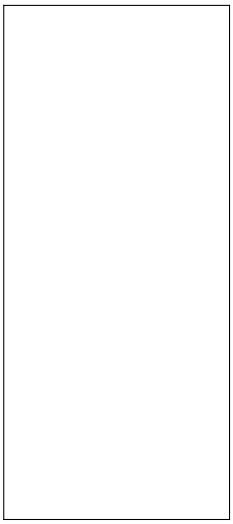
5.4.14 Popover

Popover je komponenta, která zobrazuje dodatečný obsah v malém okně, které se objevuje pod obsahem po interakci uživatele. *Popover* se často používá pro kontextové nabídky, tipy nebo formuláře, které vyžadují uživatelovu pozornost, aniž by opustil aktuální stránku.

Obrázek 5.13 popover

5.4.15 Select

Select je vstupní komponenta umožňující uživatelům vybrat jednu možnost z rozbalovací nabídky. Používá se v formulářích a nastaveních, kde je potřeba vybrat z předdefinovaného seznamu hodnot. *Select* komponenty mohou být jednoduché nebo víceúrovňové.



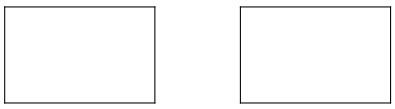
Obrázek 5.14 select

5.4.16 Sidebar

Sidebar je vertikální panel, který poskytuje navigaci nebo další obsah vedle hlavního obsahu stránky. Sidebary se používají pro menu, seznamy položek nebo další interaktivní prvky, které zůstávají přístupné během procházení hlavního obsahu.

5.4.17 Switch

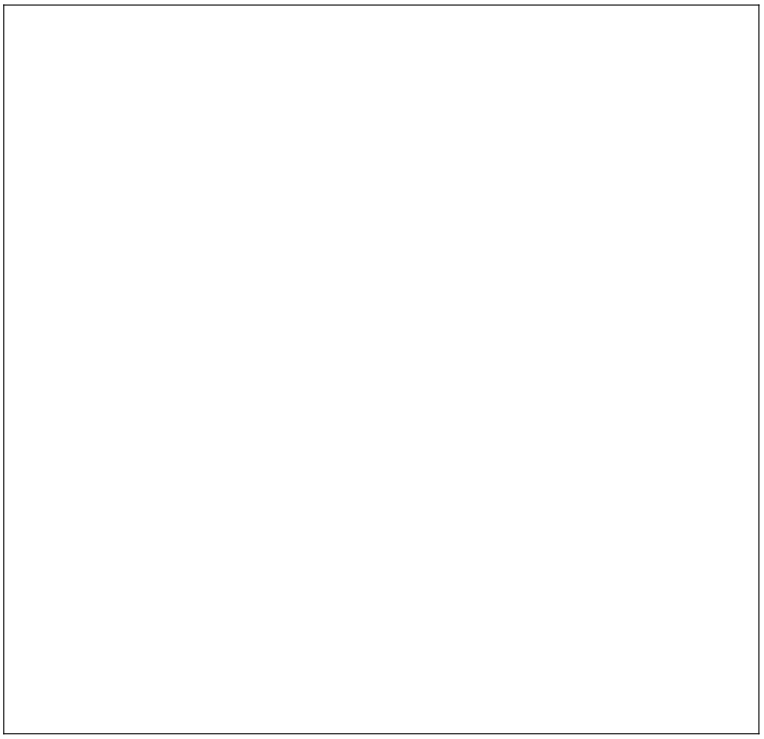
Switch je komponenta, která umožňuje uživatelům přepínat mezi dvěma stavy, obvykle zapnuto a vypnuto. Používá se pro nastavení, která mají binární volbu, jako je povolení nebo zakázání funkcí.



Obrázek 5.15 Switch

5.4.18 Table

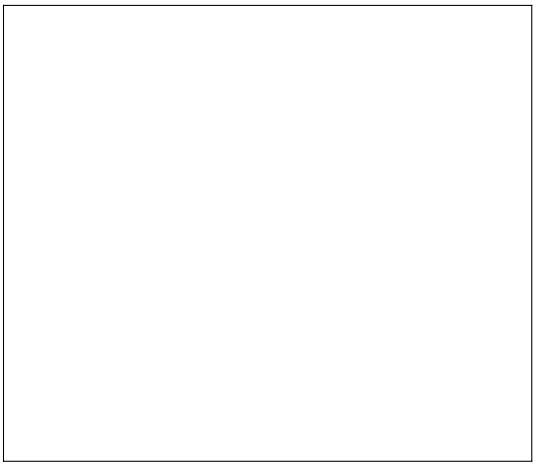
Table je komponenta pro zobrazení strukturovaných dat ve formě tabulky. Tabulky se používají k organizaci informací do řádků a sloupců, což umožňuje snadné srovnání a analýzu dat. Jsou běžně používány v dashboardech, reportech a dalších datově orientovaných aplikacích.



Obrázek 5.16 table

5.4.19 Tabs

Tabs jsou komponenty, které umožňují uživatelům přepínat mezi různými částmi obsahu na jedné stránce. *Tabs* zlepšují organizaci a přehlednost, když je potřeba prezentovat velké množství informací v omezeném prostoru. Každá karta obsahuje jiný obsah, který se zobrazí po kliknutí na záložku.



Obrázek 5.17 tabs

5.4.20 Textarea

Textarea je vstupní komponenta, která umožňuje uživatelům zadat víceřádkový text. Používá se v formulářích, kde je potřeba více místa pro text, například v komentářích, poznámkách nebo popisech.

Obrázek 5.18 textarea

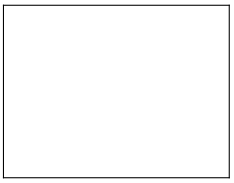
5.4.21 Text input

Text input je základní vstupní pole, které umožňuje uživatelům zadávat text. Používá se v formulářích pro zadání jednoduchých textových informací, jako jsou jména, e-maily nebo hesla. *Text input* může mít různé typy, například text, email, password nebo number.

Obrázek 5.19 textinput

5.4.22 Tooltip

Tooltip je komponenta, která poskytuje dodatečné informace o prvku, když na něj uživatel najede myší nebo se na něj jiným způsobem zaměří. *Tooltipy* se používají ke zlepšení uživatelského zážitku tím, že poskytují kontext nebo vysvětlení bez nutnosti zabírat místo na obrazovce.



Obrázek 5.20 tooltip

5.5 Bloky

Bloky jsou větší, komplexní komponenty, které fungují jako samostatné sekce webových aplikací. Dělí se na dva typy a to sice pro jedno použití a znovupoužitelné. Mezi komponenty pro jedno použití patří např. úvodní stránka. Tyto bloky většinou obsahují statický obsah, který se nemění a je na vývojářích, aby textace a obsah doplnili dle potřeby. Naopak znovupoužitelné bloky jsou komponenty, které se mohou použít na více místech aplikace. Mezi ně patří např. sekce, které se mohou nacházet na jedné stránce několikrát. [1]

Bloky značně zlepšují efektivitu vývoje tím, že standardizují často používané UI prvky do předdefinovaných šablon. Díky tomu mohou vývojáři snadno implementovat a opakovaně používat komplexní UI struktury bez nutnosti opětovného kódování nebo redesignu těchto částí pro každý nový projekt.

Hlavní přínosy bloků spočívají v jejich schopnosti snižovat redundantní kód a zvyšovat znovupoužitelnost komponent napříč projekty. Díky konzistentní implementaci jednotlivých bloků mohou vývojáři efektivně reagovat na nové požadavky nebo změny v designu bez rozsáhlých úprav celé aplikace.

Obrázek 5.21 Blok na úvodní stránku

5.5.1 Hero

Hero sekce je úvodní část stránky, která zaujme návštěvníky a poskytuje hlavní informace nebo klíčové sdělení. Obvykle se nachází v horní části stránky a obsahuje poutavý obrázek nebo video, výrazný nadpis a krátký popis. Je navržena tak, aby upoutala pozornost uživatelů a povzbudila je k dalšímu prozkoumání webu nebo provedení určité akce, například kliknutí na tlačítko s výzvou k akci (CTA).

[1] Tento nápad byl přidán během tvorby této práce i do knihovny shadcn/ui. Vzhledem k přidání do této úspěšné knihovny se potrzuje, že tato myšlenka je správná a má smysl.

5.5.2 Section

Section je univerzální blok, který rozděluje obsah stránky do logických částí. Každá sekce může mít vlastní obsah, jako jsou texty, obrázky nebo interaktivní prvky, a slouží k organizaci informací na stránce tak, aby byly přehledné a snadno čitelné. Sekce pomáhají strukturovat stránku a usnadňují uživatelům orientaci a nalezení relevantních informací.

5.5.3 Features

Features blok slouží k prezentaci hlavních funkcí nebo výhod produktu či služby. Obvykle se skládá z řady ikon nebo obrázků doplněných krátkými popisy, které stručně a jasně vysvětlují, co produkt nebo služba nabízí. Tento blok pomáhá uživatelům rychle pochopit, jaké benefity mohou očekávat, a usnadňuje rozhodování o koupi nebo dalším využití.

5.5.4 References

References blok je určen k zobrazení referencí, recenzí nebo citací od spokojených zákazníků či partnerů. Tento blok zvyšuje důvěryhodnost a kredibilitu produktu nebo služby tím, že poskytuje sociální důkazy o jeho kvalitě a spolehlivosti. Obsahuje často fotografie, jména a krátké výpovědi od skutečných uživatelů, což pomáhá budovat důvěru u potenciálních zákazníků.

5.5.5 Contact

Contact blok slouží k zobrazení kontaktních informací a formulářů, které umožňují uživatelům snadno se spojit s provozovatelem webu nebo podpůrným týmem. Tento blok obvykle obsahuje informace jako jméno, telefonní číslo a e-mailová adresa. Může také obsahovat kontaktní formulář, který uživatelům umožňuje rychle zaslat zprávu nebo dotaz přímo z webové stránky.

5.5.6 Auth

Auth blok se používá pro autentizaci uživatelů, tedy pro přihlášení, registraci nebo obnovení hesla. Obsahuje formuláře, které uživatelům umožňují zadávat své přihlašovací údaje, vytvářet nové účty nebo obnovit zapomenuté heslo. Tento blok je navržen pro webové aplikace, které vyžadují bezpečný přístup k uživatelským účtům a osobním informacím.

5.5.7 Blog

Blog blok slouží k zobrazení seznamu článků nebo příspěvků na blogu. Tento blok obvykle obsahuje náhledy článků s titulky, krátkými úryvky a odkazy na celé články. Blog sekce pomáhá udržovat web aktuální a informativní, zlepšuje SEO a poskytuje uživatelům hodnotný obsah, který je může zaujmout a přivést zpět na web.

5.5.8 Header

Header je horní část webové stránky, která obvykle obsahuje logo, navigační menu a další prvky, jako jsou odkazy na přihlášení, nákupní košík nebo vyhledávací pole. *Header* je důležitým prvkem každé webové stránky, protože poskytuje uživatelům rychlý přístup k hlavním sekcím webu a usnadňuje navigaci. Důraz na jednoduchost a přehlednost v *headeru* zlepšuje uživatelskou zkušenost a orientaci.

5.6 CLI

Vzhledem k tomu, že se nejedná o klasickou knihovnu s principem balíčkování, není možné využít package manažerů pro instalaci. Nabízí se tedy dvě možnosti.

1. Manuální kopírování souborů do projektu.
2. Použití CLI, které bude zajišťovat inicializaci a přidávání komponent do projektů.

Manuální kopírování souborů do projektu je sice možné, ale zdlouhavé a náchylné k chybám. Na druhou stranu je tato možnost velmi jednoduchá a nevyžaduje žádné další kroky. Ideální je tedy kombinace obou možností. Pokud vývojář z nějakých důvodů nebude chtít použít CLI, může si soubory zkopírovat z Githubu a ručně je přidat do projektu. Odkazy na Github budou dostupné v dokumentaci.

Pro usnadnění práce s UI knihovnou je tedy potřeba vyvinout CLI, které pomůže s inicializací a přidávání komponent do projektů.

5.6.1 Inicializace

Inicializace knihovny prostřednictvím CLI zajišťuje, že jsou přidány všechny závislosti a další potřebné soubory jsou připraveny k použití ihned po vytvoření nového projektu. Příkaz pro inicializaci může vypadat například takto `npx @v-moravec/ui init`. V rámci tohoto příkazu je potřeba zkontrolovat, zda již soubory neexistují. Pokud by existovaly, měl by uživatel dostat možnost zvolit, zda chce přepsat stávající soubory nebo pokračovat bez změn. V případě chybějících souborů se automaticky vytvoří nové.

Získání souborů a závislostí by mělo být automatizováno, aby se zabránilo chybám a zjednodušil se vývoj knihovny. Nejjednodušší způsob je soubory zpřístupnit v rámci API části dokumentace (webové aplikace), odkud je možné s nimi pracovat na úrovní CLI. Jak toto funguje bude popsáno v kapitole 6. Pro tuto automatizaci je možné využít vlastních Nuxt modulů. Jako inspiraci lze použít modul, který využívá knihovna NuxtUI. [37]

5.6.2 Přidání komponent

Přidání komponent a bloků využije stejného principu jako inicializace projektu. Příkaz pro přidání komponenty může vypadat například takto:

```
npx @v-moravec/ui add [...componentName/blockName]
```

Při přidání komponenty je potřeba zkontrolovat, zda již soubory neexistují. Pokud by existovaly, měl by uživatel dostat možnost zvolit, zda chce přepsat stávající soubory nebo pokračovat bez změn. V případě chybějících souborů se automaticky vytvoří nové.

Z hlediska zdrojového kódu komponent a bloků bude potřeba více modulů, které na API vystaví všechny potřebné informace. Také bude potřeba v rámci každého souboru zjistit jeho závislosti a to na externích knihovnách, komponentách v rámci knihovny a dalších souborech, jako například composables. Toho je možné dosáhnout pomocí *pattern matchingu*.

5.6.3 Publikace balíčku

Jak bylo popsáno v sekci o technologiích, pro správu verzí balíčku se používá nástroj Changesets. Workflow pak vypadá následovně:

1. Vytvoření změn v kódu.

2. Vytvoření nové verze pomocí Changeset.
3. Nahrání změn na GitHub.
4. Spuštění GitHub Actions workflow pro publikaci balíčku.
5. Publikace nové verze balíčku.

Díky této automatizaci je možné rychle a bezpečně publikovat nové verze balíčku, aniž by bylo nutné manuálně upravovat potřebné soubory.

5.7 Závěr

V této kapitole byly popsány kroky a procesy vedoucí k navržení sbírky znovupoužitelných Nuxt komponent. Byly identifikovány a zdokumentovány prvky, které ovlivňují design a implementaci UI komponent.

Hlavním cílem navrhované sbírky komponent je vytvoření robustního základu, který umožní efektivní vývoj webových aplikací s využitím moderních frontend technologií. Tento návrh reflektuje nejnovější trendy ve vývoji webových aplikací a také přináší řadu nástrojů, které podporují celý životní cyklus vývoje od návrhu po nasazení a údržbu.

V další kapitole Implementace se práce zaměří na vytvoření navržených komponent. Diskutovány budou také výzvy a řešení, která byla během implementace identifikována.

Kapitola 6
Implementace

6.1 Úvod

V této kapitole bude zaměřena pozornost na implementaci sbírky znovupoužitelných Nuxt komponent. Implementace je fází vývoje, kde jsou teoretické návrhy a plánování transformovány do praktických a funkčních výsledků. Tato fáze zahrnuje konkrétní kroky a postupy potřebné k vytvoření komponent, dokumentace a CLI, které splňují stanovené požadavky a cíle.

Cílem této kapitoly je podrobně popsat strukturu projektu, jednotlivé části implementace a jejich vzájemné propojení. Dále bude věnována pozornost dokumentaci komponent, jejich kódu a blokům, které tvoří základ pro uživatelsky přívětivé a efektivní rozhraní. Implementace také zahrnuje tvorbu a použití příkazového řádku (CLI), který usnadňuje práci s komponentami a jejich integraci do různých projektů.

6.2 Struktura projektu

Pro zajištění efektivní správy a rozvoje projektu byla použita struktura monorepozitáře. Monorepozitář umožňuje uchovávat kód pro více balíčků v jednom repozitáři, což usnadňuje správu závislostí a zajišťuje konzistenci napříč celým projektem.

6.2.1 Struktura monorepozitáře

Struktura monorepozitáře je následující:

```
/v-moravec-ui
|-- /packages
|   |-- /cli
|   |   |-- /src
|   |   |   |-- /commands
|   |   |   |   |-- /add.ts
|   |   |   |   |-- /init.ts
|   |   |   |-- /utils
|   |   |   |   |-- /getPackageManager.ts
|   |   |   |   |-- /handleError.ts
|   |   |   |   |-- /checkNuxtProject.ts
|   |   |   |   |-- /installDependencies.ts
|   |   |   |   |-- /removeDuplicates.ts
|   |   |   |-- constants.ts
|   |   |   |-- index.ts
|-- /apps
|   |-- /docs
|   |   |-- /components
|   |   |   |-- /ui
|   |   |   |-- /block
|-- package.json
|-- pnpm-workspace.yaml
```

Výpis kódu 2 Struktura monorepozitáře

6.2.2 Popis jednotlivých částí

- /packages: Tento adresář obsahuje jednotlivé balíčky, které jsou součástí monorepozitáře.
- /cli: Tento balíček obsahuje CLI nástroje pro správu projektu. V adresáři src/commands se nachází jednotlivé příkazy jako add.ts a init.ts. V adresáři src/utils jsou různé užitečné funkce. Další soubory zahrnují constants.ts a index.ts, které obsahují základní konfiguraci a vstupní bod CLI nástroje.
- /apps: Tento adresář obsahuje jednotlivé aplikace. V tomto případě se jedná pouze o dokumentační aplikaci. V budoucnu mohou být přidány další aplikace, jako je například ukázková aplikace využívající komponenty.
- /docs: Dokumentační aplikace, která demonstruje použití vytvořených komponent a bloků. V adresáři components se nachází další podadresáře jako ui a block.
- /ui: Zde jsou umístěny všechny základní komponenty, které jsou součástí sbírky.
- /block: Zde jsou umístěny bloky, které kombinují jednotlivé komponenty do složitějších struktur.

- package.json: Hlavní soubor pro správu závislostí a skriptů na úrovni celého monorepozitáře.
- pnpm-workspace.yaml: Konfigurační soubor pro pnpm, který definuje strukturu workspace a zahrnuje všechny balíčky a aplikace.

Díky této struktuře je možné jednoduše vyvíjet komponenty, jejich dokumentaci a CLI bez nutnosti spravovat více repozitářů a závislostí.

6.3 Dokumentace

Implementace začala vytvořením nového projektu pomocí příkazu, který inicializuje projekt s připraveným Nuxt Content modulem.

```
npx nuxi@latest init v-moravec-ui -t content
```

Tento příkaz vytvoří základní strukturu projektu a přidá potřebné závislosti, což umožňuje rychlý start vývoje. Po vytvoření projektu byly přidány další moduly nezbytné pro vývoj, jako je Tailwind CSS, Nuxt Icon, Nuxt Color Mode a Nuxt SEO. Tyto moduly poskytují širokou škálu funkcí pro stylování, správu ikon, podporu tmavého režimu a optimalizaci SEO.

Aby bylo možné využít komponenty v rámci dokumentace, bylo potřeba je označit jako globální v souboru *nuxt.config.ts*. V rámci dokumentace je to dobré řešení pro minimalizaci duplikace kódu a zjednodušení práce s komponentami, ale pro koncové aplikace není nutné toto nastavení přidávat. Ukázka kódu 3 zobrazuje, jak lze označit komponenty jako globální.

```
1  components: {
2    dirs: [
3      {
4        path: '~/components/ui',
5        global: true,
6        prefix: 'Ui',
7      },
8      '~/components',
9    ],
10 },
```

Výpis kódu 3 Označení komponent jako globální

Nastavení Tailwind CSS vyžadovalo specifikaci složek, které má při *buildu* procházet a hledat styly, aby byly zahrnuty pouze potřebné styly díky využití *tree shakingu*. Toto nastavení bylo přidáno do souboru tailwind.config.ts. *Tree shaking* umožňuje minimalizaci velikosti výsledného balíčku tím, že odstraní nepoužívané styly, což zlepšuje výkon aplikace. Ukázka kódu 4 ukazuje, jak lze nastavit Tailwind CSS na procházení složek se zdrojovým kódem. Kromě toho je zde zobrazena i struktura, která se používá pro definování barev, velikostí a dalších vlastností.

```
1  import type { Config } from 'tailwindcss'
2  import typography from '@tailwindcss/typography'
3  import containerQueries from '@tailwindcss/container-queries'
4
5  export default <Partial<Config>>{
6      darkMode: 'class',
7      content: ['components/**/*.{vue,ts}', 'layouts/**/*.vue', 'pages/**/*.vue'],
8      theme: {
9          extend: {
10             colors: {
11                 background: 'hsl(var(--background))',
12                 primary: {
13                     DEFAULT: 'hsl(var(--primary))',
14                     contrast: 'hsl(var(--primary-contrast))',
15                 },
16                 secondary: {
17                     DEFAULT: 'hsl(var(--secondary))',
18                     contrast: 'hsl(var(--secondary-contrast))',
19                 },
20                 disabled: {
21                     DEFAULT: 'hsl(var(--disabled))',
22                     contrast: 'hsl(var(--disabled-contrast))',
23                 },
24                 border: 'hsl(var(--border))',
25             },
26             borderRadius: {
27                 sm: 'calc(var(--radius) - 4px)',
28                 md: 'calc(var(--radius) - 2px)',
29                 lg: 'var(--radius)',
30                 xl: 'calc(var(--radius) + 4px)',
31                 '2xl': 'calc(var(--radius) + 8px)',
32                 '3xl': 'calc(var(--radius) + 16px)',
33             },
34         },
35     },
36     plugins: [typography, containerQueries],
37 }
```

Výpis kódu 4 Konfigurační soubor pro Tailwind

6.4 Komponenty

Jedna ze zajímavějších věcí, kterou bylo potřeba vyřešit je jak psát znovupoužitelné komponenty. Je možné používat *sloty*, které mají jména, případně použít *wrappery* neboli obaly, které zachovávají logiku v sobě. V rámci konkurenčních řešení se často používají *wrappery*, ačkoliv to je pravděpodobně způsobeno tím, že se jedná o řešení pro React, potažmo o řešení převzaté z původní React verze. Ve Vue je jednodušší řešení přímo pomocí pojmenovaných *slotů* - poté

se kód zjednodušší a nemusí se používat direktivy jako *provide* a *inject*, které zhoršují čitelnost kódu.

Pojmenované *sloty* jsou užitečné pro složitější komponenty, které vyžadují více oblastí pro různé části obsahu. Pojmenované *sloty* se definují pomocí atributu name na <slot> značce a obsah se vkládá pomocí template tagu s atributem v-slot.

```
1  <!-- Card.vue -->
2  <template>
3    <div class="card">
4      <header>
5        <slot name="header"></slot>
6      </header>
7      <main>
8        <slot></slot>
9      </main>
10     <footer>
11       <slot name="footer"></slot>
12     </footer>
13   </div>
14 </template>
```

Výpis kódu 5 Pojmenované sloty - definice

```
1  <!-- ParentComponent.vue -->
2  <template>
3      <Card>
4      <template #header>
5          <h1>Card Header</h1>
6      </template>
7      <p>This is the main content.</p>
8      <template #footer>
9          <p>Card Footer</p>
10     </template>
11     </Card>
12 </template>
```

Výpis kódu 6 Pojmenované sloty - použití

6.4.1 Použité závislosti

Nuxt Icon

Tvorba vlastních ikon je časově náročná a může být složitá. Ideální je využít již hotové ikony, které jsou k dispozici na internetu. Je více možností jak toho dosáhnout, například lze kopírovat SVG kód a uložit jej do assetů nebo třeba do Vue komponent. To může být ale neefektivní a nepřehledné. Pro použití ikon z online zdrojů existuje modul Nuxt Icon, který zpřístupňuje mnoho ikon pro efektivní použití v aplikacích. Tento modul je velmi pokročilý a proto je v dokumentaci doporučené jej použít. Do budoucna je možné přidat obal, aby se zachovalo konzistentní pojmenování komponent.

Floating UI

Plovoucí prvky nejsou triviální na implementaci. Je potřeba zajistit, aby byly viditelné vždy, když je uživatel potřebuje. Pro tento účel byla použita knihovna Floating UI, která poskytuje jednoduché rozhraní pro vytváření plovoucích prvků. Tato knihovna je velmi užitečná pro vytváření interaktivních prvků, jako jsou tlačítka, odkazy a formuláře. Byl vytvořen jednoduchý *wrapper* formou *composable*. Díky této *composable* je možné vytvářet plouvoucí prvky v komponentách pomocí jednoho řádku s parametry.

```
1  import type { Placement, Strategy, MaybeElement } from '@floating-ui/vue'
2  import { useFloating, autoUpdate, offset, flip, shift } from '@floating-ui/vue'
3
4  export const useUiFloating = ({
5    placement = 'bottom',
6    offsetSize = 5,
7    strategy = 'fixed',
8  }: Partial<{
9    placement: Placement
10   offsetSize: number
11   strategy: Strategy
12 }>) => {
13   const anchor = ref<MaybeElement<HTMLElement>>(null)
14   const floating = ref<MaybeElement<HTMLElement>>(null)
15
16   const { floatingStyles } = useFloating(anchor, floating, {
17     strategy,
18     placement,
19     whileElementsMounted: autoUpdate,
20     middleware: [shift(), flip({ fallbackAxisSideDirection: 'end' }),
   ↪  offset(offsetSize)],
21   })
22
23   return { anchor, floating, floatingStyles } as const
24 }
```

Výpis kódu 7 Composable obalující Floating UI

```
1   const { anchor, floating, floatingStyles } = useUiFloating({ placement:
    ↪  props.placement, offsetSize: props.offsetSize })
```

Výpis kódu 8 Použití composable v komponentě

V budoucnu je možné, že nebude potřeba používat žádné knihovny díky nativní implementaci přímo v prohlížečích. Všechny moderní prohlížeče toto nové API podporují, avšak pro lepší podporu starších browserů je lepší použití výše zmíněné knihovny. [38]

Embla Carousel

Jedna z dalších komponent, která je složitá na implementaci je *carousel*. Pro tento účel byla použita knihovna Embla Carousel. Byl vytvořen *wrapper* nad jejím API v rámci několika komponent, které je možné dodatečně stylovat a ovlivňovat jejich chování.

6.5 Kód komponent a bloků

Bylo potřeba vymyslet, jak zobrazit kód bez jeho duplikace. V tom pomohl zdrojový kód knihovny Nuxt UI. Tato knihovna využívá principu vygenerování *endpointů* pro zobrazení zdrojového kódu přímo ze souborů s komponentami, popřípadě ze souborů s ukázkami použití. [37]

Pomocí několika modulů, které byly vytvořeny pro tento účel, bylo možné poskytnout kód komponent, bloků a dalších pomocných souborů na API aplikace, která obsahuje dokumentaci. Díky tomuto způsobu je možné kód zobrazit v dokumentaci a také použít v rámci CLI pro přidání komponent do projektu.

V první části bylo potřeba získat informace o komponentách a blocích. To bylo dosaženo pomocí *hooku*, který poskytuje seznam všech komponent a bloků. Tento seznam byl filtrován podle určitých kritérií a následně byl uložen do proměnné. Ukázka kódu 10 ukazuje, jak byl získán seznam komponent a bloků.

```
1  nuxt.hook('components:extend', async (_components) => {
2      components = _components
3          .filter(
4          (v) =>
5              v.shortPath.includes('components/content/examples/') ||
6              v.shortPath.includes('components/block/') ||
7              v.shortPath.includes('components/ui/') ||
8              v.shortPath.includes('components/content/blocks/')
9          )
10         .reduce((acc, component) => {
11         acc[component.pascalName] = component
12         return acc
13         }, {})
14     await stubOutput()
15 })
```

Výpis kódu 9 Prvotní získání informací o komponentách

Také bylo využito dalšího *hooku* `nuxt.hook('vite:extend', (vite: any) => {})`, který umožňuje při startu *buildu* volat funkci, ve které se všechny komponenty zpracují a přidají do proměnné. Konkrétně se volá funkce `await fetchComponents()`. V rámci této funkce se získané komponenty transformují do chtěné podoby. Následující ukázka vynechává kontroly dat, které nejsou nijak zajímavé a ukazuje pouze základní transformaci dat.

```
1   const code = await fsp.readFile(component.shortPath, 'utf-8')
2
3   function removeDuplicates(arr?: string[]) {
4       if (!arr) {
5       return undefined
6       }
7       return arr.filter((v, i) => arr.indexOf(v) === i)
8   }
9
10  components[component.pascalName] = {
11      code,
12      shortPath: component.shortPath,
13      pascalName: component.pascalName,
14  }
```

Výpis kódu 10 Transformace dat komponent

Poté se obsah proměnné uloží do souboru a jeho obsah se vrací v rámci *endpointu* API. Je nutné přidat *endpoint* do souboru *server/api/component-list.get.ts*, který obsahuje logiku pro získání dat. Jak se soubory spojí je ukázáno v příloze 13, obsah samotného *endpointu* v ukázce 14.

Ještě jedna důležitá část, která se v rámci ukázek kódu neřeší jsou různé závislosti na balíčky, další komponenty a podpůrné soubory jako *composables*. Aby bylo možné zjednodušit proces instalace nových komponent, bylo potřeba přidat tato data. Proto vznikly další moduly pro komponenty, bloky a *composables*. Transformace dat je zobrazena v ukázce 15.

Pro zobrazení kódu má Nuxt Content zabudovanou funkci. [39] Avšak bylo potřeba si vytvořit ještě vlastní komponentu, která bude zobrazovat kód získaný z API, protože zvýraznění kódu je nativně podporované pouze v Markdown souborech. Tato komponenta využívá Shiki *highlighteru*, tedy stejný princip, jaký používá Nuxt Content. Ukázka kódu 16 ukazuje, jak je možné vytvořit komponentu pro zobrazení kódu.

6.6 CLI

Vytvoření příkazu, který CLI umožňuje použít vypadá následovně:

```
1   export const add = new Command()
2   .name('add')
3   .description('add components and blocks to your application')
4   .argument('[componentsAndBlocks...]', 'components and blocks to add')
5   .action(async (componentsAndBlocks, opts) => {
6       // Logika pro přidání komponent a bloků
7   })
```

Výpis kódu 11 Příkaz pro přidání komponent a bloků

Pro kontrolu, zda jsou všechna data z API v pořádku byl použit Zod. Zde je ukázka vytvoření schématu a použití pro validaci dat:

```
1  const componentAndBlockSchema = z.object({
2    code: z.string(),
3    uiDependencies: z.string().array().optional(),
4    dependencies: z.string().array().optional(),
5    composableDependencies: z.string().array().optional(),
6    shortPath: z.string(),
7    pascalName: z.string(),
8  })
9
10 const componentsAndBlocksSchema = z.record(z.string(),
   ↪  z.array(componentAndBlockSchema))
11
12 const components = componentsAndBlocksSchema.safeParse(await (await
   ↪  fetch(BASE_URL + '/component-list')).json())
13 const blocks = componentsAndBlocksSchema.safeParse(await (await
   ↪  fetch(BASE_URL + '/block-list')).json())
14
15 if (!components.success) {
16     return handleError(components.error)
17 }
18 if (!blocks.success) {
19     return handleError(blocks.error)
20 }
21
22 // Pokračování práce s daty
```

Výpis kódu 12 Validace dat pomocí Zod

6.7 Závěr

V této kapitole byla podrobně popsána implementace sbírky znovupoužitelných Nuxt komponent. Byly popsány jednotlivé části projektu, jejich struktura a vzájemné propojení, čímž byl vytvořen komplexní nástroj pro vývoj a použití této kolekce. Dokumentace komponent byla zpracována tak, aby usnadnila jejich používání, což zajišťuje jejich efektivní integraci do různých projektů. Velká pozornost byla věnována nejen samotnému kódu komponent, ale také tvorbě pomocných nástrojů.

Kapitola 7
Testování

7.1 Úvod

Testování je zásadní součástí vývoje jakéhokoliv softwarového produktu. Zajišťuje nejen funkčnost a bezpečnost aplikací, ale také zlepšuje uživatelskou zkušenost a usnadňuje adaptaci produktu koncovými uživateli. V rámci této diplomové práce se zaměřujeme na testování dokumentace, procesů instalace a praktického nasazení kolekce komponent ve vývojářských projektech. Cílem je ověřit, jak intuitivní a přístupné jsou tyto aspekty pro vývojáře různých úrovní zkušeností.

Specifickým cílem testování v této práci je posoudit, zda dokumentace poskytuje jasné a snadno dostupné informace potřebné pro efektivní využití kolekce, zda jsou instrukce k instalaci dostatečně srozumitelné a zda je implementace komponent do různých typů projektů co možná nejsnadnější. Tato kapitola popisuje metodiku testování, vybrané testovací scénáře, a představuje výsledky získané z provedených testů.

Testování bylo zaměřeno na praktické aplikace v reálných vývojářských prostředích, což umožnilo shromáždit autentické zpětné vazby a poznatky, jež reflektují skutečné použití kolekce ve vývojovém procesu. Výsledky těchto testů poskytují cenné informace pro další vývoj a zlepšení dokumentace a uživatelské přívětivosti kolekce. Tato kapitola klade důraz na důkladné vyhodnocení shromážděných dat a jejich interpretaci.

7.2 Metodika testování

Metodika byla zvolena, aby zajistila komplexní a objektivní zhodnocení testované kolekce z různých perspektiv a umožnila tak formulovat podrobné a praktické doporučení pro další vývoj a zlepšování dokumentace a uživatelské přívětivosti kolekce.

7.2.1 Výběr respondentů

Pro účely tohoto testování byli vybráni tři vývojáři, každý s jinou úrovní zkušeností – od juniora přes mid-level až po senior vývojáře. Tento rozmanitý výběr zajišťuje, že zpětná vazba a zjištěné výsledky budou reflektovat široké spektrum potenciálních uživatelů kolekce. Každý respondent byl vybrán na základě jeho předchozího zapojení do podobných projektů a zkušeností s používáním *frameworku* Nuxt/Vue.

7.2.2 Prostředí testování

Testování bylo provedeno ve dvou režimech - online a osobně. Každý respondent pracoval na vlastním počítači, což umožnilo simulaci skutečných pracovních podmínek. Tento přístup pomohl zajistit, že všechny zjištěné výsledky jsou přímo spojeny s testovanými aspekty kolekce, bez rušivých vlivů z externího prostředí.

7.2.3 Proces testování

Každý respondent dostal sérii úkolů odpovídajících jednotlivým testovacím scénářům. respondentům byly poskytnuty instrukce, které zahrnovaly specifické kroky a očekávané výstupy pro každý úkol. Účastníci byli požádáni, aby nahlas dokumentovali svůj postup a jakékoliv problémy nebo otázky, které během testování vznikly. Záznam byl prováděn pomocí nahrávání obrazovky a zapisování poznámek.

7.2.4 Zpětná vazba

Po dokončení testovacích úkolů byla od respondentů vyžádána zpětná vazba prostřednictvím rozhovorů a diskuzí nad problémy. Cílem bylo, aby bylo možné získat hlubší vhledy do uživatelské zkušenosti a identifikovaly klíčové oblasti pro zlepšení. Otázky zahrnovaly hodnocení obtížnosti úkolů, kvality informací v dokumentaci a celkového vnímání kolekce. Díky zpětné vazbě bylo možné iterovat a upravovat dokumentaci a procesy instalace tak, aby lépe vyhovovaly potřebám uživatelů.

7.3 Testovací scénáře

Každý ze scénářů byl navržen tak, aby testoval specifické aspekty kolekce a zajišťoval přesné a relevantní zpětné vazby. Výsledky těchto testů poskytují důležité informace pro další vylepšení uživatelské přívětivosti a efektivity dokumentace a instalace.

1. Scénář: Proces instalace
   - Cíl: Ověřit, zda jsou instrukce k instalaci jasné a snadno pochopitelné.
   - Kroky úkolu
     a. Otevřete si dokumentaci a krátce se seznamte s hlavními principy kolekce.
     b. Inicializujte nový Nuxt projekt pomocí příkazu nuxi init my-project v příkazové řádce.
     c. Inicializujte kolekci ve vytvořeném projektu.
   - Očekávaný výsledek: Instrukce k instalaci by měly být formulovány tak, aby vývojáři mohli kolekci nainstalovat bez potřeby vyhledávání dodatečných informací nebo žádání o vnější pomoc. Úspěšná instalace bez komplikací a dotazů ukazuje, že dokumentace je dobře strukturovaná a efektivní.
2. Scénář: Přehlednost dokumentace
   - Cíl: Ověřit, jak snadno mohou vývojáři najít informace, které potřebují.
   - Kroky úkolu
     a. Najděte v dokumentaci instrukce pro použití komponenty tlačítko a použijte ji.
     b. Najděte v dokumentaci instrukce pro použití komponenty tabulka a použijte ji.
     c. Najděte v dokumentaci instrukce pro použití komponenty sidebar a použijte ji.

- Očekávaný výsledek: Vývojáři by měli být schopni rychle a intuitivně najít potřebné informace, aniž by museli procházet nepřehlednou nebo zmatenou dokumentaci. Rychlost a snadnost nalezení informací jsou klíčové metriky pro úspěch v tomto scénáři.

3. Scénář: Použití sbírky ve vlastních projektech
   - Cíl: Zjistit, jak snadné je používat komponenty kolekce v různých projektech.
   - Kroky úkolu
     a. Vytvořte základní strukturu *landing page* pro imaginárního klienta s následujícími sekcemi.
     b. *Hero sekce*: Obsahuje slogan, tlačítka s odkazy a *carousel* s obrázky.
     c. Reference: Sekce pod *hero sekcí*, která ukazuje reference na práci klienta.
     d. Doplňující sekce: Vytvořte sekci dle vlastní představy, která by mohla být vhodným doplněním mezi referencemi a kontaktem.
     e. Kontaktní formulář: Na konci stránky.
   - Očekávaný výsledek: Vývojáři by měli být schopni bez problémů použít vybrané komponenty a bloky a integrovat je do svých projektů. Snadnost integrace a flexibilita komponent při použití v různých kontextech a projektech jsou klíčové ukazatele úspěšnosti.

7.4 Výsledky testování

7.4.1 Respondent 1

Nalezené problémy

1. Použitelnost sidebaru: Respondent nevěděl, jak do sidebaru přidat svůj obsah.

2. Typy dat z komponent: Bylo zjištěno, že export typů dat z komponent je důležitý pro jejich snadné použití, avšak u některých komponent chyběl.

3. *Container* komponenta: Absence komponenty pro kontejner ztížila respondentovi správu layoutu aplikací.

4. Typescript závislost: Objevil se *error*, kdy v konzoli bylo zobrazeno, že chybí Typescript závislosti. Respondent byl nucen ručně nainstalovat chybějící balíčky, ale vzhledem k jasné zprávě v konzoli to nebyl velký problém. Prozatím není nutné přidávat závislost automaticky, pokud by se problém opakoval, mohlo by to být užitečné.

5. Carousel a *hot reload*: Upozornění na *hot reload* pro *carousel* komponentu bylo považováno za užitečné, protože pomáhá v případě, že dojde k dynamickým změnám obsahu.

6. Odstranění odkazů: Testování odhalilo, že přítomnost odkazů v ukázkách (např. blocích) může vést k záměně s dokumentací.

7. Rozdíl mezi ukázkou a implementací komponenty: Respondent byl zmaten při rozhodování, kdy použít ukázkový kód a kdy se jedná o kód komponenty.

8. Rozlišení mezi znovupoužitelnými a specifickými bloky: Respondent měl problémy rozlišovat mezi bloky, které jsou znovupoužitelné a těmi, které vyžadují úpravy.

Pozitivní zjištění

1. Automatická instalace závislostí: Automatizace instalace závislostí byla hodnocena velmi kladně, neboť respondentovi usnadnila začlenění komponent do jeho projektu.

2. Efektivita: Dokázal si představit, že základní UI u nových projektů dokáže vytvořit za krátkou dobu, což naznačuje, že kolekce je efektivní a snadno použitelná.

3. Přirovnání k jednoduchosti použití Wordpress šablon, s rozdílem v moderním stacku.

Provedené změny

1. Zvýraznění bloků v dokumentaci - představení bloků, jak se mají používat a jaké jsou jejich dva typy.

2. Úprava stránky bloků - přidání štítků, které označují, zda se jedná o znovupoužitelný blok nebo blok, který vyžaduje úpravy.

3. Přidání ukázky, jak přidat obsah do sidebaru.

4. Přidání upozornění na restart development serveru. [1]

7.4.2 Respondent 2

Nalezené problémy

1. Nebylo jasné odkud volat příkaz pro přidání komponent.

2. Nebylo jasné, že se přidává do projektu zdrojový kód komponent - respondent si myslel, že se komponenty přidávají do složky node_modules.

3. Nedostatečné informace v rámci CLI. Např. chybějící informace o tom, že možná bude potřeba restartovat development server po instalaci, aby vše fungovalo správně.

4. Respondentovi dělalo problém najít bloky i přes přidanou informaci v dokumentaci.

5. V dokumentaci chybí informace o typography komponentách.

6. Nejasné jak se mají používat bloky - respondent blok přidal pomocí CLI, ale místo jeho použití ještě manuálně vykopíroval kód do projektu.

Pozitivní zjištění

1. Dokumentace byla hodnocena jako přehledná a snadno čitelná.

2. Respondent ocenil myšlenku přístupu ke zdrojovému kódu.

[1] Potřeba restartovat development server po instalaci je někdy potřeba, jindy zase ne. Kód se může rozbít kvůli přidání nových závislostí napozadí nebo kvůli jiným změnám. Upozornění je tedy užitečné ve chvíli, kdy uživatel narazí na neočekávané chování, které může být pomocí tohoto kroku vyřešeno.

Provedené změny

1. Do návodu na instalaci přidána informace o volání CLI z kořenového adresáře projektu.
2. Zvýraznění bloků v dokumentaci - přidány odkazy přímo na stránky konkrétních bloků.
3. Přidány informace k zdrojovým kódům v dokumentaci - k čemu slouží a jak je využívat.
4. Úprava textových informací v rámci CLI - přidání upozornění na restart development serveru a hlášky o přidání zdrojového kódu v rámci složek projektu.
5. Přidání typography komponent do dokumentace.

7.4.3 Respondent 3

Nalezené problémy

1. Hlášky o použití u zdrojových kódu někdy nejsou vidět, protože jsou pod dlouhým kódem.
2. Závislost na Typescriptu by měla být automaticky přidána, protože jinak byl respondent zmaten a myslel si, že chyba je v kódu komponenty.
3. Je nezvyklé vzít si celou sekci hotovou - je pro něj automatické si vytvořit vlastní komponentu.
4. Respondent nenavšívil stránku s bloky, protože se proklikával přímo na konkrétní stránky z dokumentace.

Pozitivní zjištění

1. Respondent ocenil jednoduchost a snadnost použití kolekce.
2. Základní struktura kolekce byla hodnocena jako přehledná a snadno použitelná.

Provedené změny

1. Přesunutí informací nad zdrojový kód v dokumentaci.
2. Přidání poznámky o nutnosti instalace Typescript závislostí u komponenty Carousel.
3. Přidání informací o blocích na *landing page*.

7.5 Závěr

Testování sbírky znovupoužitelných Nuxt komponent přineslo cenné poznatky o jejích silných stránkách i oblastech pro zlepšení. Celkově byla dokumentace hodnocena pozitivně, zejména co se týče její přehlednosti a srozumitelnosti, avšak určité části mohou být dále vylepšeny, zejména v oblasti ukázek možností komponent.

Výsledky testování ukázaly, že jednoduchá rozšiřitelnost a flexibilita komponent jsou důležitými prvky kolekce, které vývojáři ocenili. Umožňují rychlou integraci a usnadňují vytvoření webových stránek s použitím komponent a bloků.

Na základě získané zpětné vazby byly provedeny některé úpravy, jako například přidání informací o volání CLI z kořenového adresáře projektu, zvýraznění bloků v dokumentaci, a přidání poznámky o nutnosti instalace Typescript závislostí u některých komponent.

Tato testovací fáze poskytla informace pro další vývoj a zlepšování dokumentace a uživatelské přívětivosti kolekce. Bude důležité pokračovat v implementaci doporučených změn a sbírat další zpětnou vazbu, aby se zajistilo, že sbírka komponent zůstane užitečným a efektivním nástrojem.

Kapitola 8
Možnosti rozšíření

V této kapitole se zaměříme na možnosti rozšíření sbírky znovupoužitelných Nuxt komponent. Po úspěšné implementaci základní sady komponent je důležité zvážit, jakým způsobem lze tuto sbírku dále rozšiřovat a přizpůsobovat novým požadavkům.

Budou zde prozkoumány možnosti přidávání nových komponent, bloků a funkcionalit, které mohou zvýšit hodnotu a využitelnost celé sbírky. Dále se bude diskutovat o integraci dalších nástrojů a technologií, které mohou usnadnit vývoj a zlepšit uživatelskou zkušenost.

Cílem této kapitoly je poskytnout ucelený pohled na možnosti rozšíření a ukázat, jak lze existující sbírku komponent nadále zdokonalovat a přizpůsobovat novým výzvám a příležitostem.

8.1 Přidání bloků

Rozšíření sbírky o více bloků a jejich verzí umožňuje uživatelům přizpůsobit si aplikace s větší flexibilitou a znovupoužitelností. Bloky mohou být navrženy tak, aby pokrývaly širokou škálu běžných i specifických případů, od navigačních prvků po složité sekce.

8.2 Přidání komponent

Dodatek nových komponent do knihovny by měl reflektovat aktuální trendy ve webovém designu, zatímco zároveň by měl pokrývat nové požadavky uživatelů. Například integrace komponent pro pokročilé uživatelské interakce jako jsou drag-and-drop prvky, pokročilé formulářové ovládací prvky a interaktivní grafy.

8.3 Přidání ukázkových stránek/aplikací

Vytvoření sad ukázkových stránek nebo aplikací využívajících komponent poskytuje uživatelům možnost rychlejšího vývoje. Tyto stránky mohou zahrnovat různé scénáře použití od jednoduchých landing stránek po složitější platformy.

8.4 Přidání možností konfigurace

Umožnění uživatelům konfigurovat komponenty pomocí vlastního konfiguračního souboru nabízí vyšší úroveň personalizace a kontrolu. Uživatelé již nyní mohou definovat globální nastavení,

jako jsou motivy, typografie a barvy, které budou automaticky aplikovány na všechny komponenty. Nicméně někteří uživatelé mohou chtít konfigurovat např. umístění komponent a bloků a jejich prefix. Prozatím pro konzisteci napříč projekty není možné tyto nastavení měnit, avšak v budoucnu je možné přehodnotit tuto možnost.

8.5 Figma Kit

Momentálně je Figma Kit spíše statický, jeho aktualizace o responzivní design a interaktivní prvky umožní designérům a vývojářům lepší spolupráci a rychlejší iteraci designů. Tato rozšíření mohou zahrnovat varianty komponent pro různé obrazovky a zařízení, interaktivní stavy pro hover a kliknutí, a integrované prototypování funkcí. Poskytnutí těchto nástrojů v Figma kitu umožní týmům efektivně testovat a upravovat UI/UX designy před jejich implementací.

8.6 Testy komponent

Automatizované testování může být zajímavé rozšíření pro zajištění kvality a udržitelnosti knihovny komponent. Použití robustního testovacího *frameworku*, který umožňuje jak jednotkové testy, tak integrační a end-to-end testy, značně zvyšuje spolehlivost a snižuje pravděpodobnost chyb při budoucích aktualizacích komponent. Použití nástrojů jako *@nuxt/test-utils* nebo Cypress poskytuje komplexní pokrytí testů a integrovanou podporu pro simulaci uživatelských interakcí a asertace stavu UI. Navrhnout testovací scénáře, které mapují běžné a okrajové případy užití komponent, zvyšuje důvěru v kód a usnadňuje kontinuální integraci a nasazení. Na druhou stranu je otázka, jak tyto testy integrovat do koncových aplikací a zda je to vůbec možné.

8.7 Vytváření stylů

Vývoj „Theme Creator“ nástroje, který bude integrován přímo do dokumentace, umožní uživatelům v reálném čase vizualizovat a upravovat vzhled komponent. Tento nástroj by měl nabídnout intuitivní rozhraní pro výběr barev, fontů, zaoblení hran a dalších stylistických prvků. Takový nástroj značně usnadní proces designu a vývoje tím, že poskytne okamžitou zpětnou vazbu na změny provedené v tématu a umožní uživatelům exportovat výsledné styly pro použití ve svých projektech.

Kapitola 9
Závěr

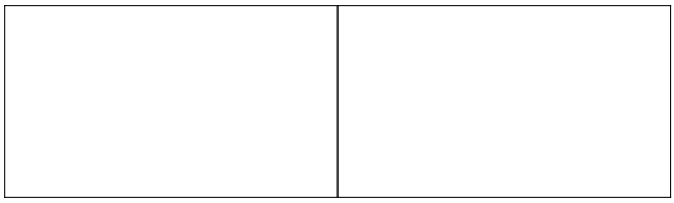
Cílem práce bylo analyzovat současný stav komponentových knihoven a na základě toho navrhnout a implementovat vlastní řešení. Povedlo se mi vytvořit ekosystém, kam budu moci v rámci diplomové práce přidávat další komponenty. Mezi části tohoto ekosystému patří dokumentace, která je dostupná na adrese `https://ui.vojtamoravec.cz`. Dále se jedná o knihovnu komponent ve Figmě `https://www.figma.com/community/file/1331024596435699926/v-moravec-ui`. Poslední část je ještě neimplementované CLI, které bude pomáhat s instalací komponent. Kromě samotných komponent počítám ještě s vylepšením celého ekosystému.

Příloha A

Snímky dokumentace

A.1 Úvodní obrazovka

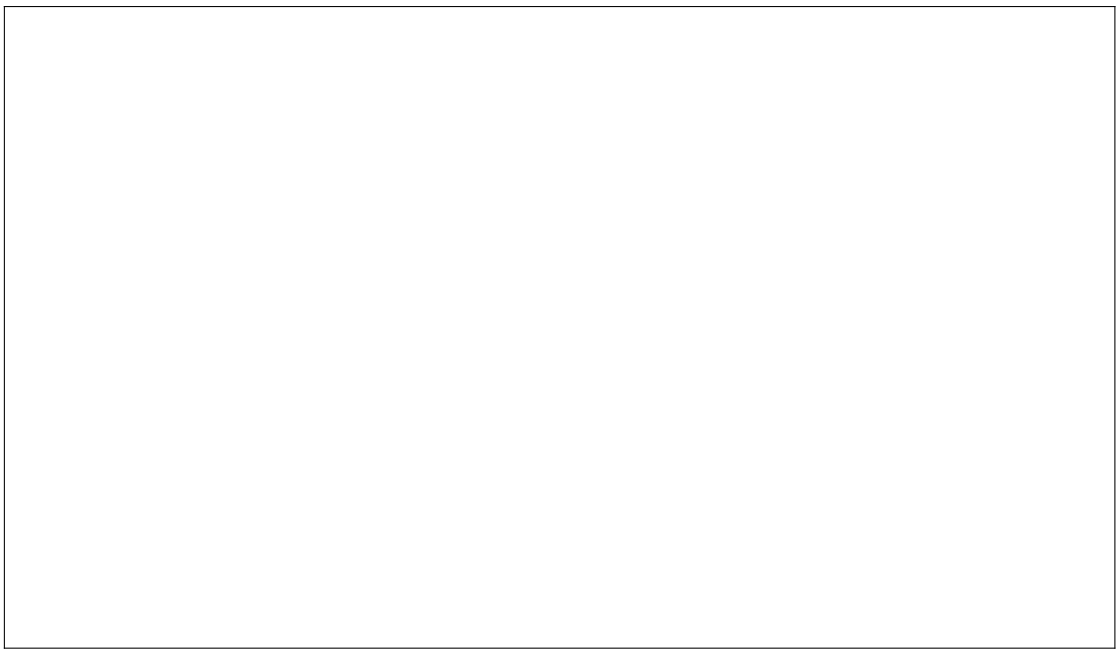
Obrázek A.1 Úvodní obrazovka



(a) Tmavá verze **(b)** Modrá verze

Obrázek A.2 Stylování

A.2 Komponenty



Obrázek A.3 Zobrazení komponenty

Obrázek A.4 Kód komponenty

A.3 Elementy

Obrázek A.5 Element složený z komponent

Příloha B

Ukázky kódu

```
1   nuxt.hook('nitro:config', (nitroConfig) => {
2   nitroConfig.virtual = nitroConfig.virtual || {}
3     nitroConfig.virtual['#component-list/nitro'] = () =>
4         readFileSync(join(nuxt.options.buildDir, '/component-list.mjs'), 'utf-8')
5   })
6
7   addServerHandler({
8     method: 'get',
9     route: '/api/component-list/:component?',
10    handler: resolver.resolve('../server/api/component-list.get'),
11  })
```

Výpis kódu 13 Vytvoření endpointu pro získání kódu komponent

```
1  import { defineEventHandler, createError, appendHeader } from 'h3'
2  import { pascalCase } from 'scule'
3  // @ts-expect-error
4  import components from '#code-examples/nitro'
5
6  export default defineEventHandler((event) => {
7    appendHeader(event, 'Access-Control-Allow-Origin', '*')
8    const componentName = (event.context.params['component?'] ||
   ↪   '').replace(/\.json$/, '')
9    if (componentName) {
10     const component = components[pascalCase(componentName)]
11     if (!component) {
12       throw createError({
13         statusMessage: 'Examples not found!',
14         statusCode: 404,
15       })
16     }
17     return component
18   }
19 })
```

Výpis kódu 14 Endpoint pro získání kódu komponent

```
1   const code = await fsp.readFile(component.shortPath, 'utf-8')
2
3   const script = code.match(/<script[^]*?<\/script>/ms)?.[0]
4   const template = code.match(/<template[^]*?<\/template>/ms)?.[0]
5
6   const dependencies = script
7     ? removeDuplicates(
8         script
9         .match(/import[^]*?from[^]*?['"]([^'"]+)['"]/gm)
10        ?.map((dependency) => {
11            return dependency.split('from')[1].trim().replace(/['";]/g, '')
12        })
13        .filter((v) => !v.startsWith('.') && !v.startsWith('~') && v !== 'vue' &&
    ↪   v !== 'nuxt')
14    )
15    : undefined
16
17  const composableDependencies = script
18    ? removeDuplicates(
19        script
20        .match(/use[a-zA-z]*?\(/gm)
21        ?.map((dependency) => {
22            return dependency.trim().replace(/['"();]/g, '')
23        })
24        .filter((v) => !v.startsWith('.') && !v.startsWith('~') && v !== 'vue' &&
    ↪   v !== 'nuxt')
25    )
26    : undefined
27
28  const uiDependencies = template
29    ? removeDuplicates(
30        template.match(/<Ui[^>]+>/gm)?.map((v) =>
31        v
32            .slice(1, -1)
33            .split(' ')[0]
34            .trim()
35            .split(/(?=[A-Z])/)[1]
36            .toLowerCase()
37        )
38    )
39    : undefined
40
41  components[component.pascalName] = {
42    code,
43    uiDependencies,
44    dependencies,
45    composableDependencies,
46    shortPath: component.shortPath,
47    pascalName: component.pascalName,
48  }
```

Výpis kódu 15 Transformace dat komponent a bloků pro využití v rámci CLI

```
1  <script lang="ts" setup>
2  import { transformContent } from '@nuxt/content/transformers'
3
4  const props = defineProps<{
5    name: string
6    class?: string
7  }>()
8
9  const data = await fetchCodeExample(props.name)
10
11 const hasCode = computed(() => data?.code)
12
13 const highlighter = await loadShiki()
14 const { data: ast } = await
   ↪  useAsyncData(`content-example-${props.name}-ast`, () =>
15   transformContent('content:_markdown.md', `\`\`\`vue\n${data?.code ??
   ↪  ''}\n\`\`\``, {
16     markdown: {
17       highlight: {
18         highlighter,
19         theme: {
20           light: 'poimandres',
21           default: 'poimandres',
22           dark: 'poimandres',
23         },
24       },
25     },
26   })
27 )
28 </script>
29
30 <template>
31   <div class="my-4" v-if="hasCode">
32     <ContentRenderer :value="ast" class="[&>div>pre]:!mt-0
   ↪  [&>div>pre]:overflow-auto [&>div>pre]:!rounded-t-none" />
33   </div>
34 </template>
```

Výpis kódu 16 Komponenta pro zvýraznění kódu

Zdroje

1. *NuxtLabs*. NuxtLabs. Dostupné také z: https://nuxtlabs.com.
2. *SuperchargedGitHub experience*. NuxtLabs. Dostupné také z: https://volta.net.
3. *Official Nuxt UI Pro Templates*. NuxtLabs. Dostupné také z: https://ui.nuxt.com/pro/templates.
4. *The official Nuxt UI Figma Kit*. Nuxt. Dostupné také z: https://www.figma.com/community/file/1288455405058138934/nuxt-ui.
5. *Radix UI*. Radix UI. Dostupné také z: https://www.radix-ui.com/primitives/docs/overview/introduction.
6. *shadcn/ui*. shadcn. Dostupné také z: https://ui.shadcn.com/docs.
7. *@shadcn/ui - Design System*. Pietro Schirano. Dostupné také z: https://www.figma.com/community/file/1203061493325953101/shadcn-ui-design-system.
8. *Why Are Large Open-Source Projects Ditching TypeScript?* Manusha Chethiyawardhana. Dostupné také z: https://javascript.plainenglish.io/why-are-large-open-source-projects-ditching-typescript-bfe788992b3a.
9. *The @returns tag*. JSDoc. Dostupné také z: https://jsdoc.app/tags-returns.
10. *Big projects are ditching TypeScript... why?* Fireship. Dostupné také z: https://www.youtube.com/watch?v=5ChkQKUzDCs.
11. *What is Vue?* Vue. Dostupné také z: https://vuejs.org/guide/introduction.html#what-is-vue.
12. *Rendering Modes*. Nuxt. Dostupné také z: https://nuxt.com/docs/guide/concepts/rendering.
13. *State of CSS*. Devographics. Dostupné také z: https://2023.stateofcss.com/en-US/css-frameworks.
14. *State of Frontend 2022*. The Software House. Dostupné také z: https://tsh.io/state-of-frontend.
15. *Utility-First Fundamentals*. Tailwind. Dostupné také z: https://tailwindcss.com/docs/utility-first.
16. *Preflight*. Tailwind. Dostupné také z: https://tailwindcss.com/docs/preflight.
17. *TypeScript is JavaScript with syntax for types.* Typescript. Dostupné také z: https://www.typescriptlang.org.
18. *Content made easy for Vue Developers*. Nuxt. Dostupné také z: https://content.nuxt.com.

19. *Introduction*. Anthony Fu. Dostupné také z: https://shiki.style/guide.
20. *Headless UI*. Tailwind Labs. Dostupné také z: https://headlessui.com.
21. *Floating UI*. Floating UI. Dostupné také z: https://floating-ui.com.
22. *Get Started*. Embla Carousel. Dostupné také z: https://www.embla-carousel.com/get-started.
23. *node.js command-line interfaces made easy*. commander.js. Dostupné také z: https://github.com/tj/commander.js.
24. *Citty - Elegant CLI Builder*. UnJS. Dostupné také z: https://unjs.io/packages/citty.
25. *Ora - Elegant terminal spinner*. Sindre Sorhus. Dostupné také z: https://github.com/sindresorhus/ora.
26. *Lightweight, beautiful and user-friendly interactive prompts*. Terkel. Dostupné také z: https://github.com/terkelg/prompts.
27. *Execa - Process execution for humans*. Sindre Sorhus. Dostupné také z: https://github.com/sindresorhus/execa.
28. *Documentation*. Zod. Dostupné také z: https://zod.dev.
29. *Bundle your TypeScript library with no config*. tsup. Dostupné také z: https://tsup.egoist.dev.
30. *Worskspace*. pnpm. Dostupné také z: https://pnpm.io/workspaces.
31. *A tool to manage versioning and changelogs with a focus on multi-package repositories*. changesets. Dostupné také z: https://github.com/changesets/changesets.
32. *About npm*. npm. Dostupné také z: https://docs.npmjs.com/about-npm.
33. *Let's build from here*. Github. Dostupné také z: https://github.com.
34. *Connect, protect and build everywhere*. Cloudflare. Dostupné také z: https://www.cloudflare.com.
35. *Developer Platform*. Cloudflare. Dostupné také z: https://developers.cloudflare.com.
36. *Build fast sites. In record time*. Cloudflare. Dostupné také z: https://pages.cloudflare.com.
37. *Content code module*. Nuxt. Dostupné také z: https://github.com/nuxt/ui/blob/dev/docs/modules/content-examples-code.ts.
38. *Popover API*. MDN Web Docs. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/API/Popover_API.
39. *Code Highlighting*. Nuxt. Dostupné také z: https://content.nuxt.com/usage/markdown#code-highlighting.