EKONOMICKÁ FAKULTA TUL

Bakalářská práce

Vývoj hypermedia aplikace pro e-commerce

Studijní program: B0688P140002 Informační management
Autor práce: **Tobiáš Tláskal**
Vedoucí práce: Mgr. Tomáš Žižka, Ph.D.
Katedra informatiky

Liberec 2024

EKONOMICKÁ FAKULTA TUL

Zadání bakalářské práce

Vývoj hypermedia aplikace pro e-commerce

Jméno a příjmení:	**Tobiáš Tláskal**
Osobní číslo:	E21000238
Studijní program:	B0688P140002 Informační management
Zadávající katedra:	Katedra informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Hypermedia a RESTful pro vývoj webových aplikací
2. Frontendové knihovny pro webové a hypermedia aplikace
3. Návrh e-commerce hypermedia aplikace
4. Implementace a testování
5. Zhodnocení a doporučení

Rozsah grafických prací:	
Rozsah pracovní zprávy:	30 normostran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

- CASTRO, Elizabeth a Bruce HYSLOP, 2022. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. 2. vydání. Brno: Computer Press. ISBN 978-80-251-5045-0.
- GROSS, Carson; Adam STEPINSKI a Deniz AKŞIMŞEK, 2023. *Hypermedia Systems*. Independently published. ISBN 979-83-940-2514-3.
- MANN, Eric, 2023. *PHP Cookbook: modern code solutions for professional PHP developers*. First edition. Sebastopol: O'Reilly. ISBN 978-1-09-812132-7.
- SOTNIK, Svitlana; Volodymyr MANAKOV a Vyacheslav LYASHENKO, 2023. online. Overview: PHP and MySQL Features for Creating Modern Web Projects. *International Journal of Academic Information Systems Research*, vol. 7, no. 1, s. 11–17. ISSN 2643-9026. Dostupné z: https://openarchive.nure.ua/server/api/core/bitstreams/99c2a8e9-aefd-47d4-a06f--838650b760db/content.
- ZANDSTRA, Matt, 2021. *PHP 8 objects, patterns, and practice: mastering OO enhancements, design patterns, and essential development tools*. Sixth edition. New York, U.S.A..: Apress. ISBN 978-1-4842-6790-5.

Konzultant: Ing. Jaromír Műller, GetReady s.r.o., SW architekt, Magento developer

Vedoucí práce:	Mgr. Tomáš Žižka, Ph.D. Katedra informatiky
Datum zadání práce:	1. listopadu 2023
Předpokládaný termín odevzdání:	31. srpna 2025

L.S.

doc. Ing. Aleš Kocourek, Ph.D.
děkan

Mgr. Tereza Semerádová, Ph.D.
garant studijního programu

V Liberci dne 1. listopadu 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

6. května 2024 Tobiáš Tláskal

Vývoj hypermedia aplikace pro e-commerce

Anotace

Tato bakalářská práce se zabývá využitím knihovny HTMX při vývoji single-page webové aplikace. Ta má podobu pokladny pro e-commerce webové stránky. Účelem práce je implementace HTMX knihovny do procesu vývoje aplikace tak, aby bylo možné omezit využití jazyka JavaScript a vytvořit tak přehlednější kód. Výsledkem práce je funkční aplikace, která je schopná provádět aktualizace části obsahu bez využití JavaScriptu. Pro využití knihovny byl vytvořen jednoduchý framework, který umožňuje sestavování takových webových stránek.

Klíčová slova: vývoj, web, HTMX, e-commerce

Development of e-commerce hypermedia application

Annotation

This bachelor thesis focuses on the use of the HTMX library in the development of a single-page web application. The app has a form of checkout for an e-commerce website. The purpose of the thesis is to implement the HTMX library into the development process in order to reduce the use of JavaScript language and thus create a more transparent code. The outcome is a functional application that is able to make updates to parts of the content without the use of JavaScript. A simple framework has been created for the use the library to support the assembly of such web pages.

Keywords: development, web, HTMX, e-commerce

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Tomáši Žižkovi, PhD. za pomoc při tvorbě práce, Ing. Jaromíru Müllerovi a Ing. Václavu Müllerovi za pomoc při vývoji aplikace. Děkuji také mé manželce Tereze za její podporu.

Obsah

Seznam obrázků

Seznam zkratek

AJAX	Asynchronous JavaScript And XML
BE	Backend
CDN	Content Delivery Network
E2E	End-to-end
FE	Frontend
JS	JavaScript

Úvod

Webové stránky nabízející produkty k prodeji nejsou ve 20. letech 21. století již žádnou novinkou. I přesto ale odvětví e-commerce obchodů zažívá nepřetržitý silný růst popularity. Virtuální trh je dnes velice rozmanitý. E-commerce platformy se stále inovují a trendy pro vývoj optimální aplikace, tedy aplikace optimalizované pro co možná nejvyšší ziskovost a maximální komfort uživatele i administrátora, se stále mění. I kvůli této dynamičnosti je složité trendy předpovědět. I přesto je ale zřejmá popularita jazyka JavaScript (dále také JS) a frameworků na něm postavených.

Jedním z významných hybatelů e-commerce světa byla pandemie viru Covid-19. Trh se kvůli restrikcím, obavám a nejistotě situace začal ještě rychleji přesouvat do virtuálního prostoru. Základní rostoucí trend popularity e-shopů s přidaným zájmem způsobeným pandemií tak vyvíjejí stále vyšší tlak na vývoj uživatelsky přívětivých aplikací.

Metodiky vývoje i technologie pro vývoj internetových aplikací mají za sebou už značnou cestu, kdy první weby spatřily světlo světa na konci 20. století. Od té doby se web vyvíjel, a stále vyvíjí, bez přestávky. Programátoři se musí neustále přizpůsobovat různým technologickým změnám a sebevzdělávání je nedílnou součástí života moderního programátora. Co vše ale programátor musí umět pro to, aby byl schopen vyvinout funkční aplikaci, která splní svůj účel?

Dnes je trendem využití rozsáhlých frameworků a systémů pro programování, sestavování, řízení i provozování (nejen) webových aplikací. Pro určitá využití je taková komplexní informační architektura vhodná. Přístup k vývoji aplikací je standardizovaný, pro obsluhu procesu se využívají definované postupy. Projekty velkých měřítek je tak možné v dlouhodobém horizontu vyvíjet systematicky a udržitelně.

Pak se ale naskýtá otázka, kdy dává smysl investovat čas a peníze do takových složitých systémů? Autor práce zastává názor, že mimo velké aplikace nebo aplikace s vysokými výpočetními nároky není vhodná snaha využívat rozsáhlou architekturu v projektech, pro které není taková robustnost zásadní.

Základní technologie, jako jsou PHP, CSS a HTML, jsou už dnes vyspělé a poskytují širokou škálu funkcí. Poměrně nedávno byla také vydána JavaScriptová knihovna HTMX, která se snaží znovuobjevit jednoduchost REST dotazů za použití systémů orientovaných na hypermedia a využít je namísto složitých JS struktur.

Tato práce se snaží otestovat tento přístup. Je možné s minimem technologií a zdánlivě jednodušším přístupem systémů zpracovávajících hypermedia vyvinout aplikaci, která by mohla zastat funkce takových aplikací, které jsou vyvinuty pomocí mainstreamových nástrojů?

Cílem práce je využít princip hypermedia-oriented systémů a vytvořit tak konceptuální webovou aplikaci za použití tohoto méně běžného přístupu k programování webových aplikací. Dalším vyplývajícím cílem je pak vyvinutí aplikace, která je schopná poskytnout podobnou funkcionalitu, jakou přináší mnohem složitější systémy v oblasti prezentování, aktualizace a interakce s front-end (dále také FE) částí aplikace.

Důraz není kladen na vizuální podobu aplikace ani na následné zpracování dat získaných z aplikace. Záměr je otestování nové HTMX knihovny a smyslu jejího využití pro vytváření běžných webových aplikací.

Snahou autora je minimalizovat počet základních technologií pro vývoj a zároveň maximalizovat využití jejich potenciálu pomocí knihovny HTMX.

1 Technologie

Základem webových aplikací je hypertextová stránka, která prezentuje obsah uživateli za pomoci webového prohlížeče. Tyto stránky jsou stylovány kaskádovými styly, které upravují jejich grafickou podobu. Všechen tento obsah poskytuje nejčastěji webový server. Následné interakce uživatele se zobrazenou stránkou jsou běžně obslouženy buďto samotným HTML, nebo JavaScriptovým kódem. Tato práce si klade za cíl JS z tohoto vzorce vynechat a jeho roli nahradit knihovnou HTMX.

Front-end frameworky a knihovny vznikly v první řadě pro zjednodušení vývoje a udržování aplikace nebo pro úpravu již existujících technologií a rozšíření jejich funkcionality. Podkladem tohoto tvrzení může být následující text odůvodňující využití front-endové knihovny React:

React byl vyvinut za účelem poskytnutí možnosti dynamické aktualizace obsahu zobrazované stránky. Od doby, kdy se stránky využívaly k odesílání formulářů nebo k prohlížení statických stránek, se uživatelské možnosti webových aplikací posunuly mnohem dále a s tím rostla i poptávka po webech s dynamickými funkcemi. Zásadním požadavkem byla možnost rychlé změny části obsahu bez nutnosti opětovného načítání celého obsahu stránky. (Kumar, 2024)

K dispozici je dnes nepřeberný počet jazyků, knihoven, frameworků, vývojových prostředí, asistentů a mnoho dalších nástrojů, které si kladou za cíl ulehčení práce při vývoji. Pro práci s tak rozsáhlými frameworky, jako jsou například React nebo Angular, je však potřeba dobrá znalostní báze těchto systémů. Ty často využívají velké množství vlastních konvencí vývoje a programátor je tak svým způsobem limitován na nastavenou funkcionalitu frameworku.

Na druhé straně spektra se nachází knihovna HTMX, která svou jednoduchostí v porovnání s již zmíněnými giganty působí jako velice základní nástroj pro vývoj webových aplikací. Svým způsobem se skutečně jedná o knihovnu jednodušší, ne vždy ale komplexnější musí nutně znamenat lepší.

Následující kapitoly slouží jako úvod ke zmiňovaným technologiím a následný podklad vývoje aplikace.

1.1 Hypermedia

Hypermedia jsou taková media, která se dokáží nelineárně větvit do jejich jiných částí. K tomu je možné využít odkazy nebo tlačítka na stránce. To je rozdíl mezi medii, například tištěným časopisem, a hypermedii.

Podmnožinou hypermedií jsou hypertexty. Tím je například HTML, textový zápis struktury webu, který s dalšími technologiemi, jako jsou například HTTP protokol, hypermedia servery a hypermedia klienty (prohlížeče), tvoří hypermedia systémy. (Gross et al., 2023)

Jak dále Gross a kol. (2023) uvádí, hypermedia nejsou novou technologii. První webová stránka byla vydaná v roce 1990 a od té doby prošly veškeré technologie hypermedia systémů velkou proměnou. Jejich využití se ale od hypermedia systémů lehce odklonilo a veliký důraz je kladen na aplikace poháněné JSON komunikací.

1.1.1 HTML

Základní stavební kámen webových stránek. Jak uvádí Castro a Hyslop (2022), HyperText Markup Language je značkovací jazyk používaný od posledního desetiletí 20. století. Poskytuje elementy pro definici jednotlivých částí stránky a jejich významu. Postupným vývojem se jazyk zdokonaloval a funkcionalita rozšiřovala.

Nejnovější verzí tohoto jazyka je HTML5, která přináší nové, modernější funkce, avšak si stále klade za cíl udržet zpětnou kompatibilitu.

Jazyk HTML využívá k zápisu tři hlavní prvky. Elementy jsou značky, kterými označujeme části textu. Některé stránce přidávají význam, jiné ale naopak textu žádný konkrétní význam nepřidávají. Elementy se ohraničují závorkami „<“ a „>“.

Dalšími prvky jsou atributy a jejich hodnoty. Tyto prvky poskytují dodatečné informace o elementu, jeho obsahu a další. (Gasston a Baše, 2016)

Základ HTML dokumentu se skládá z několika částí, velmi stručně se jedná o definici typu dokumentu, hlavičku a tělo dokumentu. Pro účely této práce není nezbytně důležitá definice těchto částí, jelikož je vyvíjeno pouze tělo aplikace, které bude vkládáno do existujících webů. Pro prezentační účely je v práci obsažena základní kostra stránky.

Pro navigaci hypermedii jsou v základu poskytnuty 2 elementy: button a anchor. Tlačítka slouží pro ovládání formulářů. Anchor elementy pak slouží k přesměrování prohlížené stránky na odkaz specifikovaný v atributu „href“ tohoto elementu.

1.1.2 HTMX

HTMX je JavaScriptová knihovna, která umožňuje programátorům vytvářet aplikace poháněné hypermedii (hypermedia-driven applications). Tento koncept využívá pro komunikaci REST server s použitím hypermedií jako komunikačního prostředku. Namísto JSON dat se tedy ze strany serveru posílají hypermedia, která se pomocí knihovny na stránku vkládají. V případě takovéto komunikace tedy není třeba žádný další systém na straně uživatele, který by musel JSON data zpracovávat a překládat je do změn ve struktuře hypertextu.

Knihovna se nesnaží upravit vývoj hypermedia aplikací využitím dalšího samostatného systému, který by bylo nutné používat namísto klasického HTML. Pouze se snaží opravit některé nedostatky HTML jazyka a učinit ho tak flexibilnějším a modernějším. Mezi tato vylepšení patří následující:

HTMX umožňuje odesílání požadavků na server jakémukoliv HTML elementu, rozšiřuje tak výčet ovládacích prvků poskytnutý jazykem HTML ze dvou na mnohem vyšší číslo. Odesílání dotazů na server není podmíněno jen kliknutím na tlačítko nebo odkaz. HTMX zároveň dokáže měnit obsah jen na části stránky, nikoliv pouze obsah celé stránky. (Gross et al., 2023)

Jak dále Gross a kol. uvádí, HTMX si neklade za cíl být nástupcem JS systémů a naznačují, že ne všechny aplikace mohou být postaveny čistě pomocí využití

hypermedií. Například uvádějí, že by nebylo výhodné využívat HTMX pro aplikace, které provádějí značné množství výpočtů nebo dynamických aktualizací při používání aplikace. Jmenovitě uvádějí například webovou aplikaci tabulkového procesoru.

1.2 CSS

Po vytvoření rozložení stránky přichází na řadu grafická stylizace webu. Tu definujeme tzv. kaskádovými styly – CSS. Jazyk CSS je vydán pár let po HTML a je od té doby standardním nástrojem pro stylizování grafické podoby webu. Od jeho vydání se funkcionalita jazyka neustále vyvíjí.

Změny v modulech následující vydání verze CSS 2.1 byly již tak rozsáhlé, že bylo nutné doporučení vyvíjet a vydávat v samostatných modulech. Namísto verzování samotné CSS specifikace tedy nyní W3C periodicky vydává tzv. obrazy nejnovějšího stabilního stavu CSS a modulů. (Mozilla Foundation, 2024a)

Nové verze CSS již tedy nemají své vlastní číslo, jsou ale verzovány jednotlivé moduly jazyka. V textu je dále uváděna verze pouze ve formě snapshotů.

1.3 PHP

PHP je známý skriptovací jazyk používaný zejména pro vývoj webových aplikací. Jedná se o jazyk, který je běžně spouštěný na serveru a uživateli poskytuje HTML výstup. (The PHP Group, 2024)

Jedná se o jednoduchý a výkonný jazyk pro vytváření HTML stránek. Běží na všech významných operačních systémech a je podporován všemi běžnými webovými servery. První verze jazyka se objevila už v roce 1994, avšak moderní verze se od této významně liší.

V následujících kapitolách jsou popsány aspekty jazyka využívané při vývoji aplikace.

1.3.1 Rozhraní

Rozhraní umožňují tvorbu kódu, který specifikuje metody, jež musí třída implementovat. Zároveň ale tyto metody rozhraní nedefinují, jakým způsobem jsou implementovány. Veškeré metody rozhraní musí být veřejné. (The PHP Group, 2024)

V praxi rozhraní umožňují vytvářet objekty různých tříd, které mohou být užity stejným způsobem, jelikož implementují stejné rozhraní. Příkladem mohou být třídy elementů, jež všechny implementují rozhraní ShowableInterface.

1.3.2 Abstraktní třídy

Abstrakty jsou rozšiřitelnými třídami, které ke klasické funkcionalitě PHP tříd mohou specifikovat abstraktní funkce a obsahovat neimplementované funkce rozhraní. Specifikum abstraktních tříd tkví v tom, že nelze vytvořit její instanci, musí být vždy rozšířeny jinou třídou. (Mann, 2023)

Tyto třídy tedy poskytují funkcionalitu rozhraní, kdy třída rozšiřující abstraktní třídu musí implementovat abstraktní funkce. Ty může definovat samotná abstraktní třída nebo rozhraní. Zároveň je ale možné v této třídě předdefinovat funkce, které bude moct rozšiřující třída využít.

1.3.3 Traity

Traity reflektují snahu o obejití některých limitací jednonásobné dědičnosti, kdy programátor může využívat metody definované traitem napříč různými na sobě nezávislými třídami. (The PHP Group, 2024)

Tyto objekty jsou podobné třídám, jejich úlohou je ale pouze sdružování určité funkcionality do samostatných objektů, kterou mohou třídy využívat. Na rozdíl od tříd ale nelze vytvořit instanci traitu. (The PHP Group, 2024)

V případě programované aplikace jsou traity využívány např. pro obsluhu atributů elementů (viz k. 5.2.1).

1.3.4 Atributy

Třída Attribute přidaná v PHP verze 8 poskytuje možnost přidání strukturovaných metadat, která mohou být za běhu čtena. Mohou tak být využita jako konfigurační jazyk začleněný přímo do kódu (The PHP Group, 2024). Tyto atributy jsou svojí funkcí podobné například anotacím v jazyce Java.

V aplikaci jsou využity atributy Override pro implementované metody rozhraní a abstraktní funkce. Výhoda využití tohoto atributu tkví v tom, že když se přidá ke třídě, která není implementovatelná z žádného rozhraní, PHP vyhodí chybu. Dají se tak rychle trasovat chyby v implementacích.

1.4 JavaScript

Nejrozšířenějším jazykem pro programování front-endových částí aplikací je jazyk JavaScript. Ten, stejně jako většina základních webových nástrojů, vznikl na konci 20. let.

Dnes je jazyk JavaScript standardizován jazykovou specifikací ECMAScipt, která je definovaná standardem ECMA-262. Velký posun jazyk zaznamenal v roce 2015, kdy byla přijata podoba specifikace ECMAScript 6, jež byla aktivně vyvíjena od roku 2009. Samotná snaha o zásadní vylepšení však pramenila z aktivního výzkumu probíhajícího už od vydání třetí verze ECMAScriptu v roce 1999. Šestá edice je tak označována za výsledek patnáctiletého výzkumu.

„ECMAScript je specifikace skriptovacího jazyka standardizovaná organizací ECMA International. Byla vytvořena za účelem standardizace JavaScriptu se snahou umožnit nezávislé a kompatibilní implementace.“ (Shute, 2019) (přeloženo autorem)[1]

Jazyk umožňuje pokročilou interakci s grafickým prostředím uživatele za běhu, tedy jak se značkovacím jazykem HTML tak i kaskádovými styly. S jeho pomocí je také možné používat asynchronní volání na endpointy, využívání tzv. listenerů, které po provedení určitých akcí spouští další části skriptu, a mnoho dalšího. Jednou z mnoha výhod je také přímý přístup a možná manipulace s DOM (Document Object Model).

[1] ECMAScript is a scripting language specification standardized by ECMA International. It was created to standardize JavaScript in an attempt to allow for independent and compatible implementations. (Shute, 2019)

2 Vývojové principy

Moderní aplikace využívají různé postupy pro jejich vývoj. Vytvoření takové aplikace, která bude schopná postupného a bezproblémového růstu, je velmi náročné. Nastavují se proto pravidla, která jasně vymezují odpovědnost objektů v kódu. (Scott, 2016)

I při vývoji kódu této práci byla vyvinuta snaha aplikovat vybrané principy vývoje.

Díky využití knihovny HTMX lze vytvářet aplikaci, která se zobrazuje na jedné stránce. Tento princip se označuje za single page architekturu. Díky této knihovně HTMX je také možné využít princip RESTful web API pro komunikaci mezi serverem a knihovnou, respektive prohlížečem.

2.1 Single-page Aplikace

Single-page aplikace je specifická tím, že načítá jediný webový dokument a poté jeho obsah pouze aktualizuje. Tyto aktualizace běžně vykonávají JS API (Mozilla Foundation, 2024b). V případě vyvíjené aplikace JS zastupuje knihovna HTMX.

Výhoda těchto aplikací tkví v menší zátěži na server a internetovou infrastrukturu, přičemž uživatelské prostředí může být mnohem dynamičtější. Na druhou stranu tyto stránky mohou být náročné na údržbu a nelze pro ně vyvinout tak dobré SEO, jako na ostatních aplikacích. (Mozilla Foundation, 2024b)

Vyvinutá aplikace může být single-page díky layoutům, které se zobrazují podle stavu příslušných proměnných ve třídě SessionData. Zobrazení stránky tak může probíhat vždy na výchozí adrese serveru.

2.2 MVC Architektura

Model-View-Controller je architektura používaná pro rozdělení aplikace do tří základních skupin. Views jsou objekty uživatelského rozhraní. Jsou to objekty, které řídí zobrazování dat uživateli a se kterými uživatel interaguje. Modely jsou objekty poskytující data objektům View. Controllery pak obsluhují uživatelské akce a řídí změny ostatních objektů. (Freeman a Robson, 2020)

Aplikace tuto architekturu implementuje, i když ne zcela přesně. FE uchovává View objekty, Controller a Model se pak nacházejí v backend (dále jen BE) části.

Controller v aplikaci slouží jako router, který zpracovává HTTP požadavky uživatele. Odpovědnost zpracování těchto požadavků byla uložena samotným objektům View. Změny v objektech Model se tak provádí z objektů View, ne Controller.

Model objekt, v případě aplikace SessionData, pak uchovává veškerá data a stavy. Podle těchto dat se řídí zobrazování objektů View.

2.3 RESTful Web API

REST je flexibilní rozhraní pro integraci aplikací a propojování komponent v microservice architekturách. Microservice architektura pak zprostředkovává samostatně fungující komponenty nebo služby. Znamená to tedy, že klientská aplikace pomocí tohoto rozhraní může přistupovat k jiné webové aplikaci nebo službě. (IBM, 2024)

Stav (state) požadovaného zdroje v daný moment se označuje jako jeho reprezentace (representation). Tato data mohou být klientské aplikaci pomocí HTTP dotazu poskytnuta v jakémkoliv formátu, nejčastěji pak ve formátu JSON (IBM, 2024). V případě aplikace spojené s touto prací se jedná o data ve formátu HTML, jelikož se jedná o hypermedia-driven aplikaci.

3 Analýza požadavků na aplikaci

Na úvod je třeba vytyčit několik požadavků na programovanou aplikaci, podle kterých se bude řídit následný vývoj. Tyto požadavky vyplývají z konzultací s vedoucím a konzultantem bakalářské práce a z průzkumu košíků největších českých a zahraničních on-line obchodů.

Při průzkumu byl kladen důraz na typy a obsah formulářů, jejich pořadí a funkce.

3.1 Obecné

Aplikace poskytne funkcionalitu virtuální pokladny pro potřeby e-shopů. Tuto aplikaci tvoří front-end část pro prezentaci a vstup dat s back-end strukturou pro manipulaci a jejich zpracování.

Koncept webové aplikace je vyvíjen se snahou vytvořit jednoduché moderní řešení, které nabízí přizpůsobitelnost a flexibilnost v nasazení pro různé typy internetových obchodů.

Apliace by měla být modulární, jednoduše upravitelná jak vizuálně tak funkčně. Jednoduchá by měla být změna pořadí částí formuláře, úprava kódu HTML elementů, které budou všechny definovány ve vlastním souboru, ošetření interakcí uživatele se stránkou a další. Z toho vyplývá, že bude aplikace univerzálně využitelná pro zákazníky s různými potřebami a požadavky.

Mělo by být proveditelné napojení částí formuláře na analytické nástroje sledující aktivitu v rámci aplikace. Pro potřeby této práce postačuje pouze příprava na integraci, jelikož je pro sledování chování zákazníků na stránce možné využít různé techniky a nástroje.

Výsledkem by mělo být ošetření manipulace s částmi formuláře, kdy se po provedení požadované akce, kupříkladu stisknutí tlačítka, zadání textu do řádku, zobrazení části formuláře a další, spustí funkce, která bude obsluhovat konečné sledovací technologie.

Dalším požadavkem je schopnost uchování dat formuláře. Objekt zajišťující uchovávání dat je zároveň jejich jediným zdrojem. Aplikace by měla uchovávat data, která uživatel zaznamená ve formuláři a data o stavu prvků grafického rozhraní.

Zásadní funkcí aplikace je také reakce na uživatelský vstup na straně FE. Ovládací prvky by měly být schopny měnit grafickou podobu stránky. Například po zaškrtnutí zaškrtávacího pole by aplikace měla být schopná vykreslit další část formuláře a po zrušení zaškrtnutí tuto část znovu skrýt.

3.2 Formuláře

Každý formulář by měl mít v základu nadpis a odstavec pro popis formuláře. Samotný obsah by měl být jednoduše definovatelný a upravitelný.

Základní verze aplikace by měla poskytovat následující formuláře webové pokladny:

- volba dodací metody
- základní kontaktní údaje
- způsob platby
- souhrn zadaných dat

Zároveň aplikace bude obsahovat stránku, která se zobrazí po vyplnění formuláře s možností stažení zadaných dat a přesměrování na novou pokladnu.

3.2.1 Dodací metody

Formulář dodacích metod by měl nabídnout uživateli možnost výběru způsobu dodání objednávky. Při výběru doručení na adresu je třeba zobrazit dodatečný formulář s poli pro zadání adresy. Výběr dodání na pobočku zobrazí formulář s polem pro určení adresy pobočky. Při zaškrtnutí vyzvednutí na prodejně se zobrazí adresa prodejny.

3.2.2 Kontaktní údaje

V sekci základních kontaktních údajů je potřeba získat kontaktní a fakturační údaje o zákazníkovi. Mělo by být možné vyplnit pouze základní údaje pro nákup, zároveň je ale třeba mít možnost vyplnit informace o podnikatelských subjektech. Je žádoucí, aby byl zákazník konfrontován s co nejmenším počtem elementů pro zachování přehlednosti. Je tak možné některá pole skrýt za checkboxy nebo do rozbalovacích menu.

3.2.3 Způsob platby

Formulář zaznamená zvolený způsob platby, který by dále bylo možné využít např. při zobrazení odkazu k platební bráně nebo platebních údajů či pokynů. Tato funkcionalita však přesahuje cíle této práce a proto není implementována.

3.3 Kompatibilita zařízení

Při tvorbě aplikace je nutné dbát na to, aby bylo možné využít jednu aplikaci s minimálními úpravami na mnoha různých zařízeních. Je tedy třeba na webovou stránku vložit aplikaci tak, aby se v různých kontextech chovala vždy ideálně naprosto stejně. Při vývoji je tedy třeba počítat s tím, že se aplikace bude zobrazovat na obrazovkách počítačů, mobilních zařízeních a dalších různých zařízeních s různými tvary obrazovek a typy vstupů uživatele.

Pro řešení tohoto problému existuje více přístupů. Jak tedy nejlépe dosáhnout toho, aby se stránky zobrazovaly správně na různých zařízeních a byly dobře ovladatelné?

3.3.1 Oddělené verze webu

Při řešení těchto problémů je možné se setkat s koncepcí tzv. „m tečka“ webů. To jsou weby, které používají místo klasické adresy *www.web.xy* tvar *m.web.xy*. Jde o dvě různé stránky s identickým či upraveným obsahem, kdy klasická stránka je programovaná pro zařízení desktopová a stránka s doménou třetího řádu ve tvaru „m“ je určená pro zařízení mobilní. (Michálek, 2017)

Příkladem takových webů může být sociální síť Facebook, kdy existuje desktopová verze *www.facebook.com* a verze pro mobily *m.facebook.com*. Podobu této stránky zobrazuje obrázek 3.1.

Obr. 3.1: Mobilní verze Facebooku

Michálek dále hodnotí aspekty využití „m" webů. Mezi největší výhody řadí rychlé tempo vývoje, kdy vývoj není zatížen zdlouhavými úpravami již existujícího webu. Obecně ale hodnotí tento přístup jako dlouhodobě neudržitelný.

Rozbor mobilní stránky Facebook

Náznak tohoto postoje můžeme pozorovat i u zmíněné stránky m.facebook.com. Vývojáři mobilní stránce v porovnání s hlavní stránkou nevěnují přílišnou pozornost, což se promítá do designu a funkčnosti stránky. Ta je na dnešní poměry poměrně zastaralá. Uživatelský komfort na této stránce je také nízký.

Řezáč ve své knize rozebírá dostupnost webu na mobilních zařízeních a vytyčuje několik pouček. Zde jsou uvedeny jejich zkrácené verze:

- Správné uspořádání
- Rychlé načítání
- Výrazné a dostatečně velké aktivní prvky
- Nezávislost na stavech po najetí myši

Mobilní stránka Facebook zobrazená na velikosti zařízení IPhone 11 Pro dle těchto pouček nevyužívá dostatečně velké ovládací prvky. Odkazy jsou malé a oddělené jen velmi malými mezerami. Uživatelské rozhraní tak není dobře přístupné a uživatelé mohou mít problém s ovládáním aplikace.

Upozadění stránky naznačuje i fakt, že se Facebook na mnoha místech snaží uživatele přesměrovat do své mobilní aplikace namísto pokračování využívání mobilního webu. Dá se tedy očekávat, že Facebook udržuje tuto stránku zejména pro zařízení, na kterých není možné pohodlně ovládat desktopovou verzi a zároveň není možné nainstalovat mobilní aplikaci.

3.3.2 Responzivní design

Responzivní design je metodika pro vývoj webu, kdy se pracuje s koncepcí zlomových bodů. Tyto body jsou ohraničení, na kterých dochází ke změnám ve vizuálním obsahu stránky. Hraniční body definují dotazy na media. (Gasston a Baše, 2015)

Toto řešení poskytuje možnost tvorby jednoho webu s tím, že se obsah stránky upravuje podle naprogramovaných media query. Je tak možné dosáhnout zásadních změn, jelikož každý dotaz na media může mít vlastní logiku a je možné dokonce vytvářet pokročilé podmínky pro aplikování stylů. Zásadním hráčem v responzibilitě budou proto jistě kaskádové styly. Jak ale uvádí Gasston, je možné využít dotazy na medium i v jazyce JavaScript. Psaní responzivního webu se tedy nesoustřeďuje pouze na CSS.

3.3.3 Adaptivní design

Po boku responzivního designu existuje další metodika vývoje zvaná adaptivní. Ta také implementuje funkcionalitu media queries, tedy zlomových bodů závislých na velikosti stránky. Zásadním rozdílem ale je, že se stránka při změně velikosti přepíná mezi designy s pevnou šířkou. Responzivní design je na druhé straně schopen obsah dynamicky přizpůsobovat pohyblivé šířce. (Gasston a Baše, 2015)

V praxi je tedy nutné při využití adaptivního designu programovat stránku pro každou velikost obrazovky zvlášť.

3.3.4 Zhodnocení

Tvorba oddělených webů je funkční možnost, ne každý e-shop ale využívá oddělené stránky pro různá zařízení a dlouhodobě je pracné stránku udržovat aktuální. Převážně proto není zmíněná metodika v tomto případě efektivním řešením.

Adaptivní metodika také nebude ideálním řešením vzhledem k pevné šířce obsahu, je žádoucí obsloužit co nejširší skupinu zařízení.

Z výše uvedených možností je tedy nejlépe vyhovující metodika responzivního designu.

3.4 Front-end aplikace

Aplikace bude pro aktualizaci obsahu využívat knihovnu HTMX, která za pomoci vlastních HTML atributů poskytuje funkčnost asynchronous JavaScript and XML (dále jen AJAX) dotazů přímo v HTML kódu. Výhodou je možnost redukce JS kódu v celé aplikaci, v případě této práce je snaha dokonce zcela vyřadit technologii JavaScript. Úspornost knihovny HTMX, která celkem zabírá kolem 16 kB místa na disku. Veškeré změny ve struktuře formuláře, datové přenosy a uživatelské akce tedy budou ovládány HTMX.

Prostředí pokladny bude tvořit několik sekcí popsaných v požadavcích na aplikaci. Každou sekci reprezentuje samostatný HTML element, ve kterém jsou obsaženy další prvky. Sekce bude označena odpovídajícími atributy class a id.

Vložení formuláře na web bude zajišťovat controller, který pomocí souborů s rozvržením stránky, tzv. layoutů, zobrazí úvodní stránku aplikace. Následně se bude vložená stránka pouze upravovat, nebude se načítat znovu. Pozici celého formuláře pak bude definovat jeden obalový element, do kterého se vloží celá stránka aplikace sestavená backendem.

Jednotlivé formuláře, stejně jako jejich vstupy, budou definovány ve vlastních šablonách. Do šablony formuláře pak backend vloží příslušné definované vstupy. To umožňuje efektivně upravovat celou skupinu vstupů bez potřeby procházet celou HTML strukturu formuláře.

3.5 Back-end aplikace

Backend bude využívat technologii PHP serveru. Hlavní úlohou BE je poskytování přístupu na endpointy, které zpracovávají HTMX dotazy nebo poskytují statický obsah bez nutnosti zpracování dat.

Na serveru bude k dispozici router, který bude přesměrovávat příchozí dotazy na samotné objekty formulářů. Každý endpoint pak bude mít jasně definovanou funkci a jeho logika bude definována v samostatném souboru.

Pro potřeby vývoje práce nebude BE napojen na databázi, ale bude data uchovávat v superglobální proměnné SESSION. Důvodem jsou četné metody implementací aplikace, kdy komunikace s databází probíhá různými způsoby.

4 Vývoj webové aplikace

Vzhledem ke snaze vytvořit aplikaci s minimem technologií, resp. závislostí, bylo při programování omezeno využití knihoven třetích stran. Proto bylo při vývoji vyvinut vlastní základní framework. Tato abstraktní vrstva má za úkol z objektů vytvořit ucelenou a prezentovatelnou webovou aplikaci.

Aplikace je rozdělená na backend a frontend. FE uchovává objekty, které reprezentují grafické prvky a ovládají jejich vykreslování. Dále se zde nachází layouty, které definují podobu stránek. Sekce BE obsluhuje HTTP dotazy a poskytuje objekty pro práci s frontendovou částí aplikace. FE je tak základně rozdělen podle MVC architektury, která je popsaná v kapitole 2.2.

Veškeré soubory aplikace jsou dostupné na přenositelném mediu přiloženém k této práci.

Následující části této kapitoly čtenáři popisují implementaci knihovny HTMX, přibližují strukturu a ideu vývoje aplikace. Podrobnější rozbor tříd a objektů poskytuje kapitola 5.

4.1 Prostředí a technologie

Pro vývoj byl využit operační systém Linux (Ubuntu 22.04.4 LTS) a vývojové prostředí PhpStorm. Některé popsané postupy a nástroje jsou pro tento systém specifické, je možné ale docílit stejné nebo podobné funkcionality na i na ostatních systémech. Aplikace je vytvořena pomocí základních verzí HTML 5 a CSS Snapshot 2023. BE je připravený pro PHP verze 8.3 a pro spouštění serveru je využíván PHP interpreter pro příkazový řádek z balíku PHP-cli.

Pro zjednodušení opakované činnosti spouštění PHP serveru je možné využít GNU Make. Nástroj slouží pro generování spustitelných a nezdrojových částí programu z jeho zdrojových souborů. Samotný proces využívá direktivy ze souboru zvaného makefile. (Free Software Foundation, 2023)

V případě této aplikace je pouze vytvořen příkaz v makefile, který v terminálu změní aktuální adresář na kořenovou složku aplikace a následně ji pomocí PHP interpretera spustí. Výslednou výhodou je skutečnost, že se nemusí dokola vypisovat PHP skript se všemi parametry na správné cestě, pouze se spustí Make příkaz v adresáři se souborem makefile.

4.2 Implementace HTMX

Tato kapitola popisuje implementaci a využití knihovny HTMX v kódu aplikace. Popis knihovny vychází z dokumentace, kterou prezentuje Gross a kol. (2023).

Implementace knihovny do kódu HTML je jednoduchá. Nejprve je nutné knihovnu nainstalovat. Toho je možné dosáhnout několika způsoby. Všechny mají ale jeden krok společný, a to vložení elementu `<script>` s atributem `src` odkazující na zdroj knihovny do elementu <head>.

Knihovnu je možné poskytnout přímo stažením souboru knihovny a jeho vložením do aplikace, nainstalovat ji pomocí nástroje `npm`, nebo do skriptu vložit odkaz na knihovnu uloženou v content delivery network (dále jen CDN), kdy se aplikace získává z on-line úložiště.

```
<head>
<meta charset="utf-8">
<meta name="viewport" content="…">
<title>Checkout page</title>
<link rel="stylesheet" href="css/style.css">
<script src="https://unpkg.com/htmx.org@1.9.11" …>
</script>
…
```

Blok kódu 4.1: Skript poskytující HTMX v hlavičce

Vyvíjená aplikace k poskytnutí knihovny využívá CDN, v produkčním prostředí je však doporučeno využít metodu jinou. Po provedení instalace je možné ihned začít knihovnu využívat v aplikaci.

Instalaci knihovny do aplikace v obalovém kódu HTML prezentuje blok 4.1, který je uložen v souboru `/frontend/index.phtml`.

Využití samotné knihovny probíhá pomocí atributů s prefixem „hx-“, které se vkládají do elementů, které mají být knihovnou upraveny.

4.2.1 Odesílání dotazu

Jak uvádí Gross a kol. (2023), základní funkce HTMX je umožnění jakémukoliv elementu odesílat http dotazy. Tuto funkci specifikují HTML atributy `hx-get`, `hx-post` a další pro ostatní HTTP metody. Hodnotou tohoto atributu je cesta, na kterou má být dotaz odeslán. HTMX v odpovědi ze serveru očekává hypertextový kód, kterým pak ve výchozím nastavení nahradí vnitřní kód elementu. Zde využitý atribut `hx-swap` popisuje kapitola 4.2.2.

Příkladem využití může být formulář platebních metod, který odesílá data na prefix komponenty uvedený v bloku 4.2.

```
<form id="platebni-metoda"
      hx-post="/platba"
      hx-swap="outerHTML">
```

Blok kódu 4.2: Využití atributu hx-post

Samotné odeslání dotazu spouští události uvedené v atributu `hx-trigger` Odeslání dotazu mohou spouštět také události HTML elementů, tedy událost `change` pro `input` elementy, událost `submit` pro element `form` a pro ostatní elementy událost `click`.

Další možností spuštění odeslání dotazu je využití HTTP hlaviček. Pomocí vrácení hlavičky „`HX-Trigger:`“ ze serveru lze na FE spustit event, na který reagují atributy `hx-trigger` HTML elementů. Tuto funkci využívá např. komponenta souhrnu zobrazená v bloku 4.3.

V tomto případě komponenta reaguje na serverem vrácenou odpověď s hlavičkou „HX-Trigger: form-changed“ a následně odesílá dotaz na cestu „/summary“. Hypertext odpovědi pak nahrazuje celý element.

```
<form id="summary"
hx-get="/summary"
hx-trigger="form-changed from:body"
hx-swap="outerHTML"
hx-post="/summary" >
```

Blok kódu 4.3: Využití atributu hx-trigger

4.2.2 Cíle HTTP odpovědi

HTMX v základu odpovědi zpracovává tak, že se jejich obsah vkládá namísto stávajícího obsahu elementu odesílajícího dotaz. Toto chování lze změnit atributem hx-swap. Aplikace využívá hodnoty „innerHTML“ (výchozí) a „outerHTML“, které nastavují změnu vnitřního kódu nebo kódu včetně elementu. Knihovna ale poskytuje i hodnoty pro vkládání odpovědi za nebo před element, smazání elementu nezávisle na odpovědi a nakonec i hodnotu „none“ pro ignorování odpovědi.

Hypertext HTTP odpovědi je možné nasměrovat i na jiný element. To znamená, že element vyvolávající dotaz zůstane nezměněný, ale změní se jiný cílený element. Nastavení probíhá pomocí atributu hx-target. Ten je využit např. v bloku 4.4, který zobrazuje textové pole pro zadání IČ.

```
      <input type="checkbox"
             name="ic-checkbox1"
             value="ic-sale"
             id="ic-checkbox"
             hx-post="/contact/ic-select"
             hx-trigger="click"
             hx-target="#kontakt-component"
             hx-swap="outerHTML">
```

Blok kódu 4.4: Využití atributu hx-target

Odpověď odeslaného dotazu v tomto případě nahradí celou komponentu s id `kontakt-component`. Důvodem je to, že se po spuštění akce `ic-select` vrací celé tělo komponenty, jelikož vykresluje s přidaným polem pro zadání IČ.

4.3 Struktura front-end části

Při vývoji je třeba logicky uspořádat soubory aplikace. Navigace v uspořádané adresářové struktuře je předpokladem pro bezproblémový vývoj. Objekty spojené s vykreslováním grafické části jsou proto sdruženy do adresáře `/frontend`. Celkovou strukturu FE popisuje následující stromový diagram:

```
frontend/
├── api/
│   ├── GetInterface.php
│   ├── PostInterface.php
│   └── RoutableInterface.php
├── components/
│   ├── abstract/
│   │   ├── ComponentAbstract.php
│   │   ├── DivComponentAvstract.php
│   │   └── FormComponentAbstract.php
│   ├── DokoncenoComponent.php
│   ├── DopravaComponent.php
│   ├── KontaktComponent.php
│   ├── PlatbaComponent.php
│   └── SouhrnComponent.php
├── css/
│   └── style.css
├── elements/
│   ├── api/
│   │   ├── ButtonInterface.php
│   │   ├── CheckInterface.php
│   │   ├── InputInterface.php
│   │   ├── RequiredInterface.php
│   │   ├── ShowableInterface.php
│   │   └── ValidationInterface.php
│   ├── traits/
│   │   ├── AttributesTrait.php
│   │   ├── ErrorTrait.php
│   │   └── HttpHeadersTrait.php
│   ├── ButtonElement.php
│   ├── CheckboxElement.php
│   ├── InputElement.php
│   ├── LinkElement.php
│   ├── ParagraphElement.php
│   └── RadioElement.php
└── layouts/
    ├── api/
    │   └── LayoutInterface.php
    ├── IndexLayout.php
    └── SuccessLayout.php
```

Obr. 4.1: FE adresářová struktura

Velkou část objektů v tomto adresáři tvoří třídy, které reprezentují grafické elementy, a jejich traity nebo rozhraní. Dále jsou zde uloženy třídy komponent, které pomocí elementů sestavují větší funkční celky, jejich abstrakty a rozhraní. V neposlední řadě jsou zde také uloženy kaskádové styly a layouty aplikace.

V následující části jsou popsány skupiny objektů obsažené v adresáři a jejich stručná funkcionalita.

4.3.1 Rozhraní

Pro frontend jsou vytyčena základní routovací rozhraní, která slouží pro zpřístupnění třídy HTTP dotazům. Jedná se o rozhraní RoutableInterface, PostInterface a GetInterface. Uložena jsou v adresáři `/frontend/api/`. Implementace těchto rozhraní třídami rozhoduje o tom, jestli je možné tyto objekty zavolat pomocí HTTP dotazů a jaké dotazové metody jsou třídy schopny zpracovat.

Objekty podle implementovaných rozhraní zpracovává backend třída Controller umístěná v adresáři `/backend/controllers`, přičemž rozhraní jsou zde významná při dynamickém směrování objektů. Pro představu je možné popsat následující ukázku ze jmenované třídy:

```
switch ($requestInterface){
  case PostInterface::class:{
    $response = $routable->post($_POST, $actionPath);
    break;
  }
  case GetInterface::class:{
    $response = $routable->show($_GET, $actionPath);
    break;
  }
}
```

Blok kódu 4.5: Využití rozhraní při směrování HTTP dotazů

V tomto kusu kódu třída Controller zpracovává dotaz podle rozhraní metody dotazu použitím bloku switch-case. Na základě toho, jakou metodou je dotaz proveden, resp. jaké rozhraní metody je stanovené v proměnné `$requestItnerface`, spustí

Controller na routovatelné třídě funkci post nebo show. Routování popisuje blíže kapitola 4.5.2.

Ve FE části se nachází další rozhraní související se třídami elementů, ta jsou popsána v kapitole 4.3.2. Výčet veškerých rozhraní aplikace spolu s jejich popisem uvádí kapitola 5.1.

4.3.2 Elementy

Nejmenším zobrazitelným prvkem v aplikaci je HTML element. Do PHP byly proto tyto elementy převedeny do podoby tříd a jsou uloženy v samostatné složce `/frontend/elements/`.

Příkladem může být element div zobrazující textové pole s jeho popisem. To je na stránce zobrazeno pomocí následujícího kódu:

```
<div class="simple-input prijmeni">
  <label for="prijmeni">Prijmeni</label>
  <input name="prijmeni" id="prijmeni" value="">
</div>
```

Blok kódu 4.6: Element div s textovým polem

Tento HTML kód se převádí do podoby třídy v PHP. Funkce, která vykresluje HTML reprezentaci kusu kódu uvedeného v bloku 4.6, vypadá následovně:

```
#[Override] function show(): string
  {
  …
  return <<< EOL
    <div class="simple-input $this->id">
      <label for="{$inputIdentifier}">{$this->label}</label>
      <input {$htmlAttributes}>
      {$error}
    </div>
  EOL;
}
```

Blok kódu 4.7: Třída reprezentující div element

Elementy mají vlastní rozhraní uložená ve složce `/elements/api/`, která třídy implementují. Tato rozhraní se zpravidla liší mezi různými elementy, obzvláště mezi těmi pro získávání dat. To je způsobeno tím, že se každý vstupní element HTML používá trochu jiným způsobem.

Například rozhraní InputInterface reprezentující HTML element `<input>` pracuje pouze s hodnotou pole.

```
interface InputInterface extends ShowableInterface
{
  function setValue($value);
  function getValue();
}
```

Blok kódu 4.8: Rozhraní InputInterface

Zato rozhraní CheckInterface zastupující `<input type="checkbox">` musí umět zpracovat stav zaškrtnutí pole, vrátit hodnotu pole a název skupiny.

```
interface CheckInterface extends ShowableInterface
{
  public function setChecked(bool $checked);
  public function getChecked(): bool;
  public function getGroupName(): string;
  public function getValue(): string;
}
```

Blok kódu 4.9: Rozhraní CheckInterface

Všechny elementy ale rozšiřují základní rozhraní ShowableInterface. To třídě nařizuje deklaraci funkce, která vrací HTML reprezentaci obsahu elementu, třída se tak stává „zobrazitelná“.

4.3.3 Komponenty

Pomocí elementů můžeme stavět větší celky, které se sdružují do jednoho rodičovského elementu. Tyto sekce se v aplikaci nazývají „komponenty“ a nachází se ve složce `/frontend/components`. V případě vyvíjené aplikace se jedná

nejčastěji o formulářové komponenty, k dispozici je ale i třída DivComponent. Třídy komponent tak představují blok obsahu, nejčastěji tedy samotný formulář.

Obsah komponenty má programátor možnost definovat pomocí funkce `elements()`, kterou poskytuje libovolný abstrakt komponenty. Výstupem této funkce je pole, jehož obsah pak rozšířený abstrakt zpracovává. Příklad lze uvést na ukázce části funkce `elements()` třídy KontaktComponent v bloku 4.10. Některé funkce byly záměrně zkráceny kvůli přehlednosti.

```
protected function elements(): array
{
  $elements = array_merge([],
  [
  "jmeno" => new InputElement("jmeno", "Jméno"),
  "prijmeni" => new InputElement("prijmeni", "ŕPíjmení"),
  "email" => new InputElement("email", "Email"),
  "telefon" => new InputElement(…),
  "ic-checkbox" => new CheckboxElement(…),
  ],
  $this->icoForm(),
…
```

Blok kódu 4.10: Funkce elements() třídy KontaktComponent

V příkladu této funkce je viditelná definice pole, které obsahuje elementy pro formulář kontaktů. Dále je možné vidět využití funkce `icoForm()`, která reprezentuje další část formuláře. Důvod využití funkce namísto definice samostatných elementů vyplývá ze samotné funkce uvedené v bloku 4.11. Kód je pro přehlednost zkrácen.

Funkce `icoForm()` totiž pomocí objektu třídy SessionData uchovaném v proměnné `$dataProvider` zjišťuje, jestli je komponenta ve stavu, kdy má zobrazovat část formuláře pro zadání IČO. Pokud tedy má daný formulář zobrazit, funkce vrací pole s jeho elementy. Pokud podmínka není splněná, vrací funkce prázdné pole.

Tímto využitím funkcí je dosaženo toho, že lze formulář sestavovat i ze skupin elementů, jejichž zobrazení je podmíněno stavem či hodnotou dat komponenty nebo jiného objektu.

```
    private function icoForm(): array
    {
      if ($this->dataProvider->getState("show-ico-form")) {
        return [
        "message" => new ParagraphElement(…),
        "ico" => new InputElement(…)
        ];
      }
      return [];
  }
```

Blok kódu 4.11: Funkce icoForm() třídy KontaktComponent

Logika komponent je komplexnější než u předchozích, menších tříd elementů. I proto jsou pro komponenty připraveny abstraktní třídy, které zásadní část této logiky skrývají. Jádro komponent představuje třída ComponentAbstract uložená na cestě `/frontend/components/abstract`.

```
    protected function frontend(…): Response
    {
      $body = <<< EOB
      <form id="{$id}" {$htmlComponentAttributes}>
      <h2>{$this->heading}</h2>
      <div class="description">{$this->description}</div>
      {$htmlElements}
      </form>
      EOB;
      return new Response($this->getHeaders(), $body);
  }
```

Blok kódu 4.12: Funkce frontend(...) třídy FormComponentAbstract

V bloku 4.12 je zobrazena implementace funkce frontend(...) třídou FormComponentAbstract. Tato třída tak poskytuje HTML kód pro rozšiřovanou třídu ComponentAbstract, jež provádí sestavení celého HTML kódu. Proměnná `$htmlElements` zastupuje veškeré elementy, které se mají v příslušné komponentě zobrazit.

4.3.4 Layouty

K zobrazení elementů na stránce již existují třídy s potřebnou logikou. Nyní je třeba nastavit pořadí, jakým se budou na stránce zobrazovat. Tuto funkci zastávají objekty tzv. layoutů.

Layouty jsou poměrně jednoduché objekty a není pro jejich funkčnost nutná složitá výpočetní logika. Jejich úloha spočívá v uložení pořadí komponent. Když třída (běžně Controller) potřebuje získat komponenty v určitém pořadí, objekty Layout jsou schopny vykreslit seřazené komponenty a vrátit již sestavenou HTML stránku.

4.4 Struktura back-end části

Zpracování veškerých HTTP dotazů na aplikaci probíhá v BE sekci aplikace, na cestě `/backend`. Objekty v této části mají za úkol zprostředkovat komunikační kanál mezi uživatelem a objekty FE, datové zdroje těchto objektů a další obslužné třídy.

Vstupním bodem celé aplikace je soubor Controller.php umístěný na cestě `/backend/controllers`. Tato třída slouží jako jednoduchý dynamický router, který vyřizuje dotazy podle cesty a metody dotazu. Jsou zde definované i statické cesty k jiným souborům souvisejících s grafickou podobou stránky.

4.5 Architektura frameworku

Při tvorbě kostry aplikace bylo záměrem naprogramování jednoduchého systému, jenž bude sestavovat web pomocí předdefinovaných tříd. Tyto třídy se pak podle jejich stavu a případně stavu ostatních tříd dokáží měnit a zobrazovat. Zároveň bude možné pomocí aplikace obsluhovat uživatelské interakce a vstupy.

Aplikace poskytuje veškeré základní funkce potřebné k sestavení webu ovládaného pomocí REST dotazů, který je schopen reagovat na interakce uživatele s elementy bez JS kódu. Tato kapitola popisuje výslednou podobu frameworku a jeho funkční principy.

4.5.1 Zobrazení stránky

Prvním zásadním úkolem aplikace je zobrazit celý obsah stránky při zaslání dotazu na server. Jelikož je HTMX schopné měnit obsah bez opakovaného načítání stránky, je možné vyvíjet webovou aplikaci typu „single page“. O SPA pojednává blíže kapitola 2.1. Stránka bude tedy zobrazována vždy na jedné adrese, aplikace ji zobrazuje na cestě „/“, na výchozí adrese serveru.

Po zaslání dotazu na server se ve třídě Controller vyhodnotí odpověď pro požadovanou cestu a metodu dotazu. Při volání výchozí adresy serveru je třeba vykreslit celou stránku. Ta je definovaná v Layoutu jako pole komponent, které se postupně vykreslují. Ve třídě Controller tuto funkci zastává část kódu uvedená v bloku 4.13.

Layout je možné získat pomocí repozitáře LayoutRepository. Ten následně pomocí funkce `show(…)` vykreslí všechny komponenty a vrátí odpověď jako objekt Response. Funkce `getBody()` pak z této třídy získá HTML kód odpovědi a uloží ho do proměnné.

```
switch ($path) {
  case "/":
  {
    $layoutRepository = LayoutRepository::getInstance();
    $layout = $layoutRepository::getRootLayout();

    $body = $layout->show($_GET)->getBody();
    require "frontend/index.phtml";
    break;
  }
  …
```

Blok kódu 4.13: Využití třídy Layout pro zobrazení stránky

Tato proměnná je následně zobrazena v souboru `index.phtml`. Ukázková aplikace tak stránku vrácenou Layoutem ještě obaluje do základní kostry HTML stránky. V produkční aplikaci by tuto kostru nahrazovala stránka e-shopu.

4.5.2 Routování tříd

Další nemálo důležitou úlohou aplikace je zobrazování a aktualizace částí obsahu stránky. To umožňují HTTP dotazy na prefix (základní adresu) routovatelných objektů, přičemž odpovědí dotazu by měla být běžně nějaká ucelená část aplikace. Aplikace technicky umožňuje zobrazení v podstatě jakékoliv třídy, a to pomocí implementace rozhraní RoutableInterface a rozhraní zpracovatelné metody HTTP dotazu.

Záměr je takový, že by routovatelný objekt měla být ucelená část aplikace, kterou dává smysl aktualizovat. Běžně tedy není využito routování elementů, jelikož by se při aktualizaci stránky musel odesílat HTTP dotaz pro každý element zvlášť.

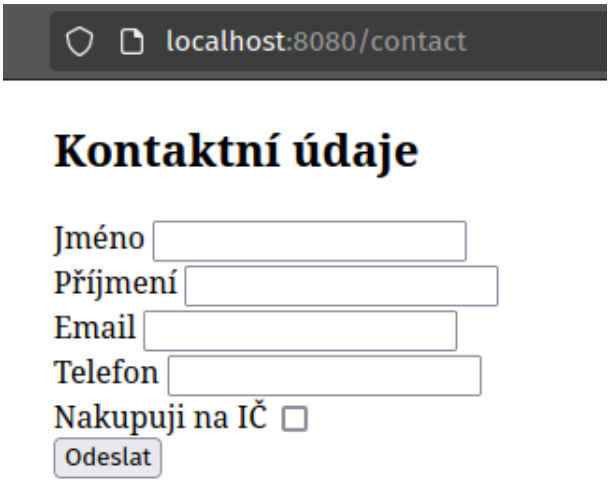
Největším routovatelným objektem je celá stránka vykreslovaná layoutem, jak popisuje kapitola 4.5.1. Dalším celkem, který dává smysl aktualizovat, jsou komponenty. Ty je potřeba aktualizovat po odeslání formuláře, při interakcí s ovládacími prvky, nebo po změně jejich stavu.

To je umožněno tím, že jsou komponenty, resp. jejich abstrakt, routovatelné. Každá komponenta má tedy vlastní **prefix**. Na tomto prefixu je možné třídu zavolat a při využití rozhraní GetInterface zobrazit, pomocí PostInterface pak vracet hodnoty formuláře serveru. Více informací o rozhraní RoutableInterface je dostupných v kapitole 5.1.6.

V praxi to znamená, že každá komponenta je samostatně zobrazitelná pomocí HTTP dotazu. Je tedy možné získat samotný HTML kód komponenty zasláním dotazu na adresu jejího prefixu.

Tato funkce spolu s HTMX poskytuje formu náhrady za JavaScript kód, který by aktualizoval prvky stránky na straně uživatele přímo v prohlížeči. V případě této aplikace je stav tříd měněn pomocí dotazů na server, který v reakci na tuto změnu znovu sestaví elementy třídy a odešle je jako odpověď na HTTP dotaz.

Podrobnosti o třídách komponent jsou vypsány v kapitole 5.4.



Obr. 4.2: Dotaz na komponentu s prefixem „contact“

4.5.3 Akce komponenty

Další funkcí komponent jsou tzv. akce. Ty podporují spouštění vnitřních funkcí komponenty, a to opět pomocí HTTP dotazů.

Motivace je následující. Na zobrazované stránce uživatel provede akci, která se v aplikaci promítne změnou stavu nebo vzhledu komponenty a jejích elementů. Je tedy třeba ošetřit jak manipulaci se stránkou ze strany uživatele, tak i zpracování akce na straně serveru.

V současných aplikacích se tyto akce běžně ošetřují na straně uživatele pomocí JavaScriptového kódu, přesněji pomocí event listenerů. Ty „poslouchají“ události spuštěné na stránce a při zachycení požadované události spustí obsluhující funkci (Nguyen a Pehlivanian, 2021). Programovaná aplikace se však snaží tradiční kód JS nahradit využitím knihovny HTMX, je proto potřeba tyto události ošetřit jinak.

K tomuto účelu slouží již zmiňované akce. Ty fungují následovně: abstrakt komponenty poskytuje veřejnou funkci setAction(...), která přijímá dva parametry. Těmi jsou cesta akce a funkce, která se spustí při odeslání dotazu na adresu. V komponentě je při nastavení možné tuto funkci zavolat, ta parametry zpracuje tak, že pro poskytnutou cestu zaregistruje její akci. Více informací o abstraktu komponent poskytuje kapitola 5.4.1.

Pro příklad registrace akce lze uvést objekt KontaktComponent, který při svém nastavení pomocí funkce setAction registruje funkci setIcFormVisible na cestu „ic--select“.

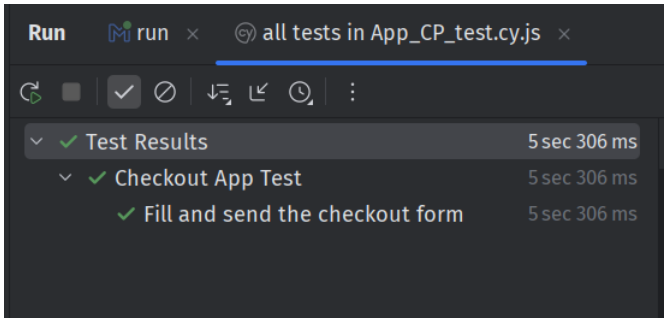
Zaregistrovanou akci lze následně spustit tak, že se odešle HTTP dotaz na server s následující cestou: „`server:port/prefix/cesta/akce`“. Server a port tvoří adresu serveru, prefix je cesta komponenty (popsána také v kapitole 4.5.2). Zbytek cesty je už jméno akce, která má být dotazem spuštěna.

V případě již zmíněné třídy KontaktComponent je možné spustit akci „ic--select“ odesláním HTTP dotazu na následující adresu spuštěného PHP serveru: „`server:port/contact/ic-select`“. Server a běžícího je opět adresa serveru aplikace, „contact“ je prefix komponenty a „ic-select“ je název spouštěné akce.

4.6 Testování aplikace

Testování je důležitou součástí vývojového procesu, jelikož při úpravách aplikace může vzniknout v kódu chyba. Zároveň je žádoucí testy provádět pravidelně pro předejití složitému opravování chyb, které se za sebou řetězí.

Dnes existuje mnoho různých nástrojů, a přístupů testování. Lze je podle rozsahu rozdělit na mikro a makro přístupy. Mikro testovacím přístupem je například tzv. „unit test“, který testuje nejmenší část aplikace, například třídu. Další přístupy testování mohou být integrační testy ověřující správné sestavení aplikace, testy služeb, funkční testy uživatelského rozhraní a makro end-to-end (dále E2E) testy. (Mohan, 2022)

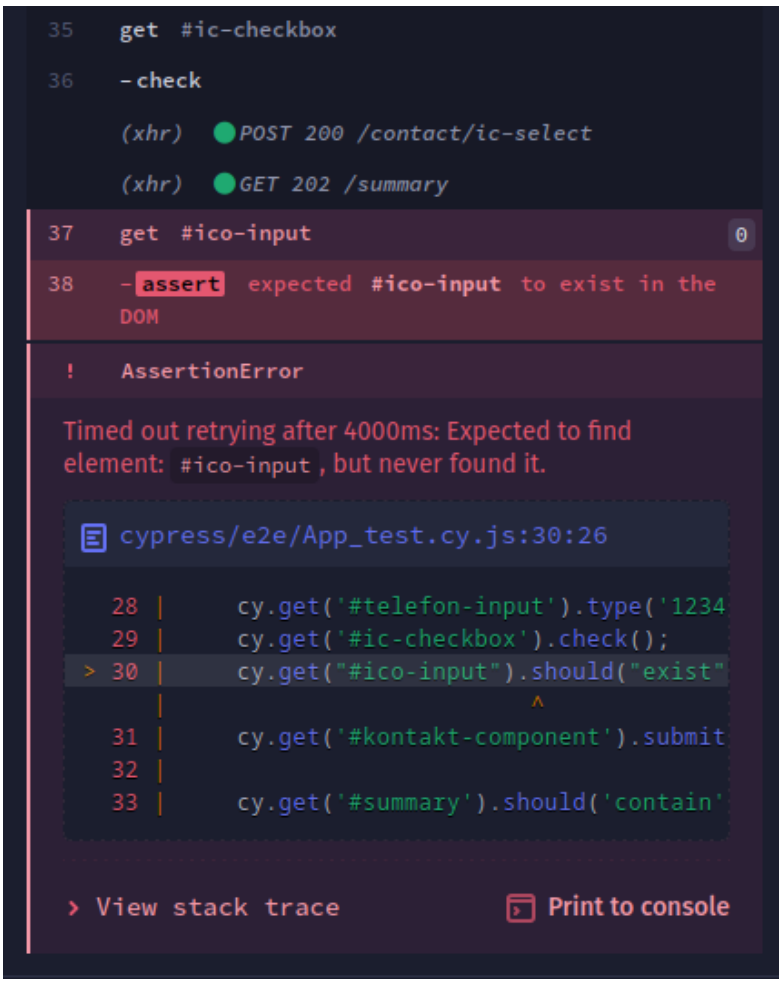


Obr. 4.3: Úspěšný test v prostředí PhpStorm

V případě vyvíjené aplikace jsou využity E2E testy, jelikož dokáží pokrýt otestování celé aplikace pomocí jednoho uceleného testu. To je samozřejmě možné díky velikosti aplikace.

Pro testování je využit nástroje Cypress, který zprostředkovává automatizované testování uvnitř prohlížeče pomocí direktiv zapsaných v jazyce JavaScript, vydaný v roce 2014. Nástroj poskytuje grafické i textové uživatelské rozhraní pro běh testů. (Mohan, 2022)

Nástroj Cypress se nachází na cestě `/tests_e2e/cypress`. Specifikace testů pak v adresáři `/tests_e2e/cypress/e2e`. Zde byl vytvořen test v souboru `App_test.cy.js`, který do formuláře s výběrem doručením „do ruky“ vyplňuje testovací data a kontroluje, jestli je server zpracovává a zobrazuje požadované formuláře.



Obr. 4.4: Neúspěšný test v grafickém prostředí Cypress

Neúspěšný test lze demonstrovat pomocí vypnutí akce „`/contact/ic-select`“. V tomto případě test očekává zobrazení pole pro vyplnění IČ, které se ale nikdy nevykreslí. Cypress po krátké chvilce vrátí chybu s hláškou, že nemohl najít požadovaný element formuláře. Případ neúspěšného testu zobrazuje obrázek č. 4.4.

4.7 Dokončená aplikace

Výsledkem vývoje je funkční aplikace obalená v základním HTML těle pro prezentační účely. Aplikace poskytuje 3 formuláře dle zadání a souhrn zadaných dat. Data jsou zobrazena jako název pole a jeho hodnota.

Aplikace po imaginárním odeslání dat dokáže zobrazit stránku se souhrnem dat a možností tato data stáhnout ve formě JSON. Na této stránce je opět viditelná komponenta souhrnu, což demonstruje opětovné využití komponenty. Komponenta pro toto zobrazení nesmí mít viditelné tlačítko k odeslání dat.

Mobilní verze upravuje jednotlivé komponenty tak, aby byly lépe dostupné na dotykových zařízeních. Vzdálenost mezi elementy je zvětšena a nadpisy textových polí jsou umístěny nad samotnými vstupy.

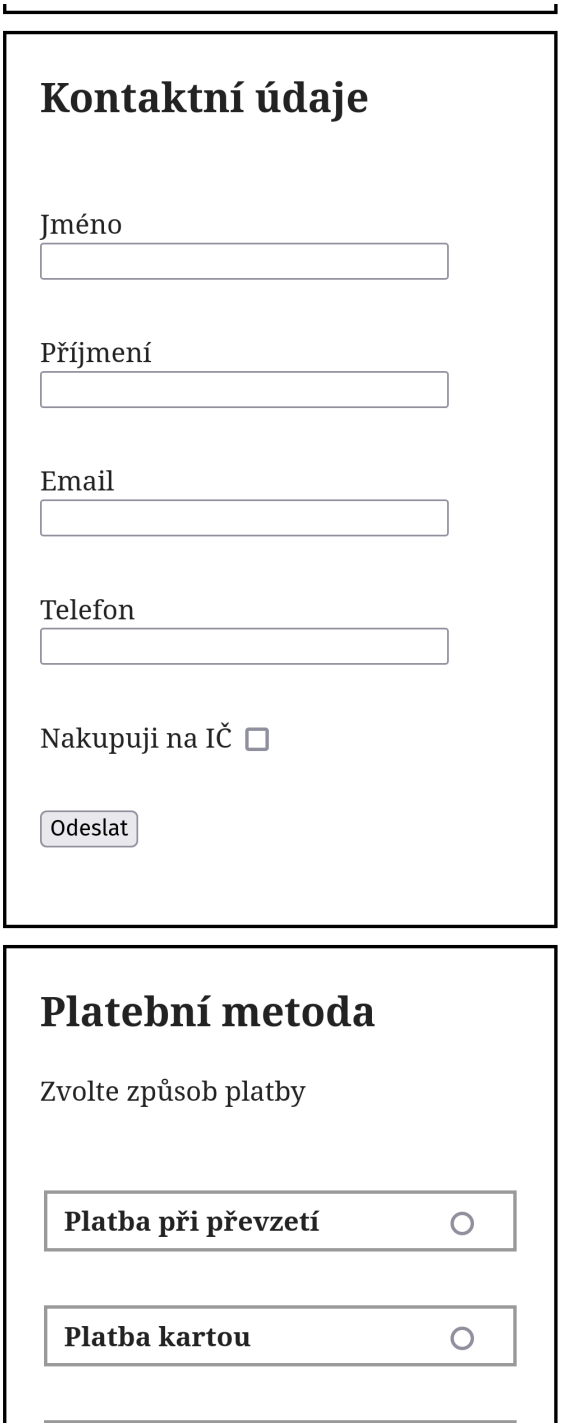
Volba metody dodání zobrazuje dle zaškrtnutí volby rozšířený formulář s dodatečnými poli potřebnými pro zvolenou dodací metodu. Tuto část obsluhuje využitá knihovna HTMX. Změny jsou prováděny dynamicky a vždy jen v dané komponentě, stránka se tedy znovu nenačítá.

Data vyplněných formulářů se po odeslání zaznamenávají pomocí třídy SessionData. Výčet těchto dat pak poskytuje komponenta SouhrnComponent. Data je možné pomocí tlačítka „odeslat“. Tato funkce ve skutečnosti není implementována a po zmáčknutí tlačítka se pouze zobrazí jiný layout s konceptem stránky, která oznamuje úspěšné odeslání objednávky.

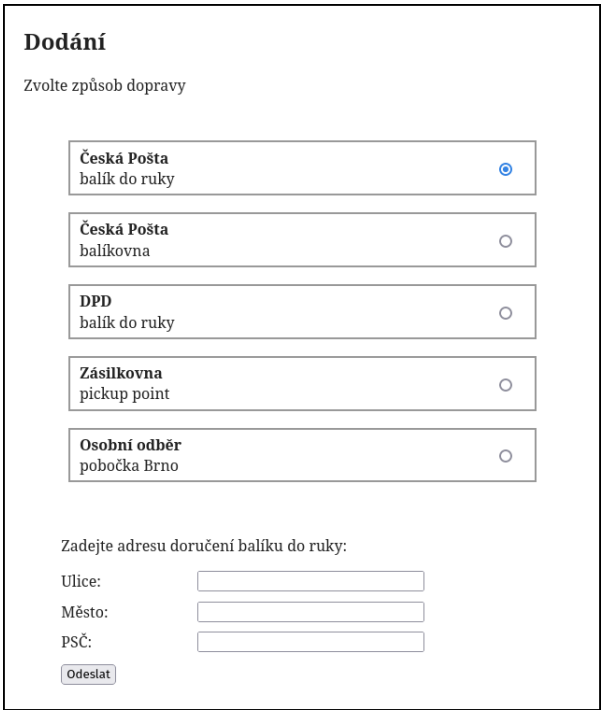
Na stránce zobrazující úspěšné odeslání a zadaná data je také možné data souhrnu ze serveru stáhnout ve formátu JSON.

Následující stránky zachycují FE aplikace v různých stavech.

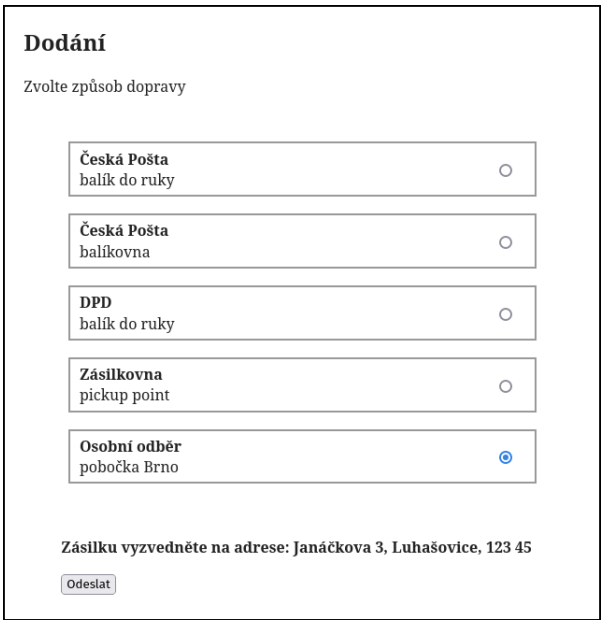
Obr. 4.5: Nevyplněná stránka aplikace



Obr. 4.6: Zobrazení stránky na mobilním zařízení

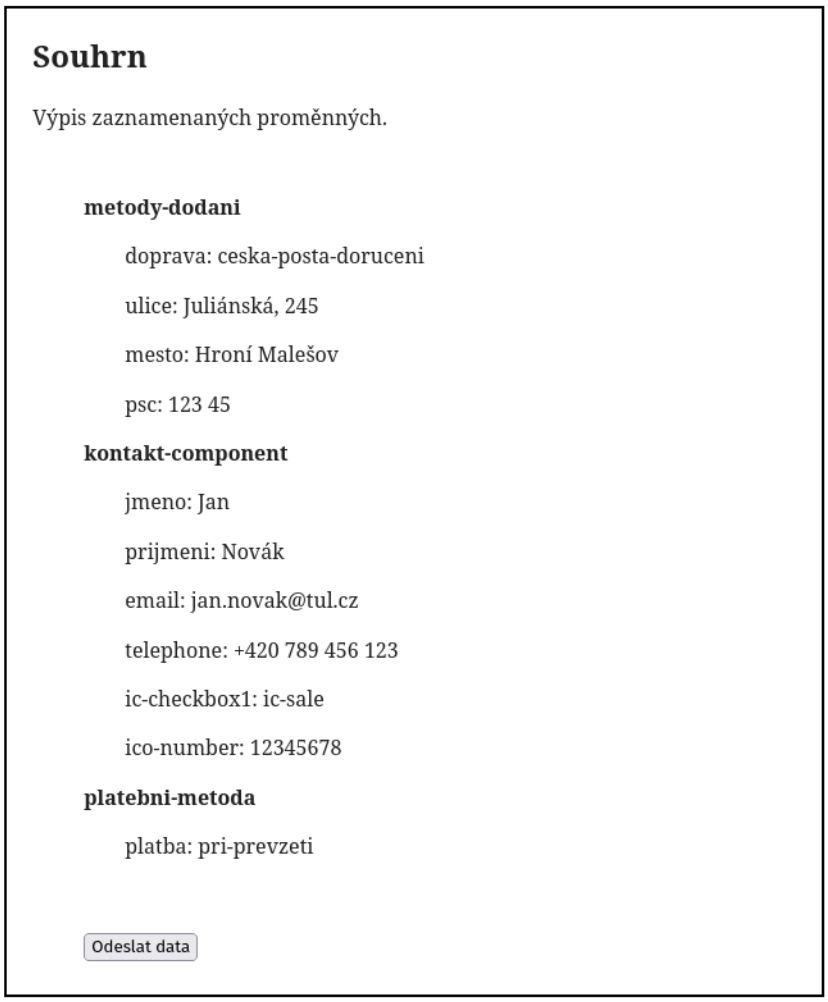


Obr. 4.7: Formulář pro dodání českou poštou



Obr. 4.8: Formulář při volbě osobního odběru

Obr. 4.9: Vyplněný formulář stránky



Obr. 4.10: Souhrn dat formulářů

Úspěch!

Objednávka č. 1714755400 úspěšně uložena!.
Můžete pokračovat na další stránku.

Pokračovat

Souhrn

Výpis zaznamenaných proměnných.

metody-dodani

doprava: ceska-posta-doruceni

ulice: Juliánská, 245

mesto: Hroní Malešov

psc: 123 45

kontakt-component

jmeno: Jan

prijmeni: Novák

email: jan.novak@tul.cz

telephone: +420 789 456 123

ic-checkbox1: ic-sale

ico-number: 12345678

platebni-metoda

platba: pri-prevzeti

success

order-number: 1714755400

Pro stažení dat objednávky stiskněte tlačítko níže.

Stáhnout

Obr. 4.11: Stránka po odeslání zásilky

Exportovaná data této pokladny by vypadala následovně:

```
{
  "metody-dodani":{
    "doprava":"ceska-posta-doruceni",
    "ulice":"Juliánská, 245 ",
    "mesto":"Hroní Malešov",
    "psc":"123 45"
  },

  "kontakt-component":{
    "jmeno":"Jan",
    "prijmeni":"Novák",
    "email":"jan.novak@tul.cz",
    "telephone":"+420 789 456 123",
    "ic-checkbox1":"ic-sale",
    "ico-number":"12345678"
  },

  "platebni-metoda":{
    "platba":"pri-prevzeti"
  }
}
```

Blok kódu 4.14: Exportovaná data z pokladny ve formátu JSON

5 Dokumentace objektů

V této kapitole jsou popsány stěžejní objekty aplikace a jejich funkce. Záměrem kapitoly je poskytnutí podrobného popisu fungování těchto objektů.

Objekty v této kapitole nejsou seskupeny podle adresářů, ale jsou seřazeny podle typu.

5.1 Rozhraní

Rozhraní třídám určují funkce, pro které musí vytvořit implementaci. Třídy pak mohou být zpracovávány podle funkcí implementovaných rozhraní, nikoliv pouze podle funkcí instance třídy.

Tato kapitola obsahuje abecedně seřazený výčet veškerých rozhraní aplikace.

5.1.1 ButtonInterface

Rozhraní udávající funkci tlačítka implementující třídě elementu. Jedná se o jediné prázdné rozhraní v aplikaci, přičemž rozšiřuje ShowableInterface. Při implementaci je tedy nutné do kódu přidat jen metody ShowableInterface.

5.1.2 DataProviderInterface

Toto rozhraní je určeno pro DataProvidery. Objektům využívajícím rozhraní nařizuje implementaci dvou funkcí.

Funkce **`setData($key, $value)`** slouží k zaznamenání dat uložených v proměnné `$value` do úložiště pomocí klíče `$key`.

Tato data jsou dostupná pomocí funkce **`getData($key)`**, která opět podle klíče `$key` vrátí zaznamenanou hodnotu.

5.1.3 GetInterface

Jedno z rozhraní definující metodu HTTP dotazu, kterou je schopna routovatelná třída zpracovat, je GetInterface.

Jedinou definovanou funkcí je funkce **`show($get, $actionUrl = null)`**. Tato metoda má být volána Controllerem ve třídě, která umí zpracovat metodu dotazu na dané adrese.

Parametr `$get` zastává data ze superglobální proměnné `$_GET`, tedy data předaná HTTP dotazem. Controller tato data předává funkci `show()` pro další zpracování.

Druhý parametr `$actionUrl = null` zastává adresu akce, která má být třídou spuštěna. Při volání nemusí být tento parametr uveden, jelikož má výchozí hodnotu null. Pro více informací o akcích viz k. 4.5.3.

5.1.4 InputInterface

Základní textový vstupní element definuje rozhraní InputInterface. Vstup je obsluhován metodami **`setValue($value)`** a **`getValue()`**. Tyto metody zajišťují nastavení a získání textového obsahu pole.

5.1.5 PostInterface

Jak již z názvu vyplývá, toto rozhraní umožňuje routovatelným třídám zpracovávat dotazy s metodou post. To je možné díky funkci **`post($post, $actionUrl = null)`**, která v parametrech získává data z HTTP dotazu pomocí proměnné `$post`. Parametr `$actionUrl = null` může poskytovat adresu akce, která má být pomocí HTTP dotazu spuštěna.

5.1.6 RoutableInterface

Rozhraní vytyčující funkce routovatelné třídy. Takovou třídu je možné zavolat pomocí HTTP dotazu a ona na tento dotaz reaguje, podle využité metody dotazu, požadovanou funkcí.

Pro možnost zavolání takové třídy je tedy nezbytná implementace rozhraní metody dotazu. V případě ukázkové aplikace je tedy pro routovatelnou třídu nutná implementace rozhraní GetInterface, PostInterface nebo obou. V případě, kdy aplikace obsahuje více rozhraní metod HTTP dotazu, není počet implementovaných rozhraní v jedné třídě omezen. Může tedy implementovat všechna dostupná rozhraní metod HTTP dotazu.

Toto rozhraní definuje funkce, které umožňují zpracování adres routovatelné třídy a přidání akce třídy. Níže jsou rozebrány funkce podrobně.

Funkce **`isRouteTo($route)`** vrací hodnotu boolean v závislosti na tom, jestli je třída implementující toto rozhraní dostupná na adrese uvedené v parametru `$route`. S adresami je spojená i funkce **`getRoutePrefix()`**, která vrací hodnotu string s prefixem, tedy kořenovou adresou routovatelné třídy. Funkce **`getRoutes()`** vrací veškeré adresy, na kterých je možné implementující třídu zavolat.

Poslední funkcí tohoto rozhraní je **`setAction($route, callable $callback)`**, která umožňuje implementující třídě přidat volatelné akce. Parametr `$route` je adresou, na které bude akce dostupná pro HTTP dotazy. Spustitelný objekt `callable $callback` pak představuje callback funkci, která bude spuštěná při zpracování HTTP dotazu odeslaného na adresu akce.

5.1.7 SelectableInterface

Rozhraní elementů typu input, které lze vybrat nebo zaškrtnout. Funkce **`setChecked(bool $checked)`** umožňuje změnu stavu zaškrtnutí pole.

Dalším specifikem těchto polí je skupina, která pole sdružuje. K získání názvu skupiny slouží funkce **`getGroupName()`**.

Poslední vlastností těchto vstupních elementů jsou hodnoty, které se předávají při odeslání pole. Pro získání této hodnoty v kódu je v rozhraní definovaná funkce **`getValue()`**.

5.1.8 ShowableInterface

Třídy implementující toto rozhraní jsou pro aplikaci potenciálně zobrazitelné, tedy převeditelné do HTML kódu.

Funkce rozhraní **show()** pak umožňuje samotné zobrazení třídy. Požadovaným výstupem této funkce je HTML kód vrácený jako objekt typu string.

Rozhraní dále poskytuje funkci **getName()**, která slouží k získání „jména“, resp. identifikátoru zobrazitelné třídy.

5.2 Traity

Traity jsou struktury podobné třídám, ale na rozdíl od třídy nemůže být vytvořena jejich instance. Jediný způsob, kterým mohou být využity, je jejich zakomponování do třídy pomocí klíčového slova `use`. Svým způsobem traity obchází jednonásobnou dědičnost tak, že třídy mohou využívat jejich implementace nezávisle na jejich rodičovských třídách. (Zandstra, 2021)

Aplikace na cestě `/frontend/elements/traits` obsahuje celkem tři traity, využity jsou pouze dva z nich, a to AttributesTrait a HttpHeadersTrait. Je zde připraven ještě ErrorTrait, ten ale není využit a slouží jako příprava pro obsluhování chyb elementů.

5.2.1 AttributesTrait

Tento trait je využíván jako implementace metod pro práci s atributy elementů. Poskytuje proměnnou `$attributes`, která ukládá atributy elementů. Parametry jsou v proměnné uloženy stylem asociativního pole, kdy klíč pole reprezentuje název atributu a hodnota pole je hodnota atributu. Dále pak trait poskytuje 4 následující obslužné funkce.

setAttributes(array $attributes)

Pole předané parametrem je po zavolání funkce sloučeno s existující proměnnou atributů. Používá se tak pro nastavování atributů elementů.

removeAttribute($attribute)

Po zavolání funkce kontroluje, jestli je v poli s uloženými atributy definován atribut s klíčem $attribute. Pokud ano, je tento záznam z úložiště atributů vymazán.

getHtmlAttributes()

Prochází všechny záznamy uložených atributů a převádí je do textové podoby, kterou je možné vložit přímo do HTML kódu.

getAttributesArray()

Getter vracející pole s uloženými atributy elementů.

5.2.2 HttpHeadersTrait

Pro správu hlaviček odesílaných při zobrazování komponent slouží trait HttpHeadersTrait. Ten definuje proměnnou `$headers`, která pomocí funkce `setHeaders(…)` ukládá hlavičky pro odeslání. Dále poskytuje funkci `getHeaders()`, která vrací pole těchto hlaviček.

5.3 Elementy

Třídy elementů slouží jako objektová reprezentace HTML kódu a přímo zastupují jednotlivé elementy, nikde pro ně tedy nejsou specifikované HTML šablony. Znamená to, že pro element, který se na stránce často opakuje, je vhodné vytvořit vlastní třídu, kterou je možné dále využít pro sestavování komponent.

Jasnou funkcí tříd elementů je jejich zobrazení. Je tedy nutné, aby každá třída implementovala rozhraní ShowableInterface (viz k. 5.1.8), která vytyčuje tyto funkce.

Dalším předpokladem k vykreslení elementu je jeho HTML kostra. Ve třídách elementů se tyto kostry běžně definují ve funkci `show()`, mohou se ale vytvářet i v jiných funkcích, které kostru vrací. Kostra se při zobrazení musí naplnit potřebnými proměnnými a stává se tak plnohodnotným elementem. Zmiňovaná funkce `show()` vrací konečnou podobu HTML kódu elementu a musí být implementována.

Příkladem může být zkrácený kód funkce `show()` třídy ButtonElement uvedený v bloku 5.1. Zde kostru HTML elementu tvoří div s parametrem class, do kterého se vkládá po boku základní třídy HTML elementu i identifikátor tlačítka. Dále kostra obsahuje samotné tlačítko s dynamicky přidanými parametry a popisem. Před ukončením elementu div je ještě vyhrazené místo pro případné chyby.

```
$body = <<< EOL
<div class="simple-button {$this->id}">
<button {$htmlAttributes}>{$this->label}</button>
{$error}
</div>
EOL;

return $body;
```

Blok kódu 5.1: Sestavení HTML kódu třídy ButtonElements

Veškeré elementy nějakým způsobem pracují s HTML atributy. Kvůli stejnému principu u všech tříd elementů byl vytvořen AttributesTrait, který poskytuje metodu pro správu atributů elementů.

Tato konstrukce je u všech elementů podobná, liší se pouze kostry a obslužné metody. Elementy implementují vlastní rozhraní, ta jsou popsána v kapitole 5.1.

5.4 Komponenty

Komponenty jsou stavebními prvky celé aplikace a zároveň také pravděpodobně nejkomplexnější třídy kódu. Základem všech komponent je třída `ComponentAbstract`, jež je popsána v kapitole 5.4.1.

Základní třídu rozšiřují abstraktní třídy, které definují obalový element komponenty. V případě této aplikace se jedná o `Form` a `Div` abstrakty komponent.

Komponenty, které nevyužívají HTMX pro aktualizaci obsahu, pouze implementují abstraktní funkce popsané v kapitole 5.4.1. V opačném případě je ve funkci `elements()` využita další libovolná funkce, která podle vlastní logiky zobrazuje další elementy. K tomu mohou být využity i akce komponenty (viz 4.5.3).

V následujících kapitolách je popsána abstraktní třída ComponentAbstract a specifické vlastnosti některých komponent.

5.4.1 ComponentAbstract

Abstraktní třída ComopnentAbstract obsahuje veškerou logiku, která obsluhuje nastavení komponent, vykreslování elementů a další. Lze ji proto označit za pomyslné srdce všech komponent. Samotné třídy komponent tak nejsou touto logikou zatíženy a mohou obsahovat pouze definici HTML kódu a akcí (viz kapitola 4.5.3).

Některé funkce definují rozhraní. Třída implementuje tři následující: RoutableInterface, PostInterface, GetInterface. Jejich popis je dostupný v kapitole 5.1.

Abstraktní třída definuje abstraktní funkce, které slouží k předání dat z rozšiřující třídy tomuto abstraktu. Většina těchto abstraktních funkcí je využita při vytváření objektu třídy ComponentAbstract a dochází tak k naplnění třídy daty komponenty. Definovány jsou i vlastní funkce. V následujících odstavcích je vysvětlen jejich význam.

buildElements()

Třídy komponent pomocí funkce elements(), která je popsána dále v textu, určují abstraktu, jaké elementy má komponenta zobrazit. Tyto vracené elementy jsou ale závislé na stavu komponenty, který je uložený v modelu (viz 2.2). Proto je při každém zobrazení třeba aktualizovat pole elementů k zobrazení. To zajišťuje tato funkce.

dataProvider()

Abstraktní funkce sloužící jako poskytovatel objektu DataProvider. Jelikož jsou objekty SessionData vázané na ID komponenty, musí je vytvořit samotná komponenta. Tato funkce objekty předává abstraktu, který s nimi pracuje.

elements()

Třídy komponent vytváří pole elementů, které se v komponentě mají uživateli zobrazit. Funkce `elements()` pole poskytuje abstraktu, který toto pole umí

zpracovat a vrátit HTML kód veškerých vnitřních elementů. Platí však, že abstrakt musí mít definované akce pro zpracování všech typů elementů.

frontend(...)

Funkce je určena pro využití v abstraktních třídách, které definují obalový element celých komponent. V případě aplikace tedy DivComponentAbstract a FormComponentAbstract. Pomocí funkce třídě ComponentAbstract vrací podobu obalového HTML kódu. Funkce přijímá parametry uvádějící identifikátor, atributy a elementy komponenty.

id()

Pomocí této funkce získává abstraktní třída ID komponenty z konečných tříd komponent.

routePrefix()

Tato funkce vrací prefix, který konečná třída komponenty definuje pro její routování.

setup()

Funkce spouštěná ihned po vytvoření objektu. Tato třída slouží jako jakýsi „konfigurátor“, ve kterém je možné provádět další nastavení abstraktní třídy komponent. Běžně je tato funkce využita pro nastavení nadpisů, popisků, akcí a odesílaných HTTP hlaviček komponenty.

5.4.2 DopravaComponent

Komponenta DopravaComponent je specifická tím, že pro výběr každé dodací metody je třeba zobrazit další specifickou část formuláře pro danou metodu. Funkcionalitu zajišťuje HTMX dotaz odeslaný na prefix komponenty.

Jelikož se při této události změní zaznamenaná hodnota skupiny radio elementů, lze podle této hodnoty změnit i podobu vraceného formuláře. To zajišťuje funkce getAdditionalForms() třídy, která pomocí bloku switch-case dokáže k hodnotě

zvoleného radio elementu vrátit požadovaný formulář. Tuto funkcionalitu zobrazují obrázky 4.7 a 4.8.

5.4.3 KontaktComponent

Specifikem komponenty KontaktComponent je zobrazování pole pro vyplnění IČ subjektu pomocí checkbox elementu. Zde je využit odlišný přístup od třídy DopravaComponent.

V elementech komponenty je definován checkbox „ic-checkbox“, který po události „click“ pomocí HTMX odesílá dotaz na akci `/contact/ic-select`. Třída pro tuto akci spouští funkci `setIcFormVisible(…)`, která podle hodnoty checkboxu nastaví stav `show-ico-form` na `true` nebo `false`.

Zobrazení dodatečného formuláře zajišťuje funkce `icoForm()` ve funkci `elements()`. Ta podle stavu zaznamenaného akcí buďto element zobrazí nebo nezobrazí.

5.4.4 SouhrnComponent

Tato velmi specifická komponenta reaguje na HTTP hlavičku s hodnotou `HX-Trigger: form-changed`. Po zachycení hlavičky se komponenta znovu vykresluje s tím, že se aktualizují data třídy SessionData. Je tak možné po odeslání kterékoliv komponenty okamžitě aktualizovat zobrazovaný souhrn.

5.5 Layouty

Layouty poskytují pouze 2 základní funkce. Komponenty layoutu vrací funkce `getComponents()` a objekt Response, který zobrazuje Controller, vrací funkce `show(…)`.

Odpověď pro controller je sestavována tak, že se zavolá funkce `show(…)` každé komponenty a zapisuje se do proměnné. Tato proměnná se poté odesílá jako tělo odpovědi. Následně je toto tělo zobrazeno, v konceptu aplikace je ještě obaleno kostrou HTML stránky.

5.6 Controller

Třída Controller po spuštění zaznamená adresu dotazu, prefix komponenty a případnou cestu akce (viz 4.5.3). Získaná data jsou pak využita při spuštění funkce `process(…)`. Ta funguje jako router aplikace, kdy se pomocí bloku switch-case a poskytnuté cesty dotazu zpracovávají odpovědi. Nachází se zde také cesty pevně stanovené pro CSS styly a grafiku.

Dotaz na cestu „`/`“ je zpracován layouty, kdy se na dotaz odpovídá kódem celé stránky. Výchozí akce switch-case bloku funguje již jako dynamický router, který se snaží nalézt komponentu, která by mohla být dotazem volána. Pokud sedí adresa a metoda dotazu, je spuštěna příslušná funkce komponenty.

5.7 SessionData

Třída SessionData v aplikaci zastává funkci modelu (viz 2.2). Poskytuje metody `setData(…)`, `unsetData(…)` a `getData(…)`, které všechny podle klíče a dalších parametrů obsluhují data komponent.

Další úlohou je zpracování dat stavu komponent. To plní podobným způsobem funkce `setState(…)` a `getState()`.

Třída také umožňuje získat data komponenty, všech komponent, nebo data komponenty určené identifikátorem. K tomuto účelu slouží funkce getDataArray.

6 Zhodnocení a doporučení

Vyvinutá aplikace je funkční a není složité ji spravovat. Díky využití MVC struktury pro vývoj aplikace je poměrně jednoduché upravovat a vytvářet nové vlastní komponenty a jejich elementy. Komponenty jsou pak jednoduše použitelné v třídách layoutů, které je následně vykreslí. Aplikace je tak flexibilní v tom, co uživateli prezentuje.

Dynamické upravování stránek pomocí knihovny HTMX se ukázalo jako velice jednoduché a intuitivní. Knihovna rozšiřuje základní využití HTML a umožňuje všem elementům odesílání dotazů, a tím i dynamické úpravy částí hypertextu mnoha různými akcemi. To je zásadní posun od využití `anchor` a `button` elementů v základním HTML.

Velikost knihovny HTMX je v porovnání s její funkcionalitou a následným využitím opravdu minimální. Není pravděpodobné, že by knihovna kdy mohla způsobit problémy při načítání stránky s ohledem na její velikost. Výhodou je i snadná instalace knihovny.

Otázkou také zůstává, jaký bude rozdíl v zátěži serveru v porovnání s běžnými FE frameworky a jestli je s takovou zátěží možné provozovat aplikace bez výrazného zvýšení nákladů na provoz. Většina frameworků zpracovává veškeré akce na straně uživatele pomocí JavaScriptu a server tak méně zatěžuji.Na druhou stranu při aktualizaci komponenty pomocí HTMX server odesílá jen její hypertextovou podobu a nemusí načítat žádné jiné soubory. Při kladném výsledku těchto testů by mohlo být využití této knihovny v produkčních aplikacích dobrou alternativou.

Integrace webových trackovacích nástrojů je jednoduchá, kdy požadovaný element může spustit akci, která bude trackování provádět. Stejně i napojení datových

modelů na jiná úložiště nebude díky využití jazyka PHP problém, jelikož jazyk podporuje širokou škálu databázových systémů.

Testování pomocí nástroje Cypress je jednoduché a rychlé. Využitím testovacího nástroje je možné zásadně zkrátit čas jinak manuálního testování. Dokumentace nástroje je velmi dobrá a není tak problém vytvářet základní testy ani pro programátora bez předchozích zkušeností s tímto nástrojem.

Aplikace jistě potřebuje pro nasazení do produkčního prostředí četná vylepšení. Za nejzásadnější autor považuje redesign nebo využití jiného dynamického routeru. Ten je ve stávající chvíli tvořen cyklem, který prochází pole objektů, které může zpracovat, a podmínkou kontroluje, jestli právě prvek poskytnutý cyklem má být zobrazen. Toto řešení routeru je neefektivní a pro větší aplikace nepoužitelné.

Dalšími objekty, které vyžadují práci jsou třídy layoutů. Tyto třídy nyní fungují jako specifický repozitář, který pouze ukládá komponenty a zároveň je umí vracet jako odpověď. Layouty by například mohly podporovat vlastní HTML kostry pro zobrazování stylizovaných stránek.

Obecně platí, že je třeba kód optimalizovat a ošetřit potenciální chyby v kódu. Další drobnou úpravu by si také zasloužily konstruktory elementů, ve kterých je například nutné uvést určitý počet parametrů před tím, než je programátor schopen definovat pole atributů.

Bylo by vhodné také namísto nadměrného využití abstraktních tříd do aplikace implementovat návrhový vzor typu `decorator`, který objektům dynamicky přidává funkcionalitu. Tento přístup je alternativou využití podtříd pro rozšiřování funkčnosti. (Freeman a Robson, 2020)

Závěr

Bakalářská práce se zaměřila na vývoj e-commerce hypermedia aplikace s využitím poměrně nové knihovny HTMX, pomocí které je možné využít k vývoji přístup orientovaný na hypermedia. Tato knihovna, i přes její zdánlivě jednoduchou funkcionalitu, zásadním způsobem zjednodušuje práci s REST API a dynamické aktualizace zobrazovaného obsahu.

Klíčovým cílem práce je dokázat, že lze vytvořit jednostránkovou aplikaci pomocí této jednoduché knihovny a že taková funkcionalita není podmíněna psaním JavaScriptového kódu nebo využitím složitých front-endových frameworků.

Knihovna poskytuje možnosti odesílání HTTP dotazů přímo v HTML, zaniká tedy potřeba využití dalšího kódu pro manipulaci s prezentovanou stránkou. Na rozdíl od běžných front-endových frameworků, které pro přenos dat využívají data ve formátu JSON a tato data dále zpracovávají na straně klienta, komunikuje HTMX se serverem pomocí hypermedií a ta přímo vkládají na stránku.

Výsledkem práce je úspěšné vytvoření funkčního základního frameworku a aplikace, která tento framework využívá pro zobrazení formulářů schopných samostatné aktualizace. Dále ze zkušeností s vývojem vyplývá to, že využití knihovny je vhodné pro vývoj e-commerce aplikací, které potřebují dynamicky měnit obsah a zároveň umožňovat napojení funkcí pro uživatelské akce.

Dále je také zřejmé, že vývoj zaměřený na hypermedia je dobře aplikovatelný na moderní aplikace. I přes skutečnost, že se při vývoji běžně preferuje přístup orientovaný na data, využití hypermedií je v kombinaci s knihovnou HTMX pohodlné a funkcionalitu JavaScriptového kódu v tomto případě dokáže bez problému nahradit.

Vyvinutá aplikace demonstruje, že se pomocí HTMX dá vytvořit aplikace, která je schopná efektivně manipulovat s daty a samotným hypertextem způsobem, který pro takovou funkcionalitu na straně klientské aplikace nepotřebuje další složitou strukturu kódu. Vynecháním takové struktury lze aplikaci zjednodušit a zpřehlednit.

Zároveň bylo zjištěno, že pro testování aplikací využívajících knihovnu HTMX lze bez problému využít stávající technologie, jako je, v tomto případě, nástroj Cypress.

Implementace aplikace na stávající webové stránky je také jednoduchá, kdy stačí na stránky jednoduše nainstalovat knihovnu a vložit první stránku aplikace. Dále je aplikace schopna samostatně fungovat.

Zároveň z výsledků práce vyvstávají některé otázky. Bylo by vhodné provést porovnání aplikací využívajících běžné FE frameworky s aplikacemi využívajících HTMX. Účelem je porovnání výkonnosti na stranách uživatele a klienta, kdy je žádoucí redukce zátěže na klientově straně, zároveň je ale žádoucí nenavyšování zátěže serverů obsluhujících tyto stránky.

Aplikace také potřebuje další úpravy kódu a implementace vhodnějších architektur kódu pro dosažení lepší přehlednosti a škálovatelnosti.

Seznam použité literatury

CASTRO, Elizabeth; HYSLOP, Bruce, 2022. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. 2. vydání. Přel. BAŠE, Ondřej; BAŠE, Kristýna. Brno: Computer Press. ISBN 978-80-251-5045-0. OCLC: 1329409132.

ECMA INTERNATIONAL, 2024. *ECMAScript® 2025 Language Specification* [online]. [cit. 2024-03-05]. Dostupné z: https://tc39.es/ecma262/.

FREE SOFTWARE FOUNDATION, 2023. *Make - GNU Project - Free Software Foundation* [GNU Make] [online]. [cit. 2024-04-24]. Dostupné z: https://www.gnu.org/software/make/.

FREEMAN, Eric; ROBSON, Elisabeth, 2020. *Head first Design Patterns*. Second edition. Bejiing Boston Farnham Sebastopol Tokyo: O'Reilly. ISBN 978-1-4920--7800-5.

GASSTON, Peter; BAŠE, Ondřej, 2015. *Moderní web*. 1. vydání. Brno: Computer Press. ISBN 978-80-251-4345-2. OCLC: 918007825.

GROSS, Carson; STEPINSKI, Adam; AKŞIMŞEK, Deniz, 2023. *Hypermedia Systems*. Ed. WILLIAM TALCOTT. Independently published. ISBN 979-8394025143.

IBM, 2024. *What is a REST API?* [online]. [visited on 2024-05-03]. Available from: https://www.ibm.com/topics/rest-apis.

JAN ŘEZÁČ, 2014. *Web ostrý jako břitva*. 1. vyd. BAROQUE PARTNERS s.r.o. ISBN 978-80-87923-01-6.

KUMAR, Tejas, 2024. *Fluent React*. First edition. Sebastopol, CA: O'Reilly Media, Inc. ISBN 978-1-09-813870-7. OCLC: 1416011697.

MANN, Eric, 2023. *PHP Cookbook: modern code solutions for professional PHP developers*. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly. ISBN 978-1-09-812132-7.

MICHÁLEK, Martin, 2017. *Vzhůru do (responzivního) webdesignu*. Verze 1.1. Praha: vlastním nákladem autora. ISBN 978-80-88253-00-6. OCLC: 1003286173.

MOHAN, Gayathri, 2022. *Full stack testing: a practical guide for delivering high quality software*. Sebastopol, CA: O'Reilly Media. ISBN 978-1-09-810813-7.

MOZILLA FOUNDATION, 2024a. *CSS: Cascading Style Sheets | MDN* [online]. [visited on 2024-04-23]. Available from: https://developer.mozilla.org/en-US/docs/Web/CSS.

MOZILLA FOUNDATION, 2024b. *SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [online]. [visited on 2024-05-03]. Available from: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

NGUYEN, Don; PEHLIVANIAN, Ara, 2021. *JavaScript Okamžitě*. 2. vydání. Přel. BAŠE, Ondřej. Computer Press. ISBN 978-80-251-5025-2.

SCOTT, Emmit A., 2016. *SPA design and architecture: understanding single-page web applications*. Shelter Island, NY: Manning. ISBN 978-1-61729-243-9. OCLC: ocn907182727.

SHUTE, Zachary, 2019. *Advanced JavaScript: Speed up Web Development with the Powerful Features and Benefits of JavaScript*. Birmingham: Packt Publishing Ltd. ISBN 978-1-78980-389-1. OCLC: 1085223719.

TATROE, Kevin; MACINTYRE, Peter, 2020. *Programming PHP: creating dynamic web pages*. Fourth edition. Beijing [China] ; Boston [MA]: O'Reilly. ISBN 978-1-4920--5413-9.

THE PHP GROUP, 2024. *PHP: Documentation* [online]. [cit. 2024-05-02]. Dostupné z: https://www.php.net/docs.php.

ZANDSTRA, Matt, 2021. *PHP 8 objects, patterns, and practice: mastering OO enhancements, design patterns, and essential development tools*. Sixth edition. New York, NY, U.S.A.: Apress. ISBN 978-1-4842-6790-5.